INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS – *CAMPUS* FORMIGA

ALUNO: GUILHERME BARBOZA MENDONÇA

ORIENTADOR: BRUNO FERREIRA

BLUE BUY: SISTEMA DE GERENCIAMENTO DE UMA LANCHONETE

Formiga – MG

08/10/2018

GUILHERME BARBOZA MENDONÇA

BLUE BUY: SISTEMA DE GERENCIAMENTO DE UMA LANCHONETE

Formiga – MG

08/10/2018

RESUMO

A tecnologia já se mostrou, para as empresas do atual mercado mundial, como uma aliada essencial para os negócios, atuando em diferentes contextos, desde o uso de uma inteligência artificial avançada, incumbida de analisar e minerar os dados de uma rede social, até um simples sistema de Ponto de Venda, mas igualmente útil em relação ao primeiro, de acordo com cada contexto. Esse documento visa dissertar sobre as etapas e métodos de desenvolvimento e produção do *software Blue Buy*, um sistema Web desenvolvido para auxiliar no gerenciamento de uma lanchonete ou restaurante. Para isso, são analisados, quantificados, avaliados e comentados os diversos materiais, ferramentas, tecnologias, métodos e *frameworks* envolvidos no desenvolvimento do sistema, discutindo ainda sobre as principais funcionalidades e premissas de execução do *software* final. Os principais resultados apresentados são os documentos de especificação e análise, além do próprio software. A partir desses resultados, conclui-se que é viável a criação de um sistema de informação que possa automatizar e desburocratizar o gerenciamento de empresas do ramo de venda de produtos.

Palavras-chave: Sistema de Ponto de Venda; Desenvolvimento Web, JSF, PrimeFaces;

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIAÇÕES

PDV - Ponto de venda

IFMG – Instituto Federal de Minas Gerais

CRUD - *Create, Read, Update* e *Delete* (Cadastrar, Visualizar, Atualizar e Deletar)

JSF – *JavaServer Faces*

Java EE **-** Java *Enterprise Edition*

HTTP - *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)

XHTML - EXtensible HyperText Markup Language (Linguagem de Marcação de Hipertexto extensível)

XML - *eXtensible Markup Language* (Linguagem de Marcação Extensível)

HTML – *Hypertext Markup Language* (Linguagem de Marcação de Hipertexto)

CSS - *Cascading Style Sheets* (Folha de Estilo em Cascatas)

EL - *Expression Language* (Linguagem de Expressão)

SO - Sistema Operacional

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

SGBD - Sistema Gerenciador de Bancos de Dados

ORM - *Object/Relational Mapping* (Mapeamento Objeto/Relacional)

CVS - *Concurrent Versions System* (Sistema de controle de Versão)

API - *Application Program Interface*

LAN - *Local Area Network* (Rede Local)

XP – *Extreme Programming*

SUMÁRIO

I. INTRODUÇÃO

1. JUSTIFICATIVA DO PROJETO

Em um mercado capitalista e globalizado, empresas e companhias dos mais diversos setores e categorias estão em constante disputa para alcançar e conquistar o maior número de clientes possível, sempre visando aumentar seus ganhos. Em vista disso, especialmente nas últimas décadas, o investimento de tecnologias por parte dessas empresas é considerado um grande aliado nos negócios, pois já se mostrou não apenas como um fator diferencial ou vantagem competitiva, mas como uma necessidade ou pré-requisito básico para não se tornar um empreendimento obsoleto em relação à grande maioria da concorrência. Tal fato pode ser evidenciado pelo aumento da oferta e da qualidade de serviços prestados, pelo incremento na eficiência e rapidez dos funcionários e pela diminuição dos custos de produção em contraste com a expansão do número de produtos manufaturados pela grande maioria das empresas que investem em tecnologia (DINIZ, 2005).

Portanto, este documento visa dissertar sobre o desenvolvimento do *software Blue Buy* que, seguindo os preceitos supracitados, deve servir como uma ferramenta de auxílio para funcionários de uma mercearia ou restaurante. Para isso, esse sistema deve, a princípio, permitir o fácil gerenciamento de um estabelecimento semelhante a uma lanchonete ou mercado, visando auxiliar no trabalho dos funcionários, agilizar processos como realização das vendas e gerenciar fornecimentos e saídas de objetos do estoque, podendo ser caracterizado como um *software* PDV (*Software* de Ponto de Venda).

2. CAPÍTULOS DESTE DOCUMENTO

Com a argumentação já citada acima, este primeiro capítulo buscou explicar as razões para a qual se decidiu construir o *software Blue Buy* e dissertar a respeito dos objetivos desse documento. O capítulo **II** buscará comentar acerca das premissas, justificativas e objetivos do projeto e do *software* em si. O capítulo **III** apresentará ao leitor algumas informações básicas, complementares e objetivas para a boa compreensão do documento.

O capítulo **IV** ficará incumbido de apresentar e explicar o papel de cada uma das ferramentas e *frameworks* usados no sistema. No capítulo **V**, serão discutidos os métodos usados para o desenvolvimento do *software*. Por fim, o capítulo **VI** comentará os principais resultados finais do desenvolvimento do sistema, bem como algumas conclusões acerca desse projeto.

II. CARACTERIZAÇÃO DO PROJETO

1. ESCOPO DO PROJETO

1.1. Justificativas e benefícios do *software*

O IFMG (Instituto Federal de Minas Gerais) - Campus Formiga possui dentro de suas instalações um restaurante/lanchonete pertencente ao Ronei, porém, os funcionários em que lá trabalham (incluindo o Ronei) possuem várias reclamações quanto ao atual *software* de gerenciamento usado por eles. Dessa forma, usando esse cenário como exemplo base para o desenvolvimento, foi construído o *software Blue Buy*, visando resolver os contras do atual sistema implantado no estabelecimento e, com base nas necessidades e especificações dos funcionários, atribuindo ao sistema um motivo mercadológico para ser desenvolvido. Além do mais, outra razão para desenvolvê-lo seria o aprendizado, já que as técnicas e conhecimentos empregados em sua construção podem contar como uma boa forma de adquirir experiência prática no desenvolvimento de sistemas.

1.2. Objetivos

1.2.1. Objetivos Gerais

Desenvolver um sistema para a Web que auxilie na gestão financeira e que permita o gerenciamento dos funcionários, fornecedores, clientes, estoque, etc, de um estabelecimento de vendas, sendo associado à categoria de *software* PDV (SAMPAIO, 2017).

1.2.2. Objetivos específicos do *software*

- Permitir o gerenciamento (Comumente chamado de CRUD - *Create, Read, Update, Delete* - Cadastro, Leitura de dados, Alteração e Deleção) de clientes do estabelecimento.
- Permitir o gerenciamento dos funcionários do estabelecimento e de seus respectivos cargos e atribuições.

- Permitir o gerenciamento dos fornecedores de mercadorias do estabelecimento.
- Permitir o gerenciamento dos objetos vendidos/fornecidos no estabelecimento, incluindo informações como grupos, subgrupos, tipo de objeto (produto ou mercadoria), entre outros.
- Registrar as vendas feitas no estabelecimento, guardando dados como o cliente que a fez, o funcionário que a realizou, os itens vendidos, entre outros.
- Registrar os fornecimentos ao estabelecimento, guardando dados como o fornecedor que o fez, os itens fornecidos, entre outros.
- Registrar as saídas de mercadorias, sejam elas automáticas (quando há mercadorias na venda) ou manuais.
- Fornecer relatórios de análise detalhados e explicativos dos ítens descritos acima.

III. REFERENCIAL TEÓRICO

1. DESENVOLVIMENTO *WEB* OU *DESKTOP*

Tradicionalmente, o desenvolvimento de aplicações geralmente foca sistemas para *desktop*, onde o *software* é instalado individualmente na(s) máquina(s) do clientes, se esse for o caso. Entretanto, esse método de programação se mostra atualmente ultrapassado, afinal, várias falhas e fraquezas podem ser pontuadas. Aplicações *web*, por outro lado, estão cada vez mais e mais presentes no ramo empresarial, sendo que estas apresentam inúmeras funcionalidades, aplicabilidades e vantagens em relação àquelas. (REDAÇÃO, 2018)

Visando melhorar o desempenho da aplicação, é comum que se melhore o *hardware*, porém, para um aplicativo *desktop*, fazer essa melhora para cada máquina de cada cliente pode ser muito caro e levaria muito tempo. Por outro lado, uma aplicação *web* é comumente implantada em um servidor. Com isso, a aplicação em si fica centralizada, logo, qualquer manutenção ou atualização precisa ser realizada apenas uma vez e em apenas um único lugar. (REDAÇÃO, 2018; MACORATTI, 2018)

Além disso, o bom funcionamento de uma aplicação depende da sua compatibilidade com o ambiente, sistema operacional, outros *softwares* e até com o *hardware* em que será instalada. Com isso, aplicações *desktop* devem ter versões diferentes para as mais variadas plataformas, ao passo que sistemas web precisam apenas ser implantados em um servidor compatível e o restante da interpretação do código da aplicação ficará com o *browser*, sem falar que a grande maioria dos usuários já está acostumada com navegadores, o que geralmente permite dispensar tutoriais. (SALES, 2009)

É fato que o desenvolvimento *desktop* já está bem mais consolidado, ou seja, há inúmeras ferramentas e *frameworks* com as mais variados escopos, aspectos e funcionalidades, porém, novas tecnologias e métodos de desenvolvimento *web* surgem a uma periodicidade extremamente curta, buscando tornar as aplicações para a *internet* muito mais fáceis de se codificar e de se usar do que um *software desktop*. (MACORATTI, 2018)

2. MVC

O MVC (*Model-View-Controller*) é um padrão de arquitetura, cujo principal objetivo é separar as funcionalidades ou aspectos (como a interface e a regra de negócio) de uma aplicação, promovendo uma codificação mais concisa e organizada, auxiliando especialmente no desenvolvimento em equipe, sendo assim uma das arquiteturas de *software* mais usadas em sistemas e *frameworks*. Formalmente, esse padrão divide o sistema em basicamente três camadas: *Model* (Modelo), *View* (Visão) e *Controller* (Controle). (JARVIS, 2011)

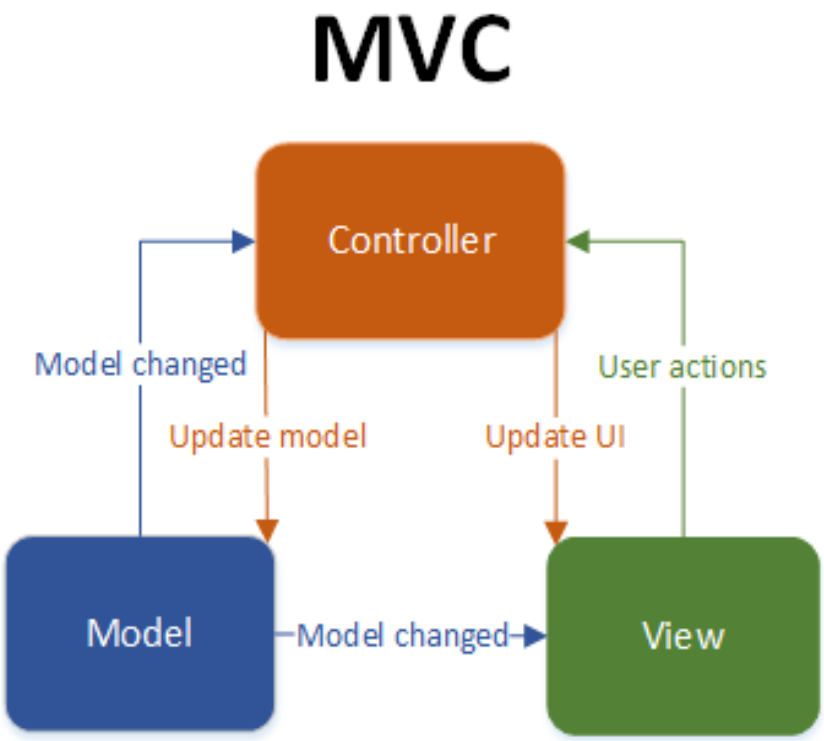
A Modelo é responsável por gerenciar e manipular os dados da aplicação, controlando também funções e comportamentos fundamentais do sistema e ainda tendo o papel de se comunicar com o banco de dados, realizando inserções (INSERT), edições (UPDATE), consultas (SELECT) e deleções (DELETE). Além disso, ela se comunica principalmente com a camada Controle, enviando e respondendo a requisições, porém, em situações mais raras (com *frameworks* específicos), essa camada pode também se comunicar diretamente com a Visão ao, por exemplo, atualizar a interface caso um erro ocorra durante a comunicação com o banco de dados. (MEDEIROS, 2013)

A camada Visão é responsável por construir a interface do sistema, estruturando dados e informações para o usuário da aplicação. Em alguns casos, esses dados devem ser enviados ou acessados no banco de dados, porém essa camada não pode se comunicar diretamente com a Modelo nem com o banco, logo ela se relaciona primeiramente com a Controle.

