UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLOGIA DA INFORMAÇÃO
CURSO DE LICENCIATURA EM COMPUTAÇÃO E INFORMÁTICA

GUSTAVO DE ARAÚJO COSTA

UM SISTEMA DE CONTROLE DE VENDAS DE PRODUTOS PARA UMA MICROEMPRESA

ANGICOS
2021

GUSTAVO DE ARAÚJO COSTA

UM SISTEMA DE CONTROLE DE VENDAS DE PRODUTOS PARA UMA MICROEMPRESA

Trabalho de Conclusão de Curso apresentado a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Licenciado em Computação e Informática, sob orientação do Prof. Dr. Francisco de Assis Pereira Vasconcelos de Arruda.

ANGICOS

2021

C837s Costa, Gustavo de Araújo.
Um sistema de controle de vendas de produtos para uma microempresa / Gustavo de Araújo Costa. - 2021.
47 f. : il.

Orientador: Francisco de Assis Pereira Vasconcelos de Arruda.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Computação e Informática, 2021.

1. Engenharia de software. 2. Sistema de informação. 3. Administração financeira. I. Arruda, Francisco de Assis Pereira Vasconcelos de, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automáto em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Angicos
Bibliotecário: Helder Romero Maia Duarte
CRB: 15/673

GUSTAVO DE ARAÚJO COSTA

UM SISTEMA DE CONTROLE DE VENDAS DE PRODUTOS PARA UMA MICROEMPRESA

Trabalho de Conclusão de Curso apresentado a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Licenciado em Computação e Informática, sob orientação do Prof. Dr. Francisco de Assis Pereira Vasconcelos de Arruda.

Defendido em: 17/11/2021

BANCA EXAMINADORA

_____

Prof. Dr. Francisco de Assis Pereira Vasconcelos de Arruda (UFERSA)
Orientador-Presidente

_____

Prof.ª Dr.ª Thatiana Cunha Navarro de Souza (UFERSA)
Membro Examinador

_____

Prof.ª Sara Guimarães Negreiros (UFERSA)
Membro Examinador

Aos alunos que sentem-se perdidos, desorientados e subestimados. Aos estimados professores que clamam por uma educação para todos, que suas mentiras não mais reverberem em mentes esperançosas.

AGRADECIMENTOS

A meu orientador, Francisco de Assis Pereira Vasconcelos de Arruda, agradeço imensa e profundamente o fato de ter aceitado me guiar durante o desenvolvimento do presente trabalho e, acima de tudo, ter acreditado que eu poderia concluí-lo. Sua orientação foi irretocável.

A minha família, que à sua maneira sei que me ama — Maria Vilma de Araújo e João Batista Raimundo da Costa.

Agradeço João Vitor Costa Cardoso por ter me apresentado o universo da programação, um feliz acaso como foi nossa amizade. Sua presença em toda a minha caminhada acadêmica tornaram os dias mais agradáveis.

A Maria Paula da Silva de Souza e Robson Aquino de Medeiros, caros colegas da graduação, sou grato pelas palavras de incentivo em momentos de incerteza e insegurança. Muito embora nossos caminhos não mais se cruzem, guardo com carinho as lembranças de nossos anos na universidade.

Adicionalmente, agradeço indistintamente os colegas Eclesivaldo Pedro Costa, João Érico Bezerra de Sena e José Maria Neto por momentos felizes durante a graduação.

As professoras Sara Guimarães Negreiros e Thatiana Cunha Navarro de Souza, membros da banca avaliadora, agradeço por suas considerações pertinentes e assertivas sobre os pontos de melhoria desta monografia.

Acima de tudo, sou grato ao passar do tempo pela serenidade, pelo discernimento e equilíbrio.

“De vez em quando percebia um gavião que voava das grandes rochas acima de sua cabeça, descrevendo em silêncio círculos imensos. O olhar de Julien seguia maquinalmente a ave de rapina. Seus movimentos tranquilos e possantes o impressionavam; invejava aquela força, invejava aquele isolamento. Havia sido esse o destino de Napoleão: seria o seu, um dia?”, Stendhal em O vermelho e o negro (1830).

RESUMO

Este trabalho propõe-se a apresentar o processo de desenvolvimento de uma solução de *software* para automatização de alguns aspectos da administração financeira de uma microempresa. A demanda do cliente foi de acompanhar o lucro proveniente da venda de produtos, tarefa outrora registrada inconsistentemente e manualmente em papel. Para tanto, foram utilizados preceitos da Teoria da Atividade para elucidar atividades que posteriormente foram decompostas em requisitos funcionais e não funcionais do sistema. A UML foi utilizada para documentar etapas do processo de desenvolvimento, tais quais a modelagem da aplicação e a comunicação de aspectos abstratos de sua implementação. A aplicação foi desenvolvida na linguagem de programação Java, com elementos do ecossistema Spring, versionado com Git e armazenado no GitHub. Por fim, a aplicação foi implantada em produção por meio da plataforma Heroku.

Palavras-chave: engenharia de *software*; sistema de informação; administração financeira.

ABSTRACT

This work aims to present the development process of a software solution for automating some aspects of the financial administration of a microenterprise. The costumer's demand was to track down the profit originating from the sale of products, a task that was once done inconsistently and manually on paper. Therefore, precepts of the Activity Theory were used to elucidate activities that later were decomposed into the functional and non-functional requirements of the system. The UML was used to document stages of the development process, such as application modeling but also to communicate abstract aspects of its implementation. The application was developed in the Java programming language, with elements of the Spring ecosystem, versioned with Git and stored on GitHub. Finally, the application was production deployed using the Heroku platform.

Keywords: software engineering; information system; financial administration.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

API *Application Programming Interface*

CLI *Command-line Interface*

CSS *Cascading Style Sheets*

HTML *Hypertext Markup Language*

IDE *Integrated Development Environment*

JPA *Java Persistence API*

MVC *Model–View–Control*

SQL *Structured Query Language*

STS *Spring Tool Suite*

TIC Tecnologia da Informação e Comunicação

UML *Unified Modeling Language*

SUMÁRIO

1. CONSIDERAÇÕES INICIAIS

Em uma sociedade centrada, sobretudo, no capital (DRUCKER, 2011, p. 17), a habilidade de gerenciar sabiamente o patrimônio financeiro torna-se uma competência altamente desejável, quer seja esse patrimônio individual ou empresarial. O sucesso econômico depende de uma combinação entre esforço, habilidade (CLASON, 2002, p. 4) e discernimento entre um equívoco financeiro e uma decisão acertada (BROOKS, 2015, p. 31). Gerenciar um patrimônio financeiro, nas palavras de Brooks (2015, p. 31, tradução livre), "inclui muitas atividades que criam ou preservam o valor econômico dos ativos de um indivíduo, de um pequeno negócio ou de uma corporação".

É fato observável que uma empresa que não produz diretamente o produto que comercializa encontra-se sempre na posição de intermediária entre seu fornecedor e o cliente final, ainda que não seja esse seu objetivo. O ciclo do dinheiro apresenta-se em uma situação análoga, onde o intermediário (comumente um banco) obtém lucro pelo serviço realizado em unir credor e mutuário. A empresa que nessa situação não toma nota de quanto lucra pode colocar-se em uma posição periclitante.

Preservar ou criar valor econômico é uma assertiva inflexível. Sabe-se que em situações reais, os administradores são confrontados com decisões complexas, considerando múltiplos fatores. Em pequenos negócios, a tarefa de administrar as finanças corriqueiramente recai sob mãos por vezes despreparadas, ainda que muitas vezes hábeis. Naturalmente, esses administradores se valem de conhecimentos empíricos para desempenhar essa função. Todavia, muito embora as práticas empíricas possam ser de grande valia para um negócio, o conhecimento sistematizado pode auxiliar na tomada de decisões mais eficazes.

Diante do exposto, neste trabalho é proposto o desenvolvimento de uma solução de *software* capaz de fornecer a um cliente de um pequeno negócio a autonomia na administração de suas finanças, em especial no registro de compra e venda de produtos.

1.1. Motivação e caracterização do problema

O cliente alvo da solução de *software* proposta neste trabalho possui uma microempresa e atualmente realiza a administração de suas finanças de forma manual, contabilizando em um caderno de papel quanto deve lucrar em cada produto para cobrir o preço de suas mercadorias.

Com o aumento do volume de vendas, o cliente passou a não realizar ações corriqueiras e necessárias para a administração das finanças de seu negócio, tais quais o

cálculo do lucro sobre a venda de produtos e o registro de vendas realizadas, salvo em caso de pagamento a prazo. Visualizou-se que a administração das finanças apresenta-se como de grande importância para a manutenção do negócio, sendo imprescindível para acompanhar a geração de valor a longo prazo.

Pela não realização das tarefas necessárias para a administração das finanças de seu negócio, o cliente enfrenta atualmente problemas para visualizar suas vendas e o lucro que obtém com elas.

De modo a solucionar tais problemas, o sistema proposto neste trabalho buscou automatizar uma parcela do complexo processo da administração financeira de uma empresa, provendo os meios para registrar as vendas realizadas e o lucro que as segue.

1.2. Objetivos

A formulação de objetivos implica "decidir com precisão o que se visa com o trabalho sob dois aspectos: geral e específico" (LAKATOS & MARCONI, 2003, p. 247).

Enquanto a ideia de um objetivo geral comunica o próprio propósito do trabalho, estando vinculado diretamente ao seu significado, um objetivo específico tem "âmbito mais restrito, compreende etapas intermediárias que, sob aspectos instrumentais, permite o objetivo geral" (LAKATOS & MARCONI, 2003, p. 247). Dessa forma, estão listados adiante o objetivo geral e objetivos específicos deste trabalho.

1.2.1. Objetivo Geral

Automatizar um aspecto da administração financeira de uma microempresa por meio do desenvolvimento de uma solução de *software*.

1.2.2. Objetivos Específicos

- Elicitar os requisitos para o desenvolvimento do sistema.
- Desenvolver, com base nos requisitos definidos, um sistema que atenda às necessidades do cliente.
- Documentar o desenvolvimento do sistema.
- Implantar o *software* desenvolvido.
- Validar o *software* desenvolvido junto ao cliente.

2. FUNDAMENTAÇÃO TEÓRICA

O trabalho científico deve "dar conta dos elementos necessários para o desenvolvimento do raciocínio demonstrativo, recorrendo assim a um volume de fontes suficiente para cumprir essa tarefa" (SEVERINO, 2013, p. 115). Consta nesta seção a fundamentação teórica nos campos da Administração e sistemas de informação que alicerçam a construção desta monografia.

Realiza-se na subseção 2.1 um breve apanhado histórico da Administração e as raízes do pensamento administrativo. Discorre sobre as contribuições de pensadores renomados para esse ramo do conhecimento e a importância do estudo de fontes históricas para compreender os processos administrativos.

Discute-se na subseção 2.2 como a inserção de Tecnologias da Informação e Comunicação pode ser uma solução para problemas corriqueiros da administração de empresas.

2.1. Breve histórico da administração de empresas

A história da Administração remonta à Antiguidade, posto que "pelos últimos três mil anos, pessoas vêm se preocupando com problemas da administração" (WITZEL, 2012, p. 1, tradução livre). Já no Mundo Antigo o povo se preocupava com questões de organização e liderança; na Idade Média, tanto no Ocidente quanto no Oriente, pessoas começaram a ponderar riscos, a pensar estrategicamente e refletir sobre o papel dos negócios na sociedade. Todo esse processo histórico culmina na aurora da Administração Científica, com o despontar da Revolução Industrial (WITZEL, 2012, p. 1).

Para realizar um apanhado histórico sobre as raízes do pensamento administrativo, ainda que breve relato, deve-se olhar para a "definição do que se entende por pensamento administrativo para então explicar o que faz sua história tão digna de nota" (WITZEL, 2012, p. 8, tradução livre).

Revisitar tal histórico significa recordar que "a história da verdadeira administração e suas ideias é sempre uma busca por soluções análogas e novas avaliações sobre decisões tomadas" (MARSHEV, 2021, p. 1–2). Assim sendo, se é factível obter a solução para um problema atual a partir da análise de outro semelhante, prova-se a importância da matéria enquanto fonte de conhecimento.

Negligenciar a história, por outro lado, prova-se uma decisão insensata e improdutiva. Ignorar o conhecimento acumulado, desconsiderando tudo o que foi produzido, leva somente a um ciclo infinito de redescoberta científica, um desperdício de força intelectual. Uma situação do gênero é descrita por Marshev (2021, p. 1–2, tradução livre), onde:

> administradores modernos e até mesmo teóricos da administração começaram a esquecer, gradualmente, as ideias de Charles Babbage (século dezenove) e Frederick Taylor (começo do século vinte) sobre o estudo do trabalho e hoje apresentamos ideias sobre a alocação racional do tempo do administrador como uma descoberta científica; gradualmente, começamos a esquecer a lei de Henri Fayol sobre as funções administrativas e hoje estamos lutando para explicar a viabilidade das estruturas do gerenciamento de uma organização funcional.

Logo, partindo do entendimento de Witzel (2012, p. 8) sobre o tema, busca-se compreendê-lo sob a ótica acadêmica. Tal escolha justifica-se em razão da natureza do presente trabalho, a fim de manter-se o rigor acadêmico. Além disso, considera-se o caráter do relato, que, em sua brevidade, torna impraticável a inclusão de um segmento sobre o conhecimento empírico, embora reconheça sua importância e validade para o campo científico[1].

Dentro do cânone literário na área de Administração, destaca-se o trabalho realizado por Weber no campo da burocracia na busca por eficiência e eficácia. De maneira equivalente, destaca-se o trabalho de Taylor[2] por sua contribuição para a solidificação de uma metodologia científica no campo da governança, a fim de promover o aprimoramento da eficiência de tarefas realizadas em ambiente fabril. Semelhantemente, destaca-se Fayol pela criação de um modelo racional e organizado de gerência, buscando, em um sentido amplo, o aperfeiçoamento da governança como um todo.