A camada Controle serve como um intermediário entre a Visão e a Modelo, sendo responsável por receber e enviar requisições e respostas de e para as outras camadas. Assim, ela deve, por exemplo, receber dados inseridos pelo usuário (na Visão) e enviá-los para a Modelo, onde essas informações serão processadas, gerenciadas e persistidas. Por outro lado, se a camada Modelo retornar informações (provindas do banco de dados, por exemplo) para a Controle, essa última deve redirecionar esses dados para a visão, que deve apresentá-los ao usuário. Por fim, a Controle também

consegue realizar chamadas e requisitar que determinadas funções sejam executadas nas outras camadas. (MEDIA, 2017)

Figura 1 - Comunicação entre as camadas de uma aplicação no padrão MVC



Fonte: NUNES (2017)

3. LINGUAGEM JAVA

No início da década de 1990, com a ascensão da internet e das aplicações para a web, vários paradigmas, conceitos e, consequentemente, problemas associados à computação surgiram, como questões relacionadas a ponteiros, gerenciamento de memória, pouca organização, falta de bibliotecas, reescrita de código ao mudar de sistema operacional, custo financeiro de usar a tecnologia, entre outros (CAELUM, 2017).

Concomitante com os sistemas web, James Gosling, Mike Sheridan, e Patrick Naughton, engenheiros da *Sun Microsystems* (comprada em 2009 pela *Oracle*), fundaram, em 1991, um grupo intitulado de *Green Team*, objetivando

construir uma linguagem simples, capaz de ser rapidamente codificada e implementada em vários tipos de dispositivos digitais com sistemas embarcados. Inicialmente, a linguagem seria testada em um controle para televisões a cabo interativas, entretanto, esse conceito estava muito à frente de seu tempo e o projeto não teve sequência (LUCKOW; MELO, 2017. ORACLE, 2018).

Porém, a *Sun* percebeu que uma linguagem universal como o Java, independente de plataformas, poderia ser adaptada para realizar processamentos e operações relativamente avançados nos *browsers*, trazendo inúmeras vantagens em relação ao desempenho e comodidade das aplicações. Assim, o projeto surgiu com uma ideia inicial, foi lançado com outro propósito, mas, atualmente, o Java tem maior destaque na codificação *back-end* de servidores, afinal, graças a uma grande popularidade e a uma comunidade extremamente atuante e colaborativa, a linguagem Java para esse contexto ganhou inúmeras melhorias e atualizações, se tornando uma linguagem rica, completa, robusta e, ao mesmo tempo, fácil de se compreender e codificar e com um alto desempenho e escalabilidade, resolvendo facilmente os problemas já citados acima relacionados àquela época (CAELUM, 2017).

4. JSF

O **JSF** (*JavaServer Faces*) é a especificação (ou seja, um método de desenvolvimento de aplicações) para um *framework* que faz parte de uma outra especificação, o **Java EE** (Java *Enterprise Edition*). Essa ferramenta é internamente baseada em *Servlets* (Classes Java responsáveis por receber e responder a chamadas HTTP – *Hypertext Transfer Protocol*), porém, aquela apresenta inúmeros recursos, melhorias, funcionalidades e, consequentemente, vantagens em relação a estes. (SCHALK, 2005)

O JSF é considerado uma especificação, porém, apenas com essa caracterização, não pode ser considerado um produto concreto. Para isso, é necessário recorrer a uma implementação do *JavaServer Faces*, sendo que uma das mais conhecidas e usadas no mercado é a **Mojarra**. Essa ferramenta permite, em suma, o desenvolvimento Web usando a linguagem Java,

disponibilizando ainda vários recursos, funções e aplicabilidades, separando ainda as suas funcionalidades em uma forma concisa e clara de acordo com o padrão MVC, sem que tenhamos que nos preocupar com essa organização. (LUCKOW; MELO, 2017)

Em uma aplicação web comum, cada componente da página (geralmente um arquivo .html) deve ser construído “do zero”, sem falar que, em páginas dinâmicas, cujo conteúdo varia, por exemplo, de acordo com dados inseridos pelo usuário, a complexidade é ainda maior. Porém, o JSF disponibiliza uma extensa e produtiva coleção de componentes pré-construídos de fácil manuseio, usabilidade e adequação a cada contexto, sem sacrificar a flexibilidade e o poder de desenvolvimento, variando desde um simples *outputLabel*, usado para mostrar um texto qualquer, até uma complexa *dataTable*, usada para construir tabelas de dados, sem falar que ainda dispõe de uma biblioteca “*core*” usada para fazer conversões e validações de valores na própria página. (SCHALK, 2005)

Essa implementação, porém, não se limita a auxiliar os desenvolvedores apenas no *front end*, mas também disponibiliza uma extensa API (*Application Program Interface*) em Java para auxiliar na comunicação entre a interface e a regra de negócio, oferecendo ainda uma forma mais intuitiva e transparente de se gerenciar o ciclo de vida das páginas, da sessão de cada usuário e da aplicação em como um todo. Com isso, o JSF se torna o principal e mais usado *framework* do projeto, afinal, ele é o responsável por construir basicamente toda a interface e integrá-la à regra de negócio implementada no servidor de uma forma simples e clara, porém eficiente, promovendo grande produtividade e reduzindo o tempo de desenvolvimento. (LUCKOW; MELO, 2017. SCHALK, 2005)

IV. MATERIAIS E MÉTODOS

1. *HARDWARE* USADO

O hardware usado para desenvolver o software foi um notebook Acer Aspire E5-574, processador Intel® Core™ i7-6500 com barramento de 64 bits, memória RAM DDR3 de 8GB e 1TB de HD. O SO (Sistema Operacional) utilizado será o Windows 10 Home Single Language, versão 10.0.16299.

2. LINGUAGENS DE PROGRAMAÇÃO USADAS

As linguagens de programação usadas para codificar o *software* foram:

- XHTML (*EXtensible HyperText Markup Language*): Composição entre as linguagens XML e HTML, é usada para construir as páginas Web da interface.
- JavaScript: Linguagem menos usada no projeto, sendo responsável por algumas funções mais específicas do funcionamento e da animação da página.
- CSS: Linguagem responsável por estilizar e dar um aspecto agradável às páginas.
- EL (*Expression Language*): Trabalhando em conjunto com o XHTML, é a responsável por se comunicar com classes Java e acessar dados dinamicamente desses arquivos.
- Java: Linguagem mais usada em todo o projeto, é responsável construir e gerenciar toda a Regra de Negócio, Controle e acesso ao Banco de Dados do sistema.

3. *SOFTWARES* E *FRAMEWORKS* USADOS

3.1. IDE (*Integrated Development Environment*) *NetBeans*

Por ser um sistema Web, a sua codificação envolveu várias linguagens de programação, como Java, Javascript, XHTML, CSS, etc, cada uma com seus níveis de dificuldade e aplicações. Desenvolver um *software* dessa complexidade em um mero editor de texto ou ainda em um editor de códigos não seria nem um pouco agradável, eficiente e confortável. Dessa forma, optou-se por usar a **IDE Netbeans 8.2**. Com ela, tarefas não tão simples porém essenciais e úteis, como identação, destacamento, autocompletamento de texto, identificação de erros de sintaxe, depuração, refatoração e compilação de códigos, entre outras funções, - que não podem ser encontradas em um simples editor de texto - puderam ser muito bem aproveitadas, promovendo um aumento na produtividade e na rapidez do desenvolvimento (GODOI, 2016).

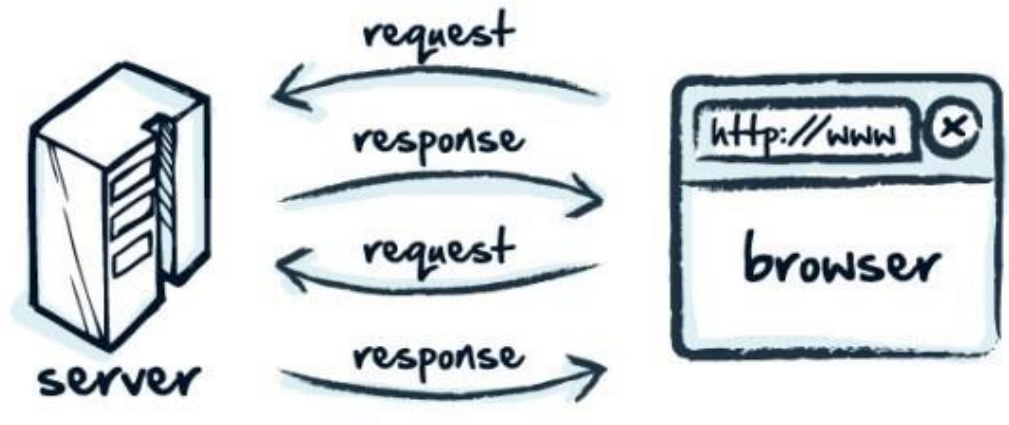
3.2. Mozilla

Por ser um sistema PDV (Ponto de Venda), haverá constante comunicação e interação entre o *software* e o usuário final - nesse caso, uma pessoa. Para isso, é essencial que haja um intermediário, ou seja, uma interface, para que essa interlocução seja efetiva. Centenas de linhas de código devem, portanto, ser compiladas, interpretadas e, então, apresentadas em forma de uma interface agradável ao usuário final. O responsável por fazer isso, no contexto desse projeto, será o ***Browser Mozilla Firefox* 61.0.2**.

3.3. GlassFish

Ainda no contexto de um sistema PDV, sabemos que um *Browser* é o responsável por apresentar os dados em uma interface, porém, tanto esses dados quanto o próprio código base para essa interface devem vir de um lugar específico, que nesse caso, será um servidor.

Figura 2 - Arquitetura básica entre um cliente e um servidor



Fonte: FOGASSIA (2017)

Toda a regra de negócio, armazenamento, controle, validação e manipulação dos dados são realizados no servidor, que tem também o papel de esperar por uma chamada (geralmente por protocolo HTTP - *Hiper Text Transfer Protocol*) e fornecer os dados requisitados e serviços solicitados para possíveis clientes (como os *Browsers*). Para isso, nesse projeto, será usado o **servidor Oracle GlassFish 4.1.2** (MARQUES, 2018).

3.4. Banco de Dados

A inserção e a manipulação de dados são, incontestavelmente, as principais funcionalidades de um sistema PDV, entretanto, essas funções seriam inúteis se, ao encerrar o programa, todos os dados gerados fossem perdidos. Portanto, nesse contexto, a persistência de dados, ou seja, o armazenamento organizado e estruturado dessas informações também se mostra como uma função necessária para a execução do *software*. Dessa forma, nesse projeto, será usado o **Banco de Dados MariaDB 10.1.32**.

3.5. MySQL Workbench

O banco de dados usado nesse projeto não tem exatamente uma interface na qual se pode fazer uma visualização mais clara e organizada dos dados, e isso pode ser um empecilho quando se tenta fazer um *debug* ou verificar se os dados estão sendo persistidos e recuperados corretamente. Existem, porém, diversas ferramentas que podem auxiliar desenvolvedores nesses aspectos, chamadas de SGBDs (Sistema Gerenciador de Bancos de Dados). Nesse projeto, a ferramenta utilizada como SGBD foi o ***MySQL WorkBench* 6.3.8.**

3.6. Hibernate

Como dito anteriormente, um banco de dados desempenha uma função essencial nesse *software*. Porém, a camada do sistema que conecta o controle/regra de negócio ao banco de dados em si também deve ser igualmente bem estruturada, organizada e codificada. Assim, visando ganhar dinamicidade e produtividade, foi usado o ***framework Hibernate 4.3.11***, um *framework* ORM (*Object/Relational Mapping* - Mapeamento Objeto/Relacional) responsável por mapear entidades Java em tabelas equivalentes no modelo relacional e eficientemente auxiliar na persistência e na consulta de dados (HIBERNATE, 2018).

3.7. JSF/Mojarra

Assim como já foi explicado no Referencial Teórico, o principal *framework* usado na construção da aplicação foi o **JSF 2.2**, juntamente com a sua já comentada especificação, a **Mojarra 2.2.12**.

3.8. PrimeFaces

Além dos benefícios do JSF já comentados na seção **III.**4, pode-se acrescentar que outro de seus pontos positivos é a sua alta extensibilidade, aceitando diversos *plugins* e *frameworks* complementares usados para adicionar funcionalidades ao sistema. Um desses mais famosos *plugins*, sendo

usado ainda para o desenvolvimento do *software* neste projeto, é o **PrimeFaces 6.2**, uma incrivelmente rica porém leve paleta de componentes JSF, oferecendo mais de 100 itens previamente configurados e de fácil implementação, variando desde simples campos de texto a grandes e complexos gráficos, tabelas e até mapas. (ÇIVICI, 2018)

Além disso, o PrimeFaces, ainda, oferece uma forma extremamente fácil e prática de implementar eventos, *listeners*, chamadas assíncronas, conversores, validadores (especialmente na "*Client Side Validation*" - "Validação no Lado do Cliente"), dentre outros, disponibilizando, também, mais de 30 temas e *templates* internos, uma documentação profundamente detalhada e um interessante *showcase* interativo em seu *site* oficial. (LUCKOW; MELO, 2017)

3.9. Maven

Como qualquer *software* relativamente complexo, ainda mais um desenvolvido majoritariamente em Java, vários dos métodos, funções e *Frameworks* usados pertencem a terceiros, como o *Hibernate*, JSF, *PrimeFaces*, entre outros, assim, para que possam ser implementados, dependências necessitam ser importadas. Entretanto, localizar, baixar e gerenciar essas bibliotecas pode ser um processo trabalhoso, cansativo e improdutivo, dessa forma, optou-se por usar o ***framework*** **Maven 3.5.0**, responsável não apenas por gerenciar suas dependências mas também auxiliar na organização e no *build* do projeto. (RICARDO, 2014)

3.10. CVS Git/GitLab

Um CVS (*Concurrent Version System*), ou Sistema de Controle de Versão) é o *software* responsável por manter um versionamento ou gerenciamento organizado e conciso das diversas versões de um programa, sendo extremamente útil quando se está trabalhando em um projeto muito grande, com vários módulos e com uma grande equipe de desenvolvimento, ajudando ainda a evitar conflitos em arquivos e a comparar arquivos de

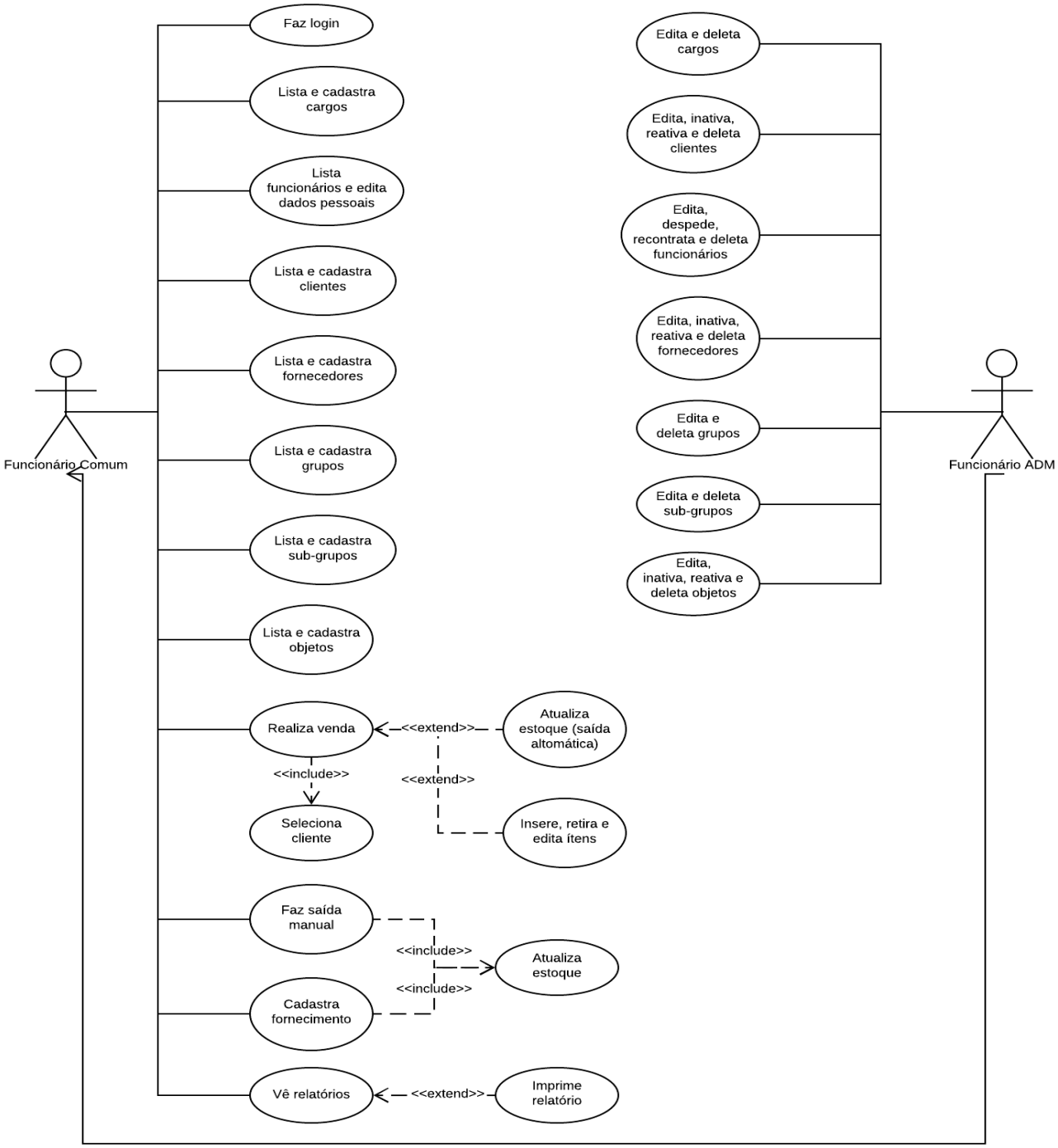
versões diferentes (REIS, 2006). Dessa forma, nesse projeto, foi utilizado o **CVS Git 1.22.1.1**, em conjunto com o repositório remoto **GitLab 11.2.3**.

V. DESENVOLVIMENTO DO PROJETO

1. DOCUMENTOS

2.1. Diagrama de Casos de Uso

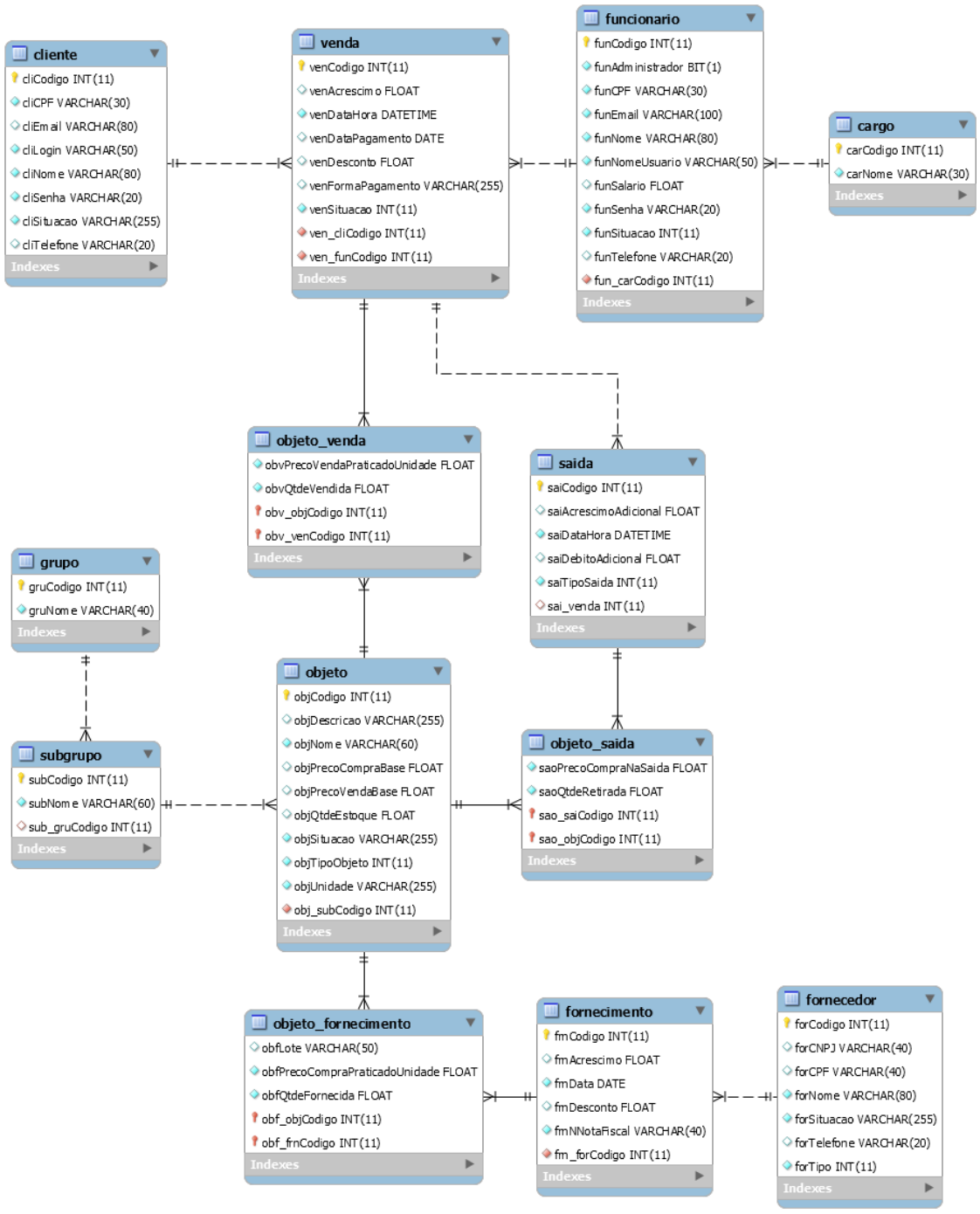
Figura 3 - Diagrama de casos de uso do sistema



Fonte: Elaborado pelo autor

2.2. Diagrama Entidade-Relacionamento

Figura 4 - Diagrama Entidade-Relacionamento do Banco de Dados



Fonte: Elaborado pelo autor

Como podemos ver na **Figura 4**, o programa em si possui relativamente poucas tabelas, porém, a quantidade de atributos e a de dados a serem inseridos, bem como a relação entre eles, torna a manipulação e o

gerenciamento dessas informações bastante complexa. Além disso, analisando cada tabela, podemos estimar e deduzir algumas regras de negócio e funcionalidades que foram implementadas no sistema.

Um **Cliente** possui um código único, um CPF, e-mail eletrônico, um nome, um telefone e a sua respectiva situação em relação ao estabelecimento (ativo ou inativo). Um **Cargo** possui um código que o identifica e o seu nome.

Um **Funcionário** possui um código único, um CPF, um e-mail, um nome, um nome de usuário e senha para fazer *login* no sistema, a sua respectiva situação em relação ao estabelecimento (ativo ou inativo), um salário, um número de telefone um dado que diz se ele é um usuário administrador ou não e o respectivo cargo ao qual pertence.