O trabalho de Weber sobre o modelo burocrático de gestão descreve um mundo em pleno funcionamento, tanto no âmbito público quanto no privado (WATERS & WATERS, p. 73). Sob o crivo do modelo burocrático, a natureza da governança é dividida em uma hierarquia rígida, "existe uma distribuição fixa das atividades regularmente necessárias para realizar os fins do complexo burocraticamente dominado, como deveres oficiais", "os poderes de mando, necessários para cumprir estes deveres, estão também fixamente distribuídos, e os meios coativos (físicos, sacros ou outros) que eventualmente podem empregar estão também fixamente delimitados por regras", "para cumprimento regular e contínuo dos deveres assim distribuídos e o exercício dos direitos correspondentes criam-se providências planejadas,

[1] O autor (WITZEL, 2012, p. 8) indica duas formas possíveis para a compreensão do pensamento administrativo, são elas: 1) sob a ótica acadêmica, a partir de teorias coerentistas ou sistemas da administração; 2) ao "aumentar a abrangência do termo e nos referirmos, mais genericamente, ao "pensar sobre a administração", ideias sobre o significado, o propósito, a função e as tarefas da administração que são importantes e relevantes, mas não necessariamente formam base suficiente para dar vida a uma teoria".

[2] Embora muitas vezes citado como "fundador da administração científica", Marshev (2021, p. 415) ressalta a tentativa de outros em aplicar métodos científicos a fim de solucionar problemas de governança administrativa.

contratando pessoas com qualificação regulamentada de forma geral" (WEBER, 1999, p. 198).

Essa divisão hierárquica, teoricamente, contribui para a eficiência e eficácia da instituição. Apesar da ampla aceitação de sua visão sobre o modelo burocrático, uma outra interpretação sugere que o propósito de seu trabalho não era aconselhar gestores nos caminhos da eficiência e da eficácia, mas "abordar um debate teórico carregado por Rousseau, Hegel e Marx sobre a natureza básica da dominação em nossa sociedade" (WEISS, 1983, p. 242, tradução livre). A despeito disso, quer tenham sido essas ou não suas intenções, suas reflexões constituem contribuições inestimáveis ao pensamento administrativo.

O modelo de gestão proposto por Fayol traça quatorze (14) princípios norteadores para práticas administrativas. Cunhados a partir do exercício da prática administrativa durante anos, atuam como guias aos gestores. Posto que não são um conjunto rígido de regras como as leis jurídicas ou as ciências naturais, mas são flexíveis (MARSHEV, 2021, p. 435–436). Para fins de esclarecimento, são listados adiante tais princípios[3] (MARSHEV, 2021, p. 436): 1) divisão do trabalho; 2) autoridade e responsabilidade; 3) unidade de comando; 4) unidade de direção; 5) disciplina; 6) subordinação dos interesses pessoais aos interesses comuns; 7) remuneração; 8) centralização; 9) hierarquia; 10) ordem; 11) equidade; 13) iniciativa; 14) união da equipe.

2.2. Administração e tecnologia

As Tecnologias da Informação e Comunicação (TICs) estão presentes em virtualmente todos os espaços de nossas vidas. Em um cenário onde os meios tecnológicos evoluem em ritmo acelerado, muitas empresas nasceram em virtude dessa mesma evolução, enquanto outras vivenciaram melhorias em seus processos internos. Quer tenham a tecnologia em si como produto final ou a utilizem como meio para otimização de rotinas, "as mudanças na realidade exigem uma mudança na postura do gestor" (DRUCKER, 2004, p. 21) em adequação a uma sociedade altamente competitiva.

Não data de hoje a preocupação com a produtividade, com a eficiência e com a eficácia — essencialmente, atenção à forma como se produz. Em fato, a reflexão sobre a otimização do trabalho foi objeto da Administração Científica de Taylor, que com "seus esforços em estudar o trabalho manual formaram o alicerce da riqueza dos países desenvolvidos" (DRUCKER, 2004, p. 24, tradução livre).

[3] Listados como: 1) The division of labor; 2) The power; 3) Discipline; 4) The unity of management (command); 5) Unity of leadership; 6) Submission of private interests to the general; 7) Remuneration; 8) Centralization; 9) Hierarchy; 10) Order; 11) Justice; 12) The constancy of the staff; 13) The initiative; 14) The unity of the staff.

Atualmente, frente a mudanças constantes nos campos da tecnologia, ciência e economia, “gestores do mundo inteiro estão conscientes de novos conceitos e novas ferramentas da administração” graças à revolução da informação (DRUCKER, 2004, p. 21–22). Nesse sentido, as “TICs permitem que indústrias desenvolvam e mantenham uma vantagem competitiva no mercado global” (KIRMANI; WANI; SAIF, 2015, p. 2, tradução livre).

Ciente que o “valor de uma empresa reside em sua habilidade de gerar receita” (BRIGHAM & EHRHARDT, 2019, p. 9), a incorporação de tecnologias da informação e comunicação aos serviços administrativos traz benefícios como diminuição de gastos, armazenamento e proteção de dados, além da possibilidade de integração com outros sistemas de informação (KIRMANI; WANI; SAIF, 2015, p. 4–5).

Paralelamente, a ausência de um sistema de informação para gestão financeira pode acarretar prejuízos em relação ao planejamento e controle de gastos, influenciando negativamente a administração orçamentária (DIAMOND & KHEMANI, 2015, p. 1).

Em última análise, a implementação de um sistema de informação para gestão financeira pode ser uma solução para problemas corriqueiros da administração, proporcionando ao gestor um conjunto de informações úteis ao planejamento de estratégias comerciais, no processo de tomada de decisão e acompanhamento do fluxo de caixa.

3. MATERIAIS E MÉTODOS

Diz-se que o método científico é um conjunto de "atividades sistemáticas e racionais que, com maior segurança e economia, permite alcançar o objetivo — conhecimentos válidos e verdadeiros —, traçando o caminho a ser seguido, detectando erros e auxiliando as decisões" (LAKATOS & MARCONI, 2003, p. 83). Isto posto, nesta seção encontram-se os materiais e métodos utilizados no desenvolvimento da solução de *software*.

Na subseção de nº 3.1 é discutido o uso de uma metodologia de desenvolvimento de *software* no projeto.

Na subseção de nº 3.2 elenca-se o ferramental empregado no decorrer do processo de desenvolvimento. São listadas ferramentas ligadas ao *front-end*, ao *back-end*, à plataforma de implantação, aos serviços de banco de dados e utilitários.

Na subseção de nº 3.3 trata-se a escolha da linguagem de programação adotada e aspectos do ecossistema Spring com Java.

Na subseção de nº 3.4 é demonstrada a elicitação de requisitos do sistema e a adoção de preceitos da Teoria da Atividade como forma de minimizar problemas nessa etapa.

Na subseção de nº 3.5 apresenta-se a modelagem conceitual do sistema.

Na subseção de nº 3.6 é relatado o percurso percorrido para a criação do protótipo da interface de usuário apresentado ao cliente. Quase simetricamente, a subseção 3.7 esclarece o trajeto concluído para a criação do protótipo navegável de alta fidelidade submetido ao crivo do cliente.

Na subseção de nº 3.8 discorre-se sobre a implementação da solução de *software*.

Na subseção de nº 3.9 é documentado o débito técnico adquirido, identificando-o para o aprimoramento futuro do sistema.

3.1. Metodologia de desenvolvimento de *software*

Considerando o desenvolvimento de um sistema de informação que tente resolver um aspecto da gestão financeira de problemas corriqueiros e considerando que o cliente é parte central no processo de desenvolvimento de *software*, buscou-se mantê-lo envolvido de modo a assegurar a entrega de um produto condizente com suas necessidades. O envolvimento do cliente, o proprietário da microempresa em que o sistema foi implantado, deu-se a partir de reuniões periódicas para levantamento dos requisitos, que foram documentadas em atas, exibidas de forma resumida no Quadro 1.

Nenhuma metodologia de desenvolvimento foi fielmente adotada. Conquanto, foram incorporados aspectos comuns ao Scrum. Tal qual no desenvolvimento dessa solução de *software*, no Scrum as iterações acontecem uma após a outra e, ao seu fim, busca-se sempre um avanço no estado do produto (SCHWABER, 2004, p. 5). Ademais, a funcionalidade decorrente do final da iteração é submetida ao crivo do cliente ou interessados (*stakeholders*) para identificação de possíveis pontos de melhoria (SCHWABER, 2004, p. 6).

Quadro 1 - ATAS DAS REUNIÕES COM O CLIENTE

Data	Atividade	Comentário	Situação
10/07/2021	Reunião inicial	Reunião inicial a fim de compreender o problema principal	Concluído
28/07/2021	Apresentação da interface	Apresentação da interface ao usuário para localização de possíveis pontos de melhoria	Concluído
05/08/2021	Apresentação do protótipo	Apresentação do protótipo clicável ao usuário para localização de possíveis pontos de melhoria	Concluído
15/10/2021	Implantação do produto	Implantação do produto no ambiente de produção	Concluído
26/10/2021	Validação do produto	Validação do produto final junto ao cliente	Concluído

Fonte: autoria própria (2021).

3.2. Ferramentas utilizadas

Optou-se nesse trabalho por implementar um sistema *web* para o desenvolvimento da solução de *software* proposta nos objetivos gerais. Para a implementação desse sistema foram usados como ferramentas: a Linguagem de Marcação de Hipertexto (HTML - *Hypertext Markup Language*), as Folhas de Estilo em Cascata (CSS - *Cascading Style Sheets*), o Spring Tool Suite como ambiente de desenvolvimento, o ecossistema Spring para desenvolvimento do *back-end* e Maven para gerenciamento de pacotes. O ferramental é detalhado a seguir.

A Linguagem de Marcação de Hipertexto sinaliza os elementos em um documento para que esses possam funcionar como hipertexto (PFAFFENBERGER et al., 2004, p. 5, tradução livre). A linguagem é formada por uma série de símbolos que informam aos navegadores a estrutura das páginas por eles renderizadas. A interface do usuário da aplicação utiliza majoritariamente HTML para estruturar suas páginas.

Para separar a estrutura das páginas de sua apresentação, utilizam-se Folhas de Estilo em Cascata. As folhas de estilo surgiram como uma forma de minimizar o esforço despendido na manutenção da aparência em páginas da *web*[4] (PFAFFENBERGER et al., 2004, p. 10). Ao mover a responsabilidade de gerenciar a aparência do documento para um arquivo externo, tem-se um significativo ganho em produtividade, pois podemos compartilhar estilos comuns a múltiplas páginas, além de, potencialmente, realizar a manutenção em um único arquivo.

[4] Para uma visão mais detalhada do histórico das folhas de estilo e sua importância para a manutenção de páginas *web* modernas, Pfaffenberger et al. (2004, p. 10-14).

O Spring Tool Suite 4 (STS4) foi escolhido como Ambiente de Desenvolvimento Integrado (IDE – *Integrated Development Enviroment*). Baseado em outro ambiente de desenvolvimento já consolidado (a IDE Eclipse), o STS provê um senso de familiaridade ao desenvolvedor, além de facilidades ao trabalhar em projetos do ecossistema Spring. Para facilitar a codificação da interface apresentada ao usuário foi utilizado o processador de modelos (*template engine*) Thymeleaf.

O versionamento do código fonte da aplicação foi feito com Git por meio da ferramenta Git bash, emulando um terminal para esse fim. O código fonte escrito era então salvo em um repositório no GitHub.

Foi empregado um sistema de banco de dados relacionais para armazenar os dados da aplicação. A conexão local foi feita com base em recursos provisionados pelo MySQL Workbench e elementos do ecossistema Spring como Spring Data JPA e Hibernate. Para realizar a conexão com o banco de dados em versão de produção, foi adicionado ao serviço de implantação o banco de dados PostgreSQL provisionado pela plataforma Heroku. Middleton e Schneeman (2013, p. 39) dizem que, com o crescimento da plataforma, houve um aumento na demanda por um serviço de bancos de dados, culminando na criação do Heroku PostgreSQL.

A mudança do MySQL para PostgreSQL ocorreu em virtude da ampla documentação que a plataforma oferece para que sejam realizadas as configurações necessárias para a conexão com a base de dados, além de minimizar o risco potencial de aprisionamento tecnológico (*vendor lock-in*) com o uso do MySQL, como alertam Middleton e Schneeman (2013, p. 39).

A aplicação foi implantada por meio da plataforma Heroku. Todo o processo foi feito utilizando Heroku CLI, diretamente do terminal emulado pelo Git bash.