Uma **Venda** possui um código identificador, a data/hora em que foi realizada, a data de pagamento, os valores de acréscimo e desconto, a forma de pagamento, a sua situação (paga ou não paga), o funcionário que fez essa venda e o cliente que a realizou. Um **Grupo** possui um código único e um nome. Um **Subgrupo** possui um código identificador, um nome e, opcionalmente, o grupo ao qual pertence.

Um **Objeto** possui um código único, um nome, uma descrição, seus preços base de compra e venda, sua unidade de medida, sua situação, sua atual quantidade em estoque, caso esse valor possa ser calculado, o tipo de objeto (produto ou mercadoria) e o subgrupo ao qual pertence. Um produto é aquele destinado exclusivamente à venda e não pode ter um gerenciamento de estoque (um prato de almoço a quilo feito pelo próprio cliente, por exemplo). Uma mercadoria, por sua vez, é aquela que pode ser fornecida e, consequentemente, ter seu gerenciamento de estoque (como um sorvete).

Um **Fornecedor** possui um código, um nome, um CNPJ (se for pessoa jurídica) ou um CPF (se for pessoa física), uma situação, um telefone de contato e o seu tipo (Pessoa física ou jurídica). Um **Fornecimento** possui um código, os valores de acréscimo e desconto, o número da nota fiscal, a data e o fornecedor que a realizou.

Uma **Saída** possui um código, a data/hora em que fora realizada, o tipo dessa saída (manual ou automática) e, opcionalmente, a venda a qual está relacionada. Uma saída manual é, obviamente, realizada manualmente pelo próprio usuário e não está relacionada a nenhuma venda. A saída automática é cadastrada automaticamente quando uma venda que possua mercadorias é realizada.

Uma venda pode estar relacionada a um ou vários objetos e um objeto pode estar presente em nenhuma, uma ou várias vendas, sendo que cada **item na venda** possui o valor de venda praticado, a quantidade vendida, a venda e o objeto aos quais está relacionado.

Um fornecimento pode estar relacionado a uma ou várias mercadorias e uma mercadoria pode estar presente em nenhum, um ou vários fornecimentos, sendo que cada **mercadoria no fornecimento** possui o valor de compra praticado, a quantidade fornecida, o lote ao qual pertence, o fornecimento e a mercadoria aos quais está relacionado.

Uma saída pode estar relacionada a uma ou várias mercadorias, sendo que um mercadoria pode estar presente em nenhuma, uma ou várias saídas, sabendo ainda que cada **mercadoria na saída** possui o valor de compra na saída, a quantidade retirada, a saída e a mercadoria às quais está relacionada.

2. DESENVOLVIMENTO

Geralmente, projetos muito complexos, realizados por um grande time de desenvolvimento, estão mais propensos a falhar, afinal, o tamanho do projeto é comumente proporcional às dificuldades a serem enfrentadas, como as falhas de comunicação, inexperiência dos desenvolvedores, tempo e custo restritos, exigências e requisitos mais profundos e detalhados, entre outros fatores. (SANTOS, 2011)

Existem, porém, várias técnicas, chamadas de metodologias de desenvolvimento, que são um conjunto de métodos, regras e postulados, implementados em um contexto específico, para atingir um determinado objetivo, como o desenvolvimento de um sistema, definindo com clareza e

objetividade as principais etapas do projeto e separando estrategicamente os papéis e funções dos envolvidos, visando, dessa forma, reduzir as dificuldades, o custo e o tempo de desenvolvimento do projeto e, por conseguinte, aumentar a produtividade da equipe e a qualidade do produto final. (MEDEIROS, 2013)

Um desses processos é o **XP - *Extreme Programming***, um dos mais conhecidos e usados atualmente no mercado. Por mais que o projeto descrito nesse documento tenha proporções de dificuldade relativamente pequenas, o uso do XP como uma metodologia para o desenvolvimento do *software* foi crucial e útil para o seu sucesso.

O Extreme Programming é uma metodologia criada por Kent Beck enquanto trabalhava em um projeto de *software* de folha de pagamento para a companhia *Chrysler* no fim da década de 90. A execução desse método ágil tem por base a aplicação de alguns valores ou princípios básicos, como ter uma boa comunicação entre todos os envolvidos, englobando tanto os desenvolvedores e responsáveis do projeto quanto os seus clientes, de tal forma que a troca de informações e ideias seja fluida e constante. Outra base, concomitante com a anterior, é o *feedback* frequente, incluindo opiniões e retornos dos clientes além de resultados obtidos de testes constantes do próprio sistema. Com essas informações, funções desnecessárias, incorretas ou que geram erros podem ser identificadas, descartadas ou corrigidas, ao passo que mudanças necessárias podem ser precocemente implementadas, o que é considerada uma boa prática, permitindo, portanto, que o *software* evolua. (REIS, 2009; MEDEIROS, 2013)

Além disso, o XP sugere a codificação em etapas ou ciclos de curta duração, realizando também testes constantes e automatizados para cada módulo produzido. Com essa estratégia, é mais fácil prever, identificar e reduzir incertezas, riscos e erros, auxiliando ainda na implementação de possíveis mudanças e melhorias no código (refatoração). (MEDEIROS, 2013)

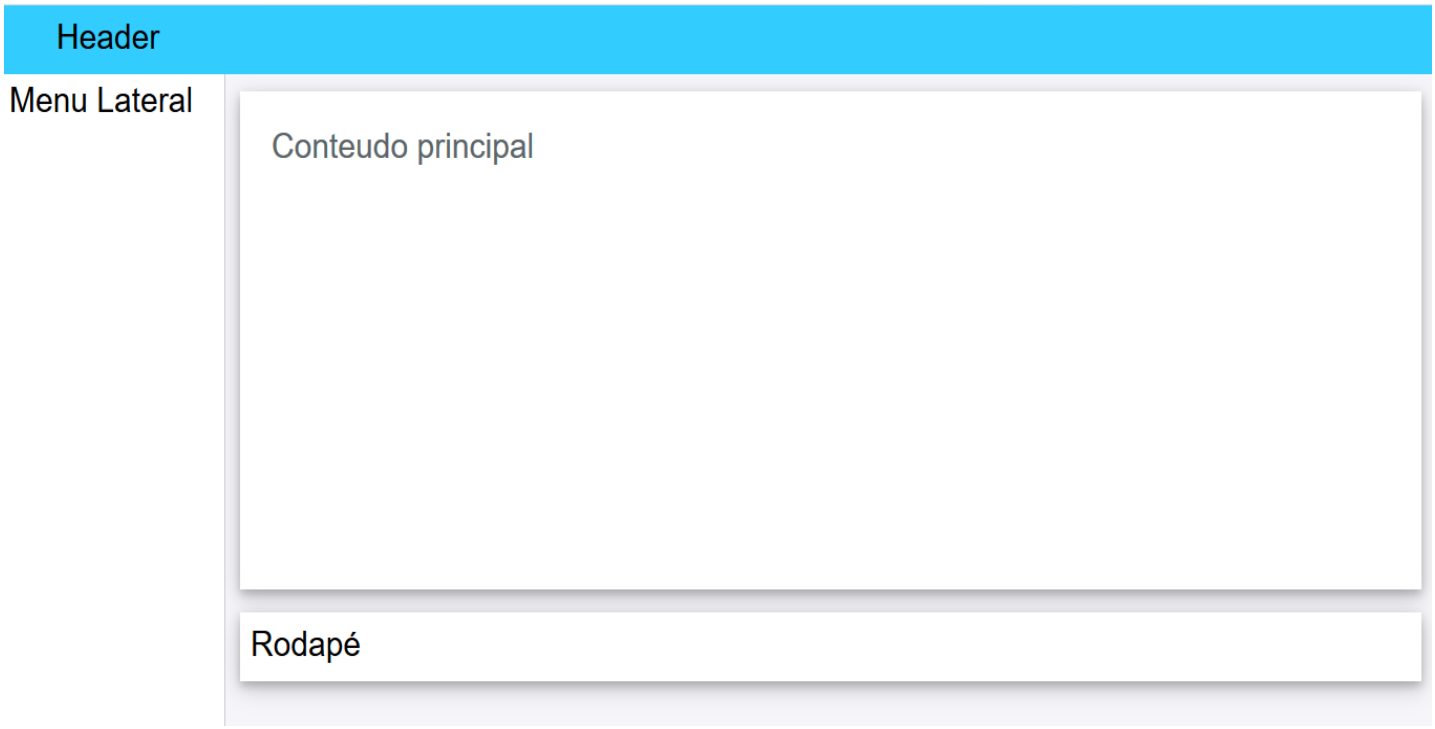
VI. RESULTADOS E CONCLUSÕES

1. PRINCIPAIS RESULTADOS FINAIS

1.1. *Template*

No início da construção da aplicação, um *template* inicial deveria ser construído, podendo ser usado e replicado em praticamente todas as telas do sistema. Assim, foi desenvolvido o seguinte *template* base:

Figura 5 - *Template* básico das páginas do sistema



Fonte: Elaborado pelo autor

1.2. Login

Além disso, tendo o escopo, os requisitos e as funcionalidades em mãos, houve a necessidade de se criar uma tela de Login para os funcionários usuários do sistema, resultando na seguinte interface:

Figura 6 - Página de Login do sistema

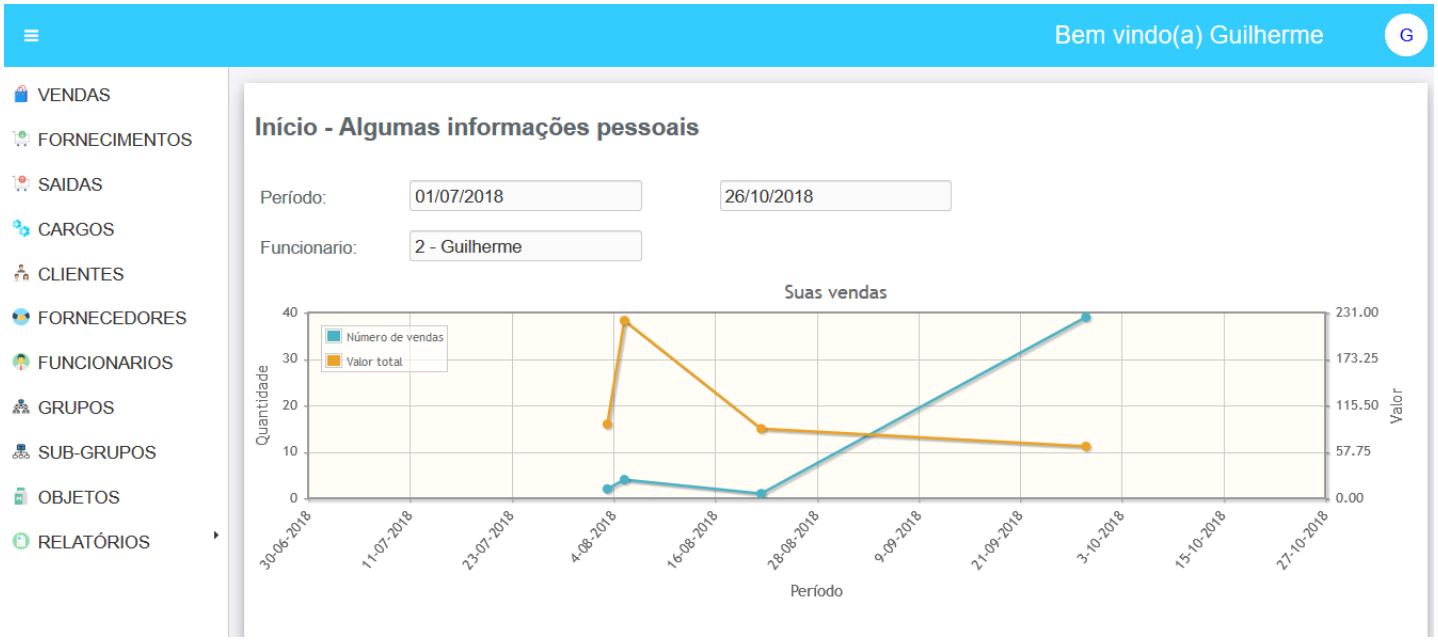
Fonte: Elaborado pelo autor

Vemos que há nessa tela, além da logo do *software* em evidência, dois campos de texto principais: um para a inserção do Nome de Usuário e outro para a senha, com os quais se faz a validação, autenticação e, caso os dados estejam corretos, o login do usuário, o levando para a tela de início.