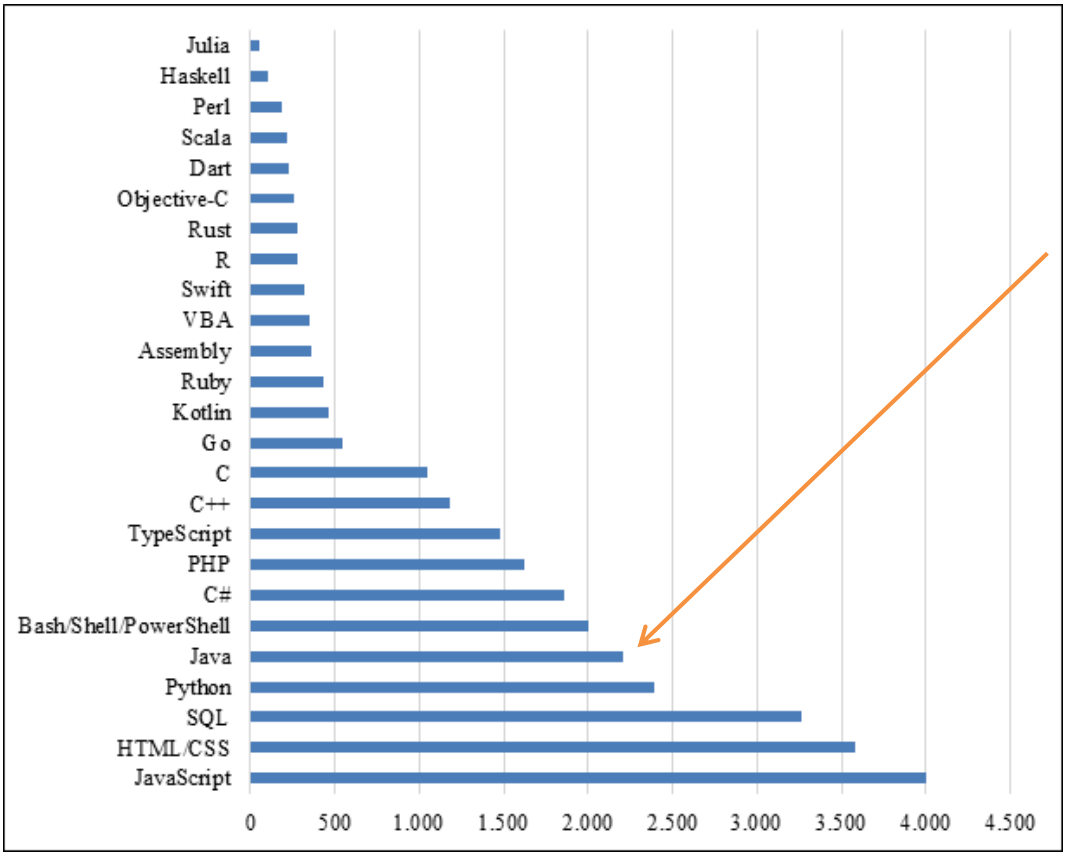
3.3. Linguagem de programação adotada

A linguagem de programação Java foi concebida para ser independente de qualquer plataforma (KNUDSEN & NIEMEYER, 2000, p. 9). O código fonte escrito é convertido em um formato universal compreendido por uma máquina virtual

Quando surgiu, em 1996, Java foi criticada por ser uma linguagem interpretada, o que poderia prejudicar sua velocidade de execução, em comparação com uma linguagem compilada como C. Apesar do fato, apresentava a seu favor o gerenciamento dinâmico de memória, uma sintaxe simplificada e o coletor de lixo (*garbage collector*) automático (KNUDSEN & NIEMEYER, 2000, p. 15).

Considerando seu lançamento em 1996, Java permanece como uma linguagem amplamente utilizada e consolidada no mercado. Conforme levantamento feito pelo fórum Stack Overflow (2021), a linguagem ocupa a quinta posição no quesito popularidade entre os 57 378 participantes da pesquisa. A distribuição quantitativa das respostas obtidas pelo levantamento está presente no Gráfico 1.

Gráfico 1 - CONCENTRAÇÃO DE USUÁRIOS POR LINGUAGEM

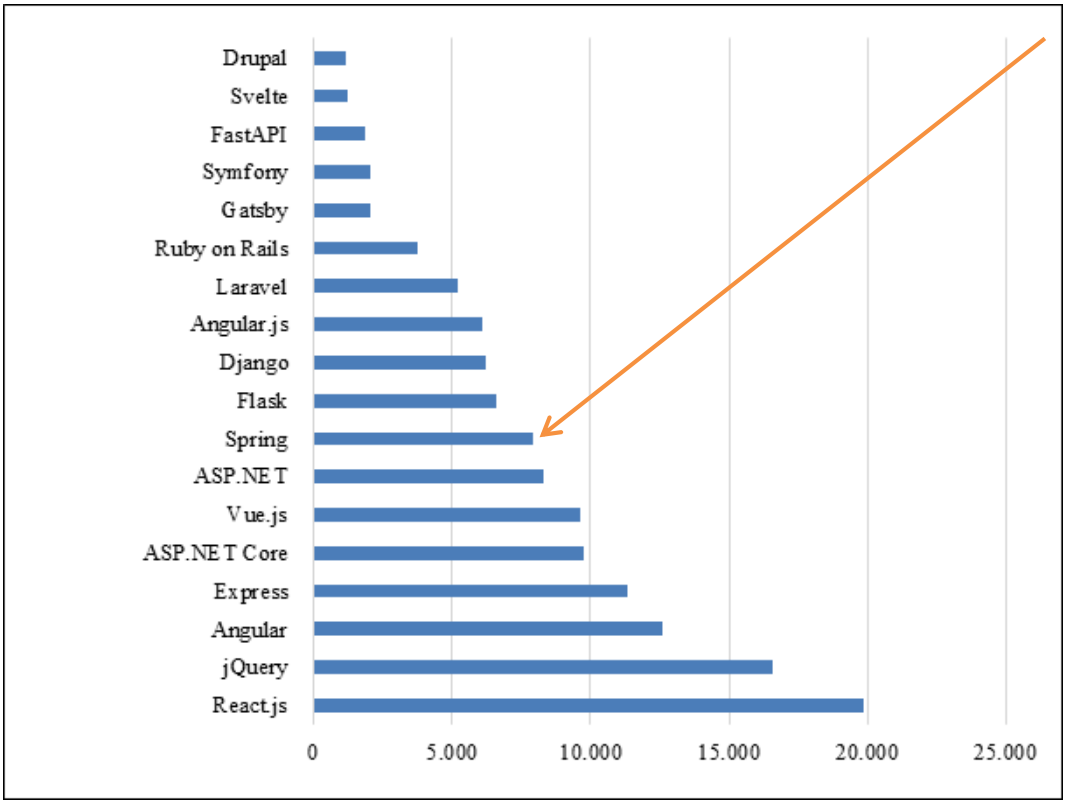


Fonte: adaptado de Stack Overflow (2021).

Considerando que uma solução de *software* deve ser entregue no prazo, ensejando simplicidade, robustez e facilidade de manutenção (JOHNSON, 2002, p. 16), Java foi escolhida como linguagem de programação juntamente com ferramentas do ecossistema Spring.

O ecossistema Spring ocupa a primeira posição no quesito popularidade tratando-se de uma ferramenta para desenvolvimento *web* em Java, segundo levantamento também realizado pelo fórum Stack Overflow (2021), tal qual demonstrado no Gráfico 2. Adicionalmente, a pesquisa constatou que o desenvolvedor Spring encontra-se contente em continuar trabalhando com a ferramenta.

Gráfico 2 - CONCENTRAÇÃO DE USUÁRIOS POR *WEB FRAMEWORK*



Fonte: adaptado de Stack Overflow (2021).

3.4. Elicitação de requisitos

Escolher a tecnologia para o desenvolvimento da solução de *software* representa uma das etapas dentro de todo o processo. Embora essa etapa tenha sua importância, compreender como entregar ao cliente aquilo que foi levantado como requisito é a chave para a criação de um produto relevante. Sendo assim, tão importante quanto saber qual tecnologia utilizar, é entender como o *software* deve se comportar.

É possível cunhar uma definição de Engenharia de *Software* a partir de Pressman, conforme Quadro 2. Para o autor, a prática de Engenharia de *Software* é alicerçada no princípio da qualidade de *software*, na relação entre a expectativa do cliente sobre o produto final, na flexibilidade e adaptabilidade para lidar com o processo de desenvolvimento.

Muito embora o autor julgue as definições propostas incipientes, é necessário notar que não há equívoco em afirmar que é preciso adotar princípios sólidos de engenharia, bem como a aplicação de uma abordagem sistemática, disciplinada e qualificável.

Quadro 2 - O QUE É ENGENHARIA DE *SOFTWARE*

Conceito de Engenharia de Software	Problematização do conceito proposto
"Engenharia de software é o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter software de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais" (BAUER & NAUR, 1969 apud. PRESSMAN, 2011, p. 39).	"Ficamos tentados a acrescentar algo a essa definição. Ela diz pouco sobre os aspectos técnicos da qualidade de software; ela não trata diretamente da necessidade de satisfação do cliente ou da entrega do produto dentro do prazo [...]" (PRESSMAN, 2011, p. 39).
"Engenharia de software: (1) Aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação da engenharia de software. (2) O estudo de abordagens como definido em (1)" (IEE, 1993 apud. PRESSMAN, 2011, p. 39).	Sobre esse conceito, Pressman pontua que uma abordagem sistemática, disciplinada e quantificável aplicada por uma equipe de desenvolvimento pode não ser adequada para outra equipe, sendo necessário atentar para a flexibilidade (PRESSMAN, 2011, p. 39). Outro ponto importante para o autor é o foco na qualidade e uma cultura de aperfeiçoamento contínuo.

Fonte: Pressman (2011, p. 39-40).

Para assegurar que a expectativa do cliente sobre o produto seja atendida, é necessário compreender o que ele deseja. Segundo Sommerville (2011, p. 59), "os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento". Conforme Pressman (2011, p. 126), "entender os requisitos de um problema está entre as tarefas mais difíceis enfrentadas por um engenheiro de software".

Sommerville (2011, p. 58) faz clara distinção entre os requisitos, separando-os em requisitos de usuário e requisitos de *software*. Segundo o autor, requisitos de usuário são "declarações em uma linguagem natural com diagramas, de quais serviços o sistema deverá fornecer a seus usuários e as restrições com as quais deve operar". Paralelamente, requisitos de sistema são "descrições mais detalhadas das funções, serviços e restrições operacionais do sistema de software".

Daltrini & Martins (1999, p. 1) propõem o uso de preceitos da Teoria da Atividade durante a elicitação de requisitos como forma de minimizar problemas, divididos em: problemas acidentais e problemas essenciais. Os autores definem os problemas acidentais durante a elicitação de requisitos como aqueles (DALTRINI & MARTINS, 1999, p. 1):

> oriundos da falta de controle sobre aquilo que precisa ser construído, dentre os quais podemos destacar: pouco esforço desprendido no levantamento de informações junto ao usuário, documentação pobre sobre os requisitos obtidos, pouca revisão dos requisitos obtidos, especificações incorretas dos requisitos e tendência em iniciar logo o processo de desenvolvimento de software.

Quanto aos problemas essenciais, intrínsecos ao próprio processo de levantamento de requisitos, estão a "dificuldade do usuário em saber efetivamente o que ele quer, dificuldade de comunicação ente usuário e desenvolvedor e a natureza mutante dos requisitos" (DALTRINI & MARTINS, 1999, p. 2). Adicionalmente, problemas na comunicação, no

entendimento, na metodologia, no processo de desenvolvimento do *software* e na organização (SERRANO et al., 2009, p. 7-15) podem surgir.

Com plena ciência de que esse é um processo delicado e crucial para o sucesso da jornada de desenvolvimento do *software*, optou-se neste trabalho por abordar o problema à luz da Teoria da Atividade: 1) foi realizada a descrição inicial do problema; 2) a partir da definição dada pelo cliente, identificaram-se procedimentos a serem realizados pelo sistema que podem ser caracterizados como atividades; 3) ocorreu a decomposição das atividades encontradas em ações e operações.

Atualmente, o cliente possui uma caderneta de papel onde anota suas vendas, a data em que vendeu o produto, por quanto vendeu e quem o comprou. Manualmente, verifica quanto lucrou na venda de um produto ao observar a diferença entre o preço de aquisição e quanto pagou para adquiri-lo do fornecedor.

A partir da declaração do problema, é possível mapeá-lo a fim de conceber o produto final. Em suma, toda atividade encontrada nesse caso particular segue um padrão: criação, consulta ou remoção. Logo, para contribuir com a brevidade dessa tarefa descritiva, é possível inferir que as atividades encontradas — venda e produto — seguem essa lógica.

Uma atividade ao passar pelo processo de decomposição revela como o *software* deve se comportar, assim como um requisito de *software*, conforme ilustrado no Quadro 3.

Quadro 3 - DECOMPOSIÇÃO DE ATIVIDADE EM AÇÕES E OPERAÇÕES

Atividade	Ação	Operações
Faturar venda	Registrar venda	- Preencher os campos da venda - Calcular o valor total da venda - Calcular o lucro proveniente da venda - Gerar uma instância da venda no banco de dados

Fonte: autoria própria (2021).

Essa abordagem, apesar de útil para compreender o que o cliente efetivamente necessita, pode ser dúbia pelo fato de reunir em um só lugar qual funcionalidade implementar, como implementá-la e seu comportamento. É de interesse do projeto, portanto, decompor esse amálgama seguindo o que sugere Sommerville, ou seja, transpondo-a na forma de requisitos de usuário e requisitos de sistema.

Sommerville (2011, p. 59) divide os requisitos de sistema em requisitos funcionais e não funcionais. Para o autor, são requisitos funcionais “as declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas e de como o sistema deve se comportar em determinadas situações”. Os requisitos funcionais propostos são apresentados no Quadro 4.

Quadro 4 - REQUISITOS FUNCIONAIS DO SISTEMA

ID	Requisito	Descrição	Prioridade
RF01	Faturar venda	O sistema deve permitir que o usuário registre uma venda	Obrigatório
RF02	Consultar venda	O sistema deve permitir que o usuário consulte e edite o registro de uma venda	Obrigatório
RF03	Remover venda	O sistema deve permitir que o usuário delete o registro de uma venda	Obrigatório
RF04	Criar a entrada de um produto	O sistema deve permitir o cadastro de produtos	Obrigatório
RF05	Consultar a entrada de um produto	O sistema deve permitir a consulta de produtos	Obrigatório
RF06	Remover a entrada de um produto	O sistema deve permitir a exclusão de produtos	Obrigatório

Fonte: autoria própria (2021).

Sommerville (2011, p. 59) destaca que são requisitos não funcionais as "restrições aos serviços ou funções oferecidas pelo sistema". Os requisitos não funcionais propostos para o sistema desenvolvido neste trabalho são apresentados no Quadro 5.