1.3. Index

Essa é a tela com a qual o usuário terá o primeiro contato quando fizer o Login, logo, ela apresenta alguns dados mais gerais, informativos e estatísticos sobre si mesmo e as vendas.

Figura 7 – Index do sistema



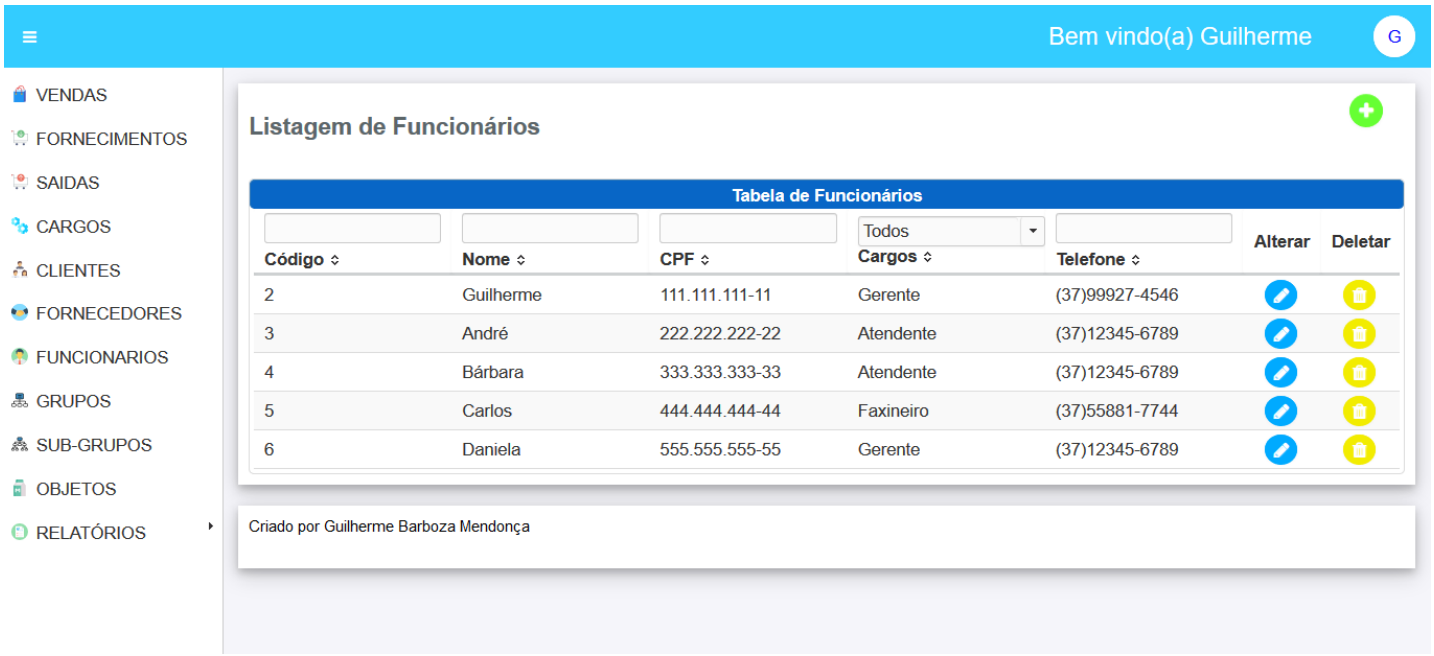
Fonte: Elaborado pelo autor

Com uma breve análise, pode-se notar que há, além do conteúdo principal, uma região contendo um menu lateral, uma contendo um cabeçalho ou *header*, e outra contendo um rodapé ou *footer* (visto apenas quando há pouco conteúdo na página ou caso o usuário avance até o final da página). O menu lateral será a principal forma pela qual o usuário trafegará pelas diversas telas do sistema. O *header* conterá um botão de *toggle* (faz com que o menu seja colapsado ou expandido, dependendo da sua posição inicial) para o menu lateral e um outro submenu com alguns atalhos. Por fim, o rodapé conterá apenas algumas informações mutáveis do sistema, como a autoria.

1.4. Listagem

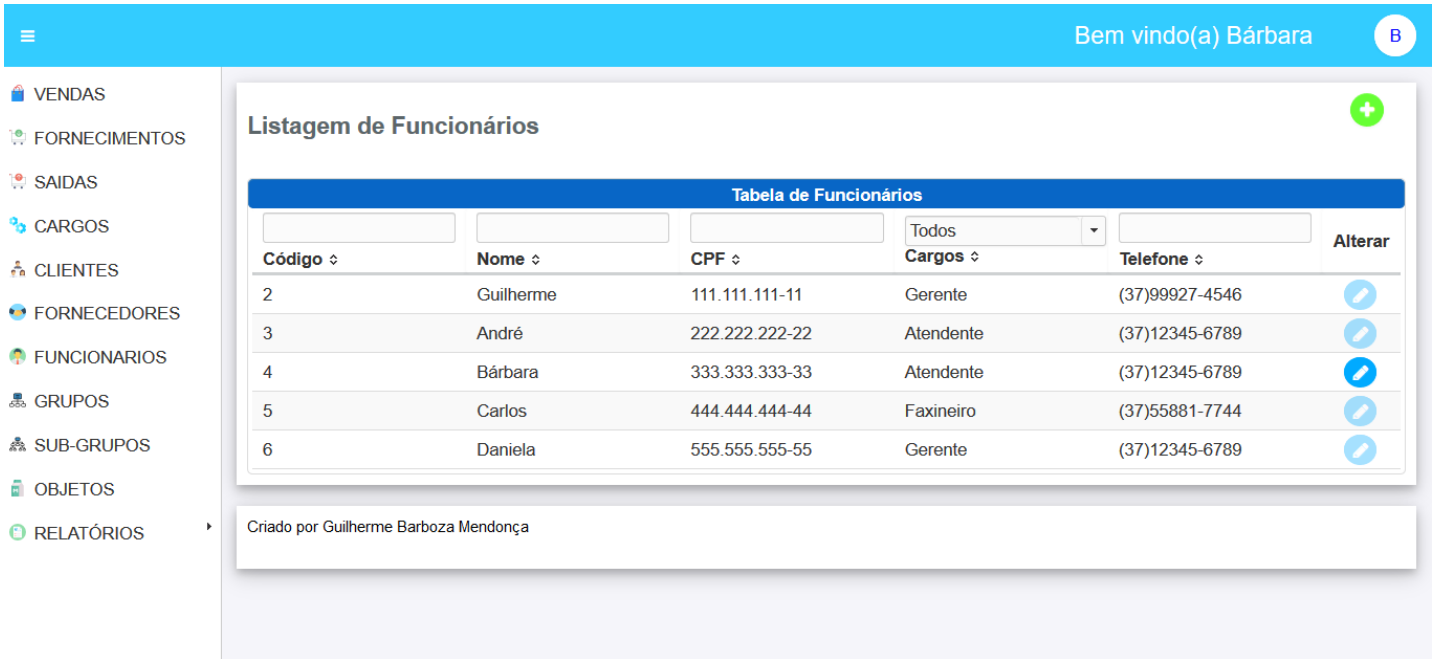
Vale ressaltar que, ainda seguindo o escopo do projeto, já foi observada a necessidade de gerenciar vários tipos de dados seguindo a estrutura dos CRUDs já comentados na seção **II.**1.2.2. Analisando o menu lateral explicado anteriormente, ao se clicar do item “CARGOS” ao item “OBJETOS”, o usuário será, por exemplo, redirecionado para uma tela de listagem para o respectivo item selecionado:

Figura 8 - Página de listagem dos funcionários



Fonte: Elaborado pelo autor

Figura 9 - Listagem dos funcionários quando um usuário não administrador está logado



Fonte: Elaborado pelo autor

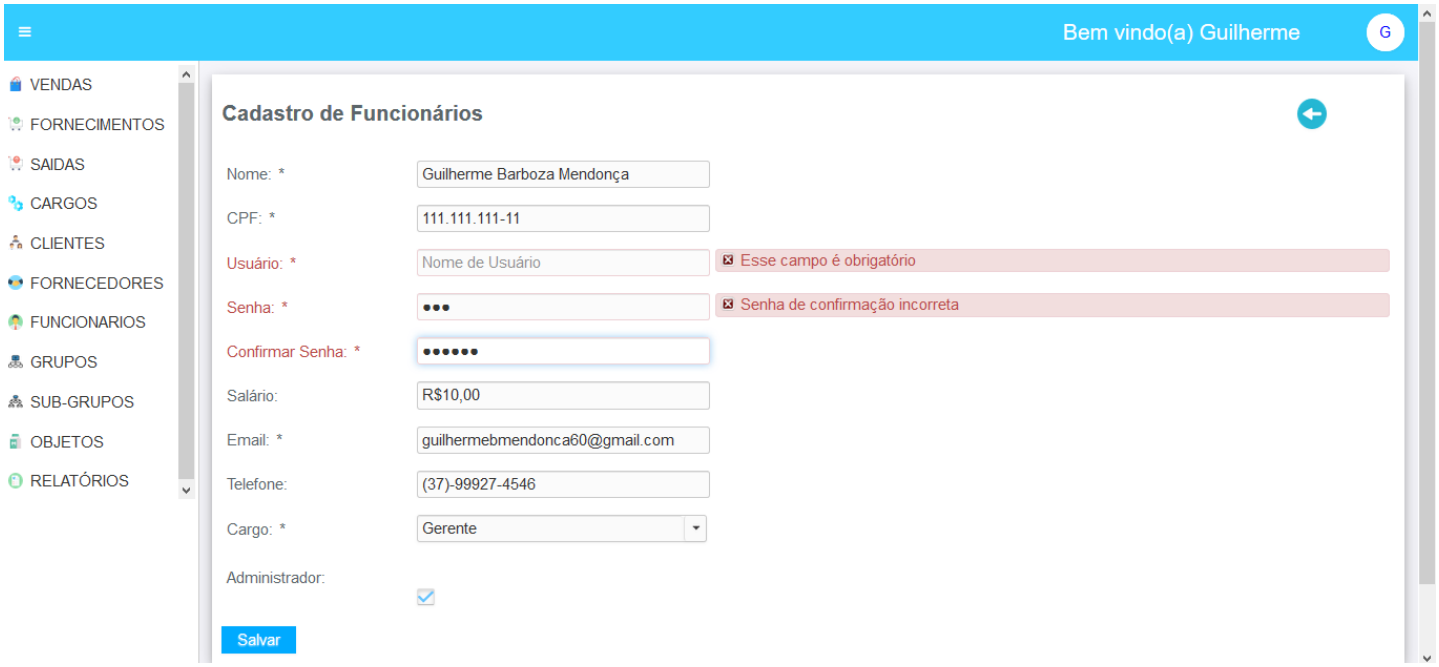
Serão apresentadas de forma tabular as principais informações acerca do item selecionado, permitindo também funções de ordenação e filtragem para cada dado. Além disso, pode-se notar a presença de um botão “+” no canto superior direito da página, bem como um botão “Caneta” e outro “Lixo” para cada linha da tabela. Quando o primeiro é selecionado, o usuário será redirecionado para a página de cadastro de um novo dado, enquanto que na seleção dos outros dois, serão executadas as funções de edição e deleção respectivamente para o dado selecionado.

Vale citar também que uma das premissas do sistema é fornecer uma estrutura hierárquica para funcionalidades exclusivas dos usuários administradores, impedindo a execução das mesmas por usuários não administradores. Isso pode ser observado na *figura 9,* pois “Bárbara” é uma usuária não administradora, logo, ela não tem permissão para editar os dados de outros funcionários ou ainda deletá-los (note que, nesse caso, nem houve a necessidade de mostrar essa coluna).

1.5. Cadastro

A Figura 9 consiste basicamente em vários campos de texto cujo valor é associado a um atributo da entidade que está sendo cadastrada, respeitando o tipo do dado (Texto, numérico, datas, etc), máscaras, filtros, obrigatoriedade, etc.

Figura 10 - Página de cadastro de um funcionário



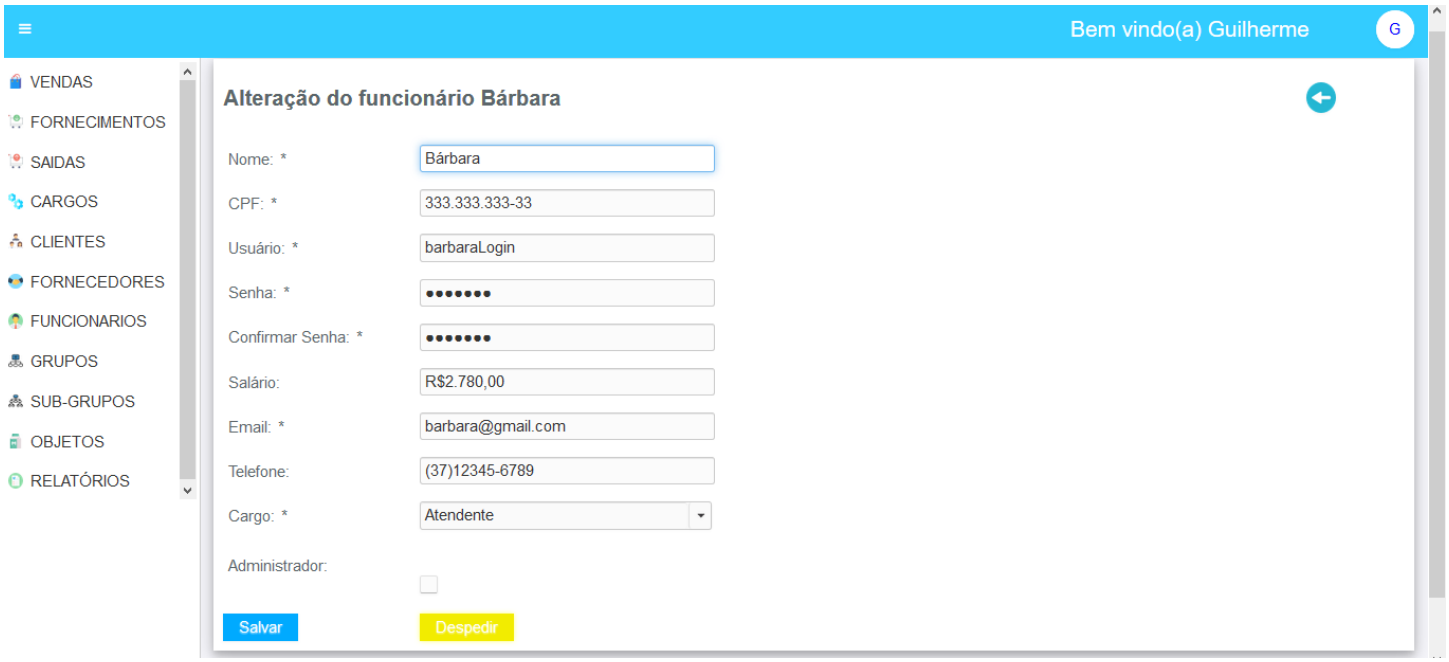
Fonte: Elaborado pelo autor

O botão “Salvar” permite, após a validação e verificação dos dados, a inserção dos mesmos no banco de dados.

1.6. Edição

A tela de edição é quase idêntica à tela de cadastro, com a diferença que ela é acessada pelo botão “Caneta” na tela de listagem e que os campos já vem pré-inseridos com seus respectivos valores. Basta, portanto, alterar os dados necessários e clicar no botão “salvar” para enviar as modificações para o banco de dados.

Figura 11 - Página de alteração dos dados de um funcionário específico

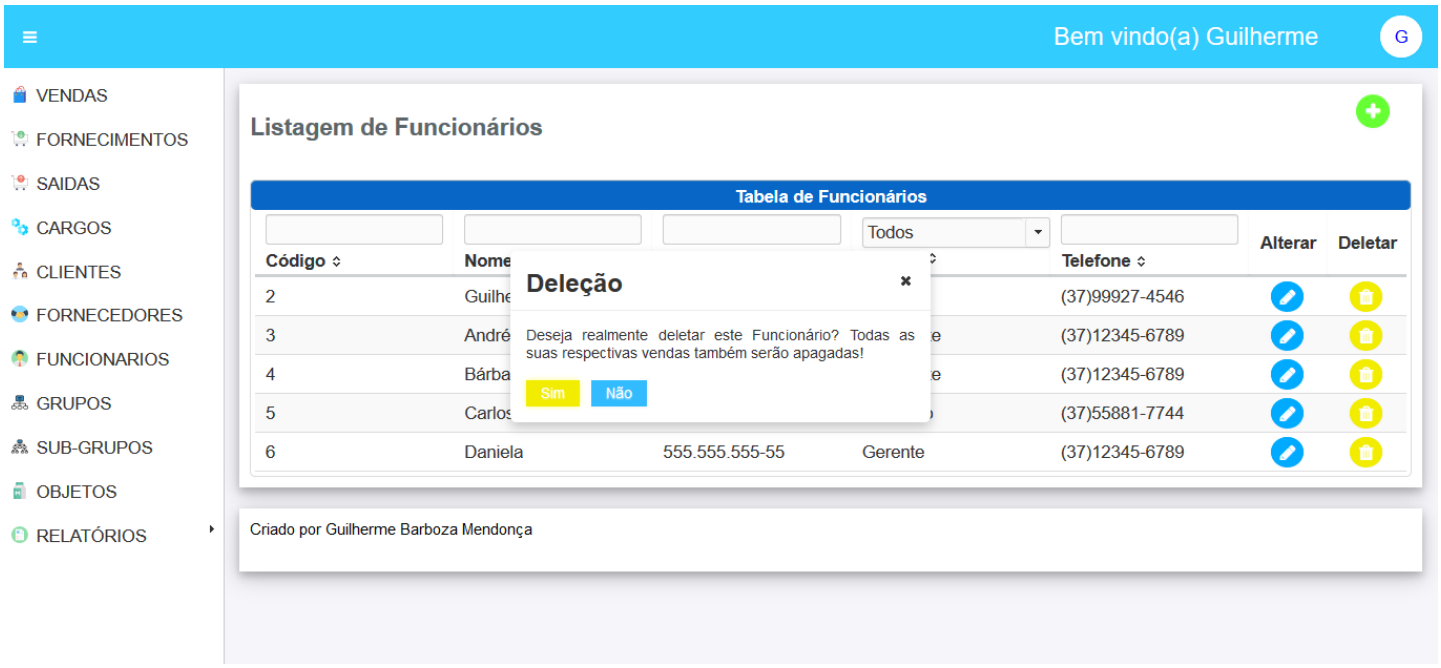


Fonte: Elaborado pelo autor

1.7. Deleção

Por fim, a última função de um CRUD é a deleção, que consiste basicamente na retirada de um dado específico do banco de dados, fim que pode ser atingido ao selecionar o botão “Lixo” na tela de Listagem (Como na *Figura 8*). Ao se clicar nesse botão, um painel será aberto para que, dessa forma, a deleção possa ser confirmada, afinal, essa é uma operação muito perigosa e prejudicial ao sistema devido à rígida integridade relacional entre os dados do banco, logo, essa é uma ação pouco recomendada. Pense, por exemplo, que a deleção de um único cargo irá, consequentemente, deletar todos os funcionários atualmente relacionados a esse cargo e, dessa forma, todas as vendas desses funcionários, assim como seus itens também serão apagados.

Figura 12 - Painel de deleção de um funcionário



Fonte: Elaborado pelo autor

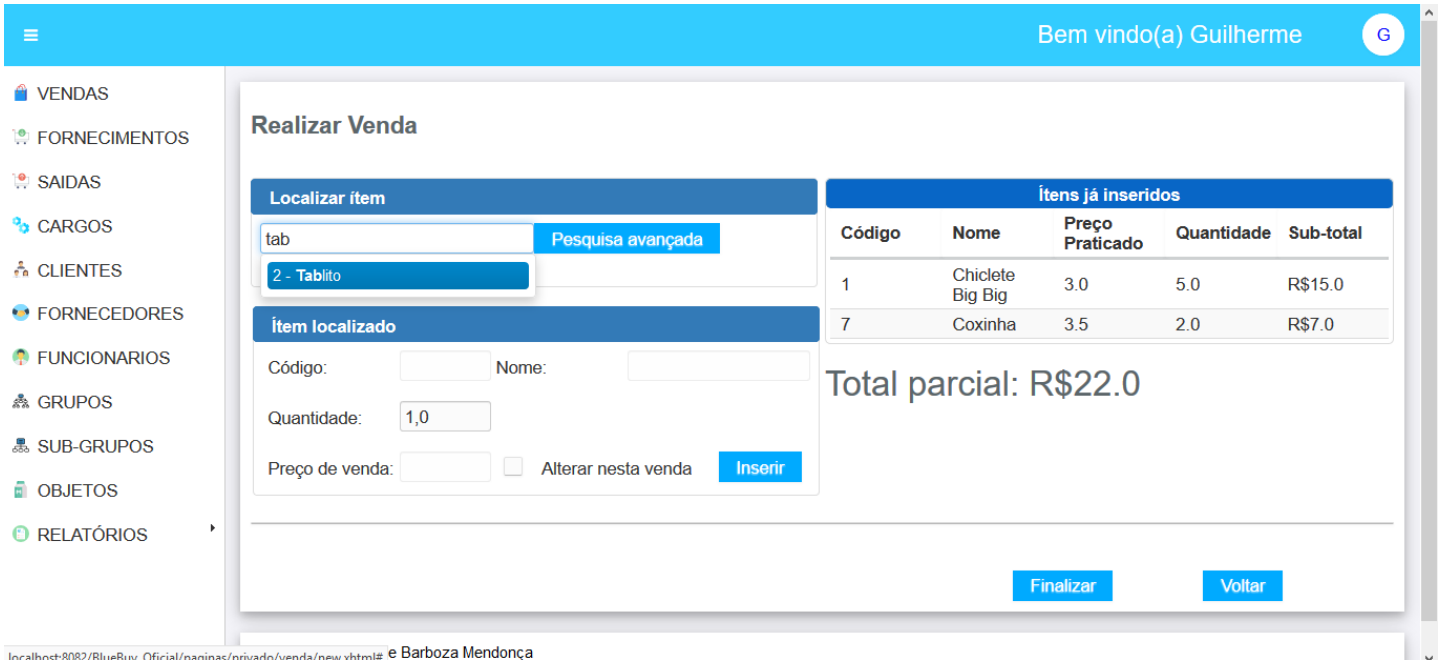
Retomando a tela de edição da *Figura 11*, podemos notar um botão “Demitir”, cuja abordagem apenas o exclui de consultas e relatórios objetivos e importantes sem ter que efetivamente retirar esse dado do banco, oferecendo, assim, uma alternativa menos impactante e perigosa.

1.8. Vendas, Fornecimentos e Saídas

A realização das vendas, dos fornecimentos e das saídas são, sem dúvidas, algumas das principais funcionalidades que um sistema PDV deve oferecer, pois são essas funções que dão ao sistema um aspecto financeiro. Essas páginas, especialmente a da venda, foram feitas para serem as mais dinâmicas e objetivas possíveis. Elas apresentam basicamente o mesmo aspecto: Todas possuem um campo onde é possível escolher um objeto, seja por meio do seu código identificador, seja pelo seu nome (ou por uma parte dele). Uma vez escolhido um item, basta inserir o preço (preço base já vem inserido por padrão) e a quantidade vendida/fornecida/retirada e enviá-lo para

uma tabela lateral. Ao final, quando todos os itens necessários forem selecionados, basta finalizar a venda, fornecimento ou saída.