Quadro 5 - REQUISITOS NÃO FUNCIONAIS DO SISTEMA

ID	Requisito	Descrição	Prioridade
RNF01	Usabilidade	O sistema deve ser intuitivo e de fácil manuseio pela parte do cliente	Opcional
RNF02	Gerar nota fiscal	O sistema deve gerar a nota fiscal ao final da compra	Obrigatório

Fonte: autoria própria (2021).

3.5. Modelagem do sistema

A Linguagem de Modelagem Unificada (UML - *Unified Modeling Language*) é uma linguagem que visa auxiliar a descrição e o design de *software*, particularmente de *softwares* baseados no paradigma da programação orientada a objetos (FOWLER, 2003, p. 1).

O autor pontua que as linguagens de modelagem estão presentes na indústria de desenvolvimento de *software* há bastante tempo e que o principal motivo de sua existência recai sobre o fato de que as linguagens de programação não fornecem abstrações suficientes para abordar questões de *design* (FOWLER, 2003, p. 1). A acepção de design, nesse contexto, refere-se ao fato de que, segundo o autor (FOWLER, 2003, p. 1), as linguagens de programação não oferecem elementos para planejar aspectos mais conceituais e abstratos do sistema, tal qual as linguagens de modelagem como a UML oferecem.

A UML é fruto da unificação de muitas linguagens de modelagem gráfica do final da década de 80 e início da década de 90 (FOWLER, 2003, p. 2). Possivelmente por esse fato,

interpretam-na e a usam com diferentes propósitos, dentre os quais podemos listar: esboços, plantas de *software* e programação[5] (FOWLER, 2003, p. 2).

No contexto da solução de *software* desenvolvida, a modelagem é usada como esboço a fim de pensar como implementar cada funcionalidade requisitada. Aplicar a linguagem de modelagem como esboço significa que cada diagrama gerado serve como guia de uma ideia geral. É notável que, em um cenário de maior complexidade, cada diagrama precisa atingir um nível maior de fidelidade, todavia, em um cenário limitado como o apresentado neste trabalho, essa tarefa foi importante para o entendimento inicial do problema.

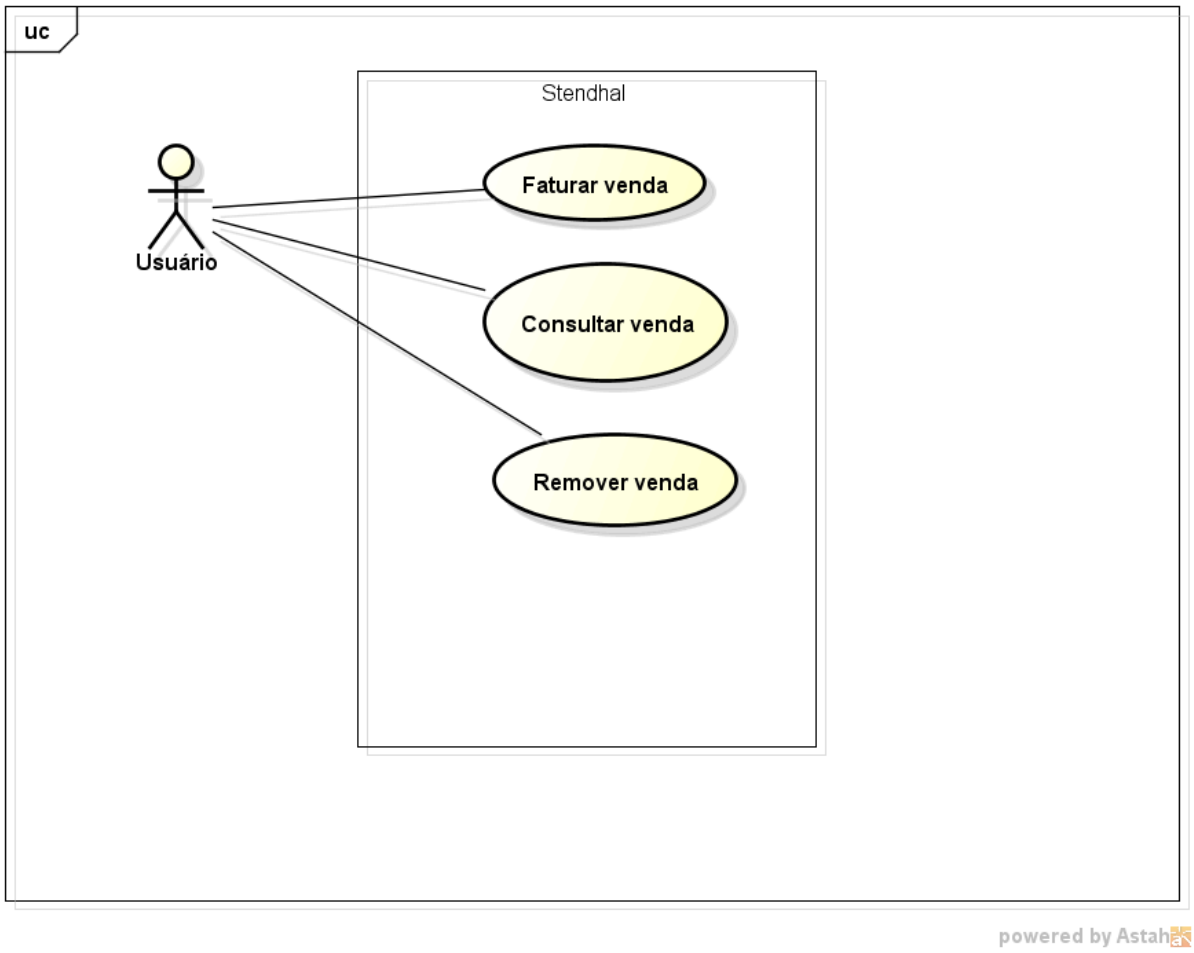
A UML oferece uma ampla gama de diagramas para representar as mais distintas etapas do desenvolvimento de *software*, como diagramas de casos de uso e diagramas de sequência.

Casos de uso são "uma técnica para capturar os requisitos funcionais de um sistema", funcionam "descrevendo as interações típicas entre os usuários do sistema e o sistema em si, proporcionando uma narrativa que demonstra como o sistema deve funcionar" (FOWLER, 2003, p. 99, tradução livre). Nesse sentido, diagramas de casos de uso nos "ajudam a determinar a funcionalidade e as características do software sob o ponto de vista do usuário" (PRESSMAN, 2011, p. 731).

Um único ator interage com o sistema e executa funcionalidades identificadas durante a etapa de elicitação de requisitos, como apresentado na Figura 1. Entrementes, a utilização de cenários, sendo um cenário uma "sequência de passos descrevendo a interação entre um usuário e um sistema", pode colaborar para o entendimento do comportamento do diagrama bem como o que ele deseja comunicar (FOWLER, 2003, p. 99, tradução livre). Logo, verifica-se mais proveitosa a aplicação de cenários na descrição de diagramas de sequência, posto que esses carregam mais informações sobre o funcionamento do sistema mediante interações do usuário.