Figura 13 - Página de realização da venda

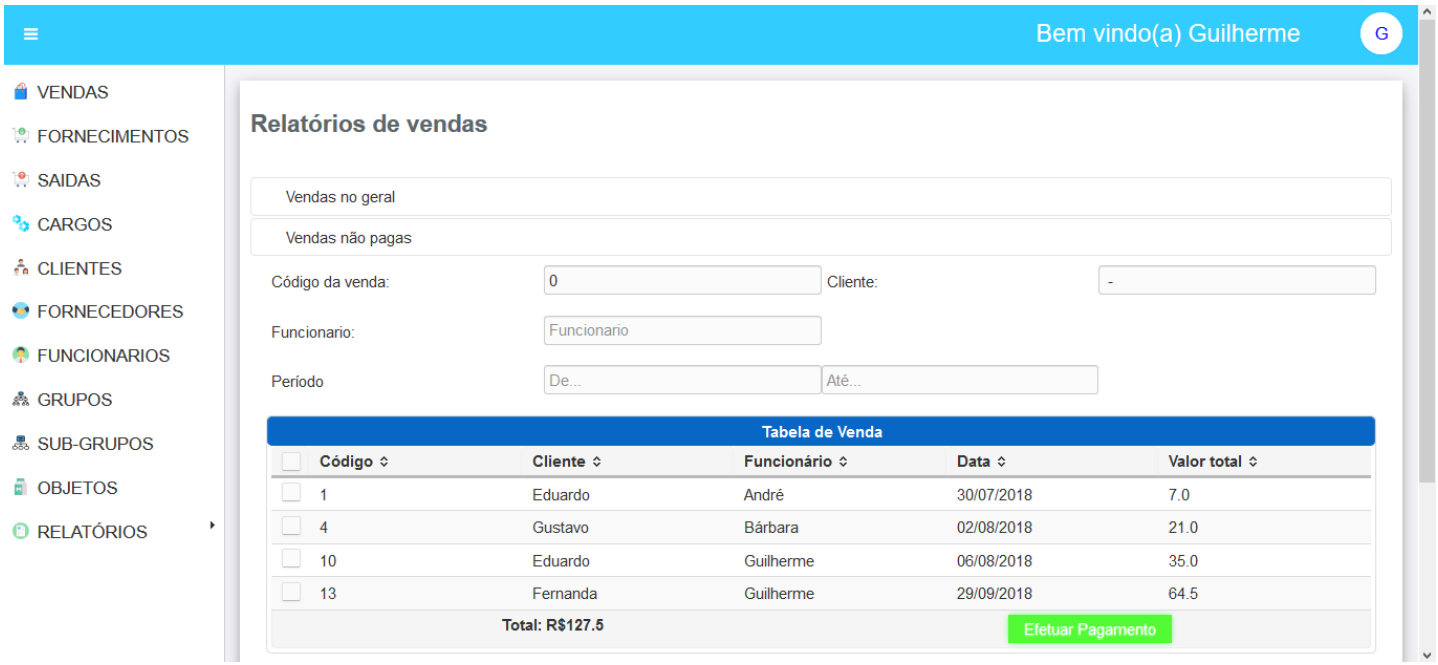


Fonte: Elaborado pelo autor

1.9. Relatórios

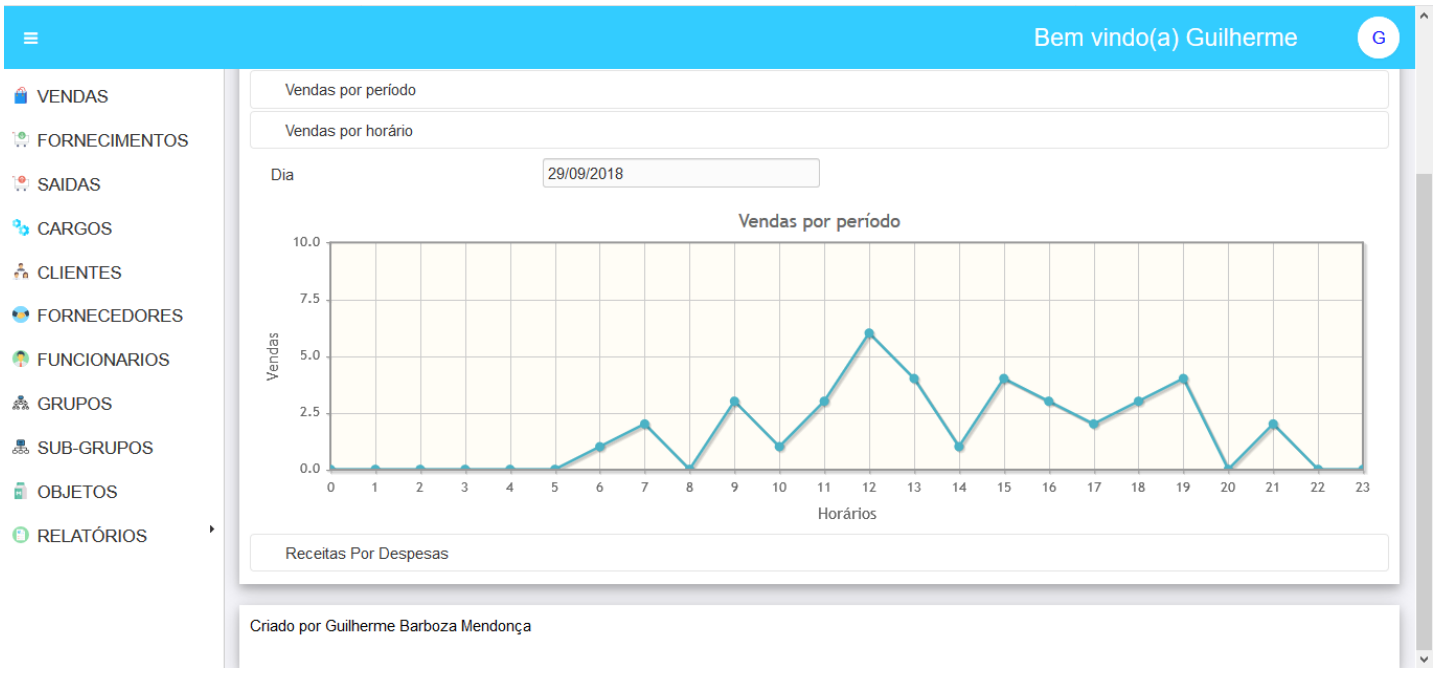
Em um sistema desse contexto, é comum o fluxo de dados inseridos ser constante e intenso, entretanto, um *software* que realiza um gerenciamento simples, superficial e medíocre das informações persistidas, não está aproveitando esses valores ao máximo. Portanto, para que um sistema de gerenciamento seja considerado minimamente bom, é essencial que ele faça uma profunda análise dos dados disponíveis, procurando relacionar fatores e aspectos diferentes e apresentar uma interpretação concisa e objetiva dessas informações, auxiliando, dessa forma, na administração do estabelecimento. Assim, o *software* BlueBuy busca oferecer essa assistência ao apresentar tais dados por meio de relatórios, tabelas e gráficos, assim como pode ser visto nas *Figuras 14* a *18*.

Figura 14 - Relatório das vendas não pagas



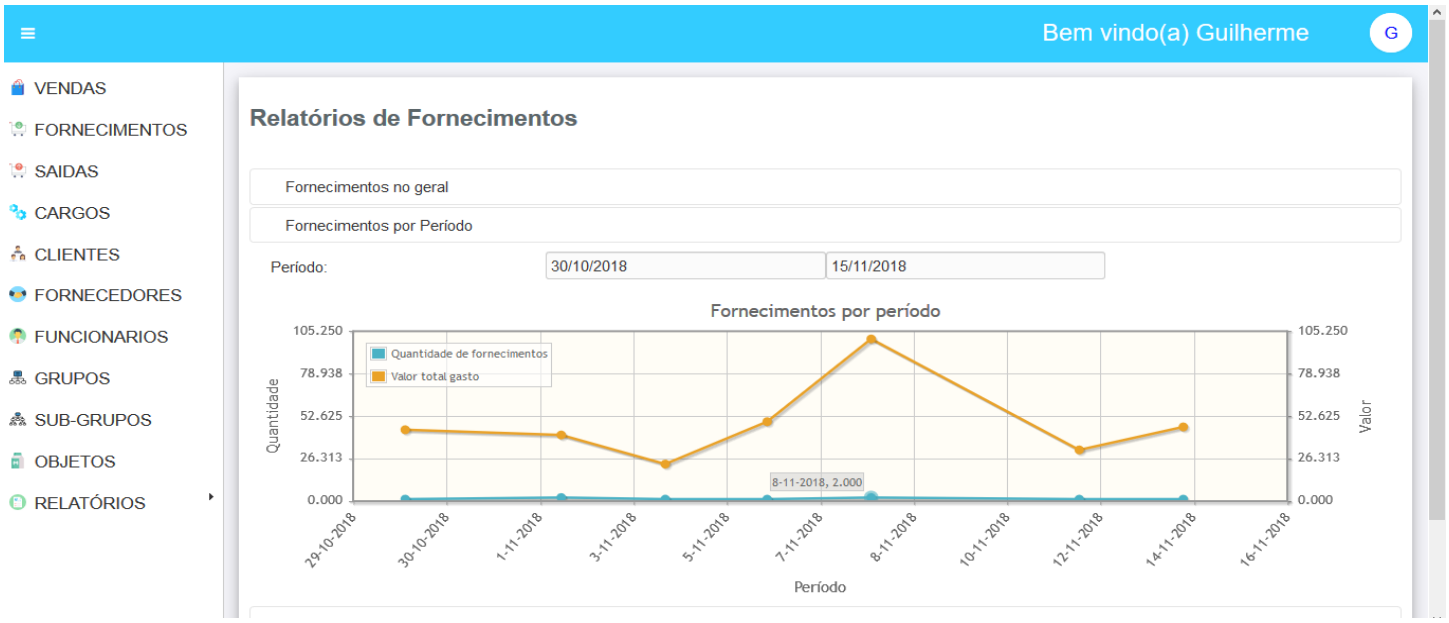
Fonte: Elaborado pelo autor

Figura 15 - Gráfico do número de vendas por horário



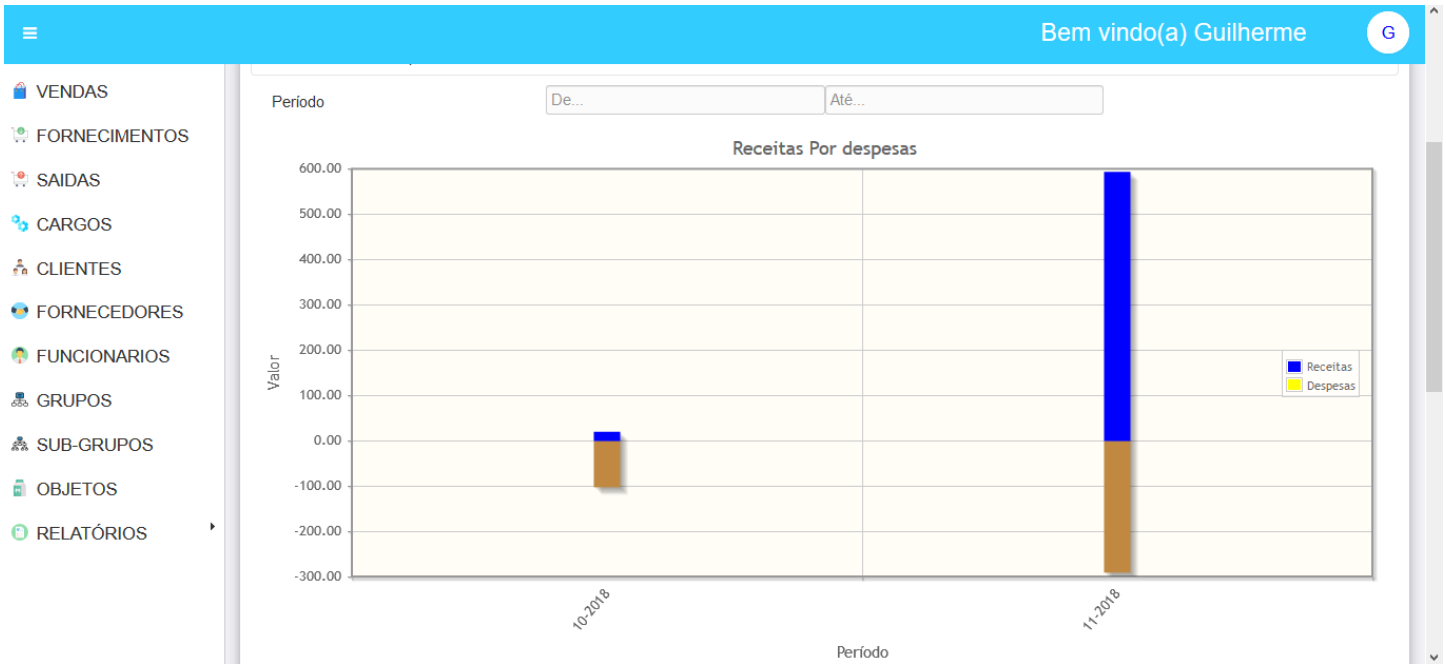
Fonte: Elaborado pelo autor

Figura 16 – Relatório de Fornecimentos por Período



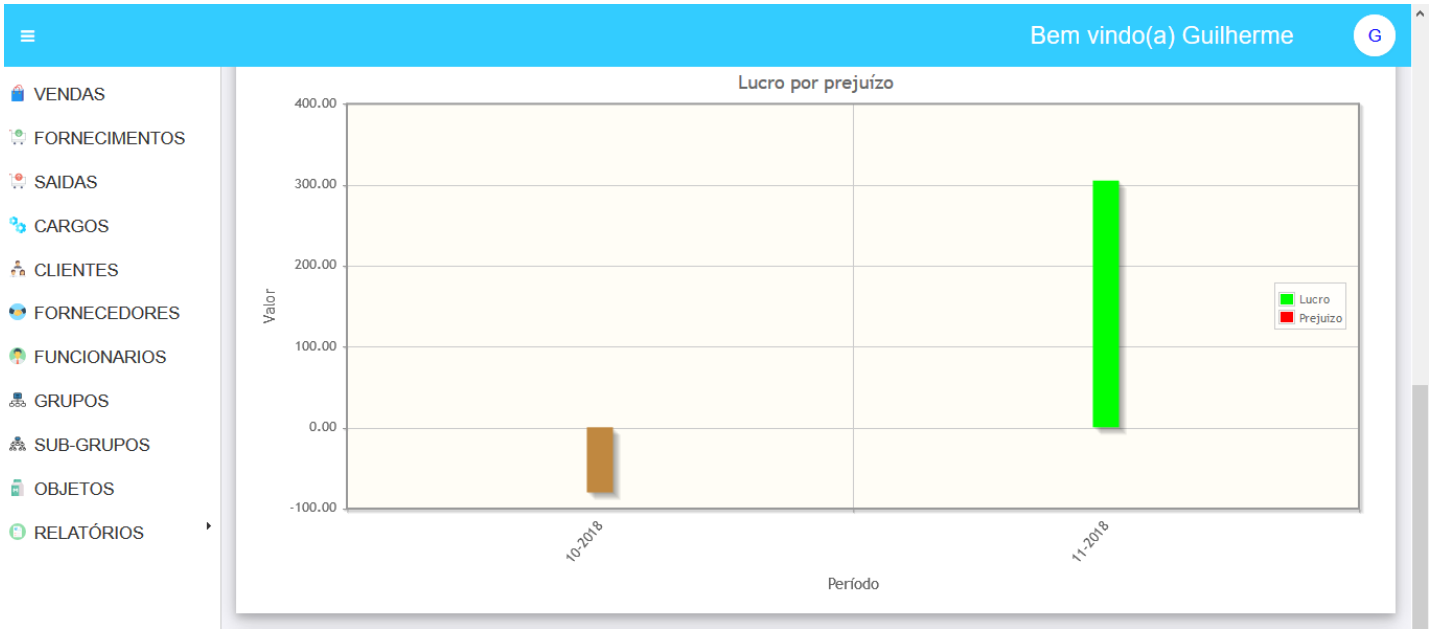
Fonte: Elaborado pelo autor

Figura 17 – Relatório de Receitas por Despesas



Fonte: Elaborado pelo autor

Figura 18 – Relatório de Lucro por Prejuízo



Fonte: Elaborado pelo autor

2. CONCLUSÃO

Este documento buscou dissertar acerca da implementação de um *software* básico de gestão empresarial, cujo principal objetivo é auxiliar no gerenciamento de uma lanchonete ou restaurante ao disponibilizar uma forma objetiva e simples de se realizar tarefas triviais, porém essenciais, como vendas, saídas e fornecimentos, contribuindo ainda na administração dos funcionários, clientes, fornecedores, produtos e mercadorias, oferecendo, dessa forma, uma visão clara acerca das finanças do estabelecimento com a ajuda de relatórios detalhados e informativos.

Para argumentar a respeito da sua construção, foi apresentada primeiramente uma breve introdução e contextualização, vindo seguida pelos principais objetivos tanto do projeto quanto do sistema em si, caracterizando suas premissas. Logo após, foram brevemente citados alguns dados úteis e necessários para se compreender o restante das informações oferecidas ao longo do documento. Subsequentemente, foram analisados e contextualizados os principais materiais e métodos usados no desenvolvimento da aplicação, incluindo o *hardware*, os *softwares*, as linguagens de programação e os

principais *frameworks* usados. Comentou-se ainda a metodologia usada na construção do sistema e, por fim, foram apresentados e analisados os principais resultados finais desse projeto.

Pode-se concluir, portanto, que o *software* resultante desse projeto se mostra confiável e operativo, no que tange a integridade dos dados financeiros, podendo muito bem ser implantado em uma situação real e, eventualmente, se mostrar como uma ferramenta extremamente útil no gerenciamento e na administração de um estabelecimento como uma lanchonete, agilizando todos os processos e auxiliando todas as pessoas envolvidas nessas atividades.

3. TRABALHOS FUTUROS

No que tange o escopo e as funcionalidades do sistema, ainda há várias melhorias e incrementos que poderiam ser realizados:

- Implementar conexões com objetos periféricos, como impressoras, balanças de medição e leitores de código de barras.
- Implementar uma responsividade ao *software*, permitindo que a sua interface se adapte ao dispositivo que a acessa, como computadores, celulares e *tablets*.
- A aplicação pode ser considerada um sistema FINANCEIRO, mas não é um sistema FISCAL. Assim, um possível trabalho futuro seria implementar todos os cálculos fiscais de boleto, nota fiscal, impostos, etc, conectando o sistema à Receita Federal e homologando o sistema como um *software* fiscal.
- O sistema funciona apenas em uma LAN (*Local Area Network* - Rede Local), podendo ser acessado em uma área muito restrita. Logo, um possível projeto futuro é implementá-lo de tal forma que ele esteja disponível em toda a internet, podendo ser acessado e usado em praticamente qualquer lugar.

- Atualmente, o *software* foi desenvolvido para ser implantado em apenas uma única empresa ou estabelecimento. Juntamente com a proposta apresentada logo acima, seria interessante implantar no sistema um suporte a várias empresas, cada uma com seus respectivos clientes, funcionários, vendas, etc.

REFERÊNCIAS BIBLIOGRÁFICAS

DINIZ, Eduardo Henrique. **Uso da web nos serviços financeiros**. 2005. Disponível em: <https://bibliotecadigital.fgv.br/dspace/handle/10438/3197>. Acessado em: 04/09/2018

SAMPAIO, Juliana. **O que é PDV e o que seu negócio perde sem ele**. 2017. Disponível em: <https://www.erpflex.com.br/blog/o-que-e-pdv>. Acessado em: 07/09/2018

REDAÇÃO. **Desenvolvimento de aplicações web**: tire suas dúvidas sobre o assunto!. 2018. Disponível em: <https://www.impacta.com.br/blog/2018/01/05/desenvolvimento-de-aplicacoes-web-tire-suas-duvidas-sobre-o-assunto/>. Acessado em: 17/10/2018.

SALES, Jomar. **Por que desenvolver aplicações para Web?** 2009. Disponível em: <http://jomarsales.blogspot.com/2009/01/por-que-desenvolver-aplicacoes-para-web.html>. Acessado em: 17/10/2018.

MACORATTI, José Carlos. **Desenvolvendo para desktop ou para Web?** 2018. Disponível em: <http://www.macoratti.net/vbn_dkwb.htm>. Acessado em: 17/10/2018.

GODOI, Matheus. **O que é uma IDE? Qual a diferença para um editor de texto?**. 2016. Disponível em: <https://esolutebrasil.com.br/blog/o-que-e-uma-ide-qual-a-diferenca-para-um-editor-de-texto/>. Acessado em: 07/09/2018

MARQUES, Thais. **O que é uma IDE? Qual a diferença para um editor de texto?**. 2018. Disponível em: <https://esolutebrasil.com.br/blog/o-que-e-uma-ide-qual-a-diferenca-para-um-editor-de-texto/>. Acessado em: 07/09/2018

RICARDO, Luiz. **Instalando, configurando e usando o Maven para gerenciar suas dependências e seus projetos Java**. 2014. Disponível em: <http://luizricardo.org/2014/06/instalando-configurando-e-usando-o-maven-para-gerenciar-suas-dependencias-e-seus-projetos-java/>. Acessado em: 08/09/2018

REIS, Rodrigo Quites. **CVS**:Concurrent Version System. 2006. Disponível em: <http://www.ufpa.br/cdesouza/teaching/labes/cvs.pdf>. Acessado em: 08/09/2018

HIBERNATE. **Hibernate ORM**: Your relational data. Objectively. 2018. Disponível em: <http://hibernate.org/orm/> Acessado em: 11/09/2018

LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. **Programação Java para a Web**: Aprenda a desenvolver uma aplicação financeira pessoal com as ferramentas mais modernas da plataforma Java. 2.ed. São Paulo: Novatec. 2017. 677p.

SCHALK, Chris. **Introduction to JavaServer Faces**: What is JSF. 2005. Disponível em: <https://www.oracle.com/technetwork/topics/index-090910.html> Acessado em:20/09/2018

ÇIVICI, Çağatay. **PrimeFaces**: User guide. 2018. Disponível em: <https://www.primefaces.org/showcase/> Acessado em: 20/09/2018

MEDIA, Traversy. **WHAT Is MVC?**: Simple Explanation. 2017. Disponível em: <https://www.youtube.com/watch?v=pCvZtjoRq1I>. Acessado em: 22/09/ 2018.

JARVIS, Dave. **WHAT is MVC, really?**. 2011. Disponível em: <https://softwareengineering.stackexchange.com/questions/127624/what-is-mvc-really>. Acessado em: 22/09/2018

MEDEIROS, Higor. **Introdução ao padrão MVC**. 2013. Disponível em: < https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308> Acessado em: 26/10/2018

SANTOS, Raphael. **O real motivo do uso de Metodologias no Desenvolvimento de Softwares**. 2011. Disponível em: <http://www.techtudo.com.br/platb/desenvolvimento/2011/09/09/o-real-motivo-do-uso-de-metodologias-no-desenvolvimento-de-softwares/> Acessado em: 22/09/2018

MEDEIROS, Higor. **Introdução ao Extreme Programming (XP)**. 2013. Disponível em: <https://www.devmedia.com.br/introducao-ao-extreme-programming-xp/29249> Acessado em: 22/09/2018

REIS, Daniel Fonseca. **Conceitos básicos sobre Metodologias Ágeis para Desenvolvimento de Software (Metodologias Clássicas x Extreme Programming)** . 2009. Disponível em: <https://www.devmedia.com.br/conceitos-basicos-sobre-metodologias-ageis-para-desenvolvimento-de-software-metodologias-classicas-x-extreme-programming/10596> Acessado em: 22/09/2018

CAELUM. **Java e Orientação a Objetos**. 2017. Disponível em: <https://www.caelum.com.br/apostila-java-orientacao-objetos/o-que-e-java/#java> Acessado em: 25/10/2018

NUNES, Felipe. **Android MVC x MVP x MVVM:** qual Pattern utilizar - Parte 1. 2017. Disponível em: <https://medium.com/@FilipeFNunes/android-mvc-x-mvp-x-mvvm-qual-pattern-utilizar-parte-1-3defc5c89afd> Acessado em: 26/10/2018

FOGASSIA, Shailesh. **Save Server Response to File Using Python in Open Event Android App Generator**. 2017. Disponível em: < https://blog.fossasia.org/save-server-response-to-file-using-python-in-open-event-android-app-generator/> Acessado em 26/10/2018