[5] Originalmente, estes propósitos estão listados como *sketch*, *blueprint* e *programming language*.

Figura 1 - DIAGRAMA DE CASOS DE USO DO SISTEMA



Fonte: autoria própria (2021).

Dentre os diagramas disponíveis, também estão presentes os diagramas de interação, que descrevem como um grupo de objetos colabora para o comportamento e interações dentro do sistema (FOWLER, 2003, p. 53).

Diagramas de interação dividem-se em quatro tipos principais de diagramas, são eles diagramas de comunicação, diagramas de tempo, diagramas de sequência e diagramas de visão geral e integração (FOWLER, 2011, p. 131-149). Dentre os citados, são de particular interesse os diagramas de sequência, usados para “indicar as comunicações dinâmicas entre os objetos durante a execução de uma tarefa” (PRESSMAN, 2011, p. 733). Esboçar como são dispostas as funcionalidades do sistema gera reflexões sobre a forma de implementação e auxilia a identificação de possíveis pontos de melhoria e correção.

Segundo Bezerra (2015, p. 171), diagramas de interação representam as mensagens trocadas entre objetos. O autor diz que a distinção entre os diferentes tipos de diagramas está na ênfase dada às interações entre os objetos. No caso particular dos diagramas de sequência,

a ênfase está na ordem temporal na qual ocorre a troca de mensagens dentro do sistema (BEZERRA, 2015, p. 171).

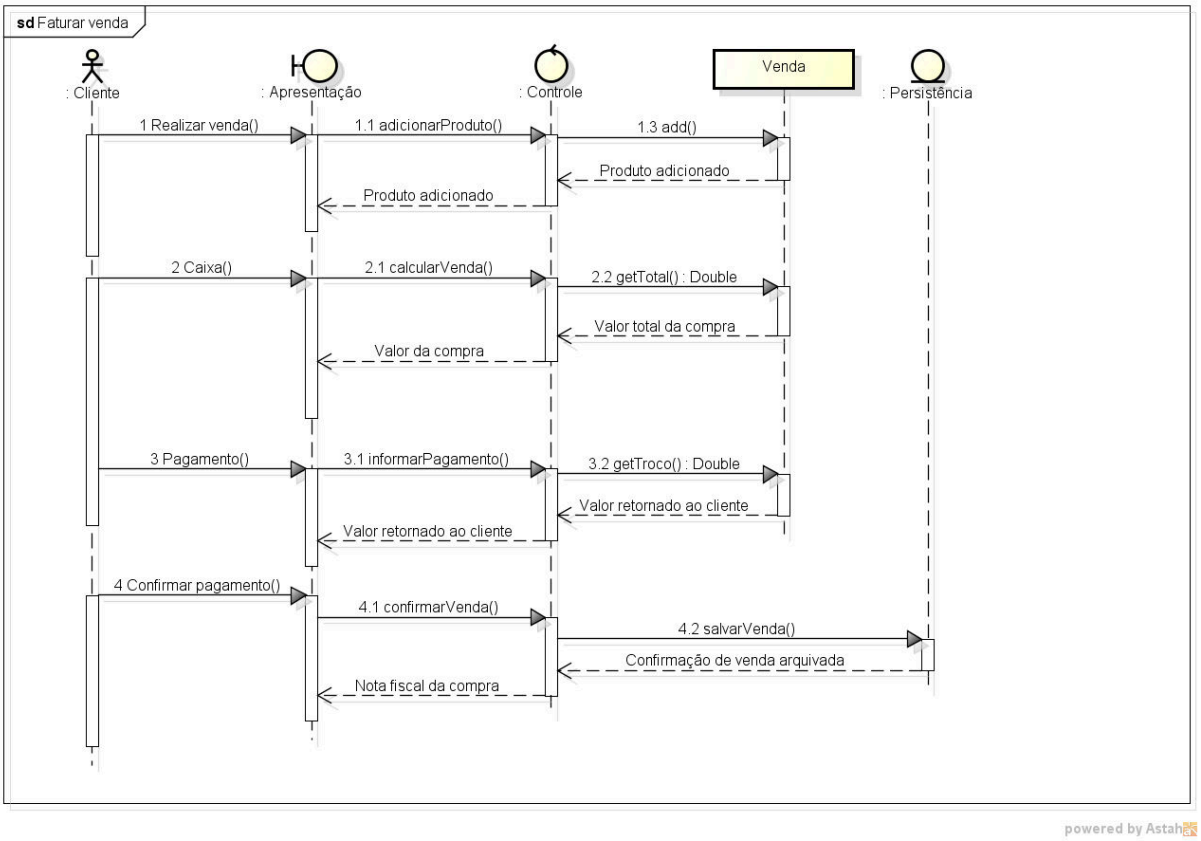
Os diagramas de sequência expostos neste trabalho procuram demonstrar, de maneira simplificada, as camadas do sistema e o conjunto de mensagens trocadas para a realização dos requisitos definidos pelo cliente. O sistema divide-se em três camadas, são elas a camada de apresentação, a camada de controle e a camada de persistência de dados[6]. A divisão em camadas, partes abstratas da aplicação, busca atribuir uma responsabilidade única a cada parcela do sistema (RUSNOK, 2020). Em nossa aplicação, essas camadas correspondem às camadas do padrão Modelo-Visão-Controle (MVC - *Model-View-Controller*).

A camada de apresentação é responsável pela comunicação e interação com o usuário. A camada de persistência é responsável pela comunicação com o banco de dados. Entre as duas, encontra-se, conceitualmente, a camada de controle. A camada de controle, quando comunica-se com a camada de apresentação, é responsável por recuperar dados e torná-los valores passíveis a serem armazenados em um banco de dados ou repassados para outras páginas do sistema; por outro lado, quando comunica-se com a camada de persistência de dados, cumpre a função de recuperar dados de um banco de dados para efetuar operações dentro do sistema. Esse processo é análogo em todos os diagramas de sequência apresentados neste trabalho.

Para criar uma venda, primeiramente o usuário deve acionar uma página da camada de apresentação que lhe possibilitará adicionar um produto a uma lista de vendas — é possível adicionar mais de uma unidade do mesmo produto em uma operação, assim como outros produtos em uma única venda. Em seguida, o usuário acionará a função de caixa, que lhe permite calcular o valor total da venda sendo realizada. Terceiro, o usuário informa o valor recebido para arcar com o preço total da venda e lhe é indicado o valor de retorno, se houver. Por último, o usuário deve confirmar a venda, que é salva em uma base de dados caso não haja falha durante a conexão com o banco de dados, encerrando o processo ao apresentar os detalhes da transação, conforme pode ser visto na Figura 2.

[6] É importante notar que a nomenclatura empregada é meramente uma forma de transmitir ao engenheiro de *software* responsável qual arquitetura será empregada no desenvolvimento do sistema. Paralelamente, estes nomes poderiam ser substituídos pelos termos em inglês *presentation*, *business*, *access*, sem prejuízo ao seu significado.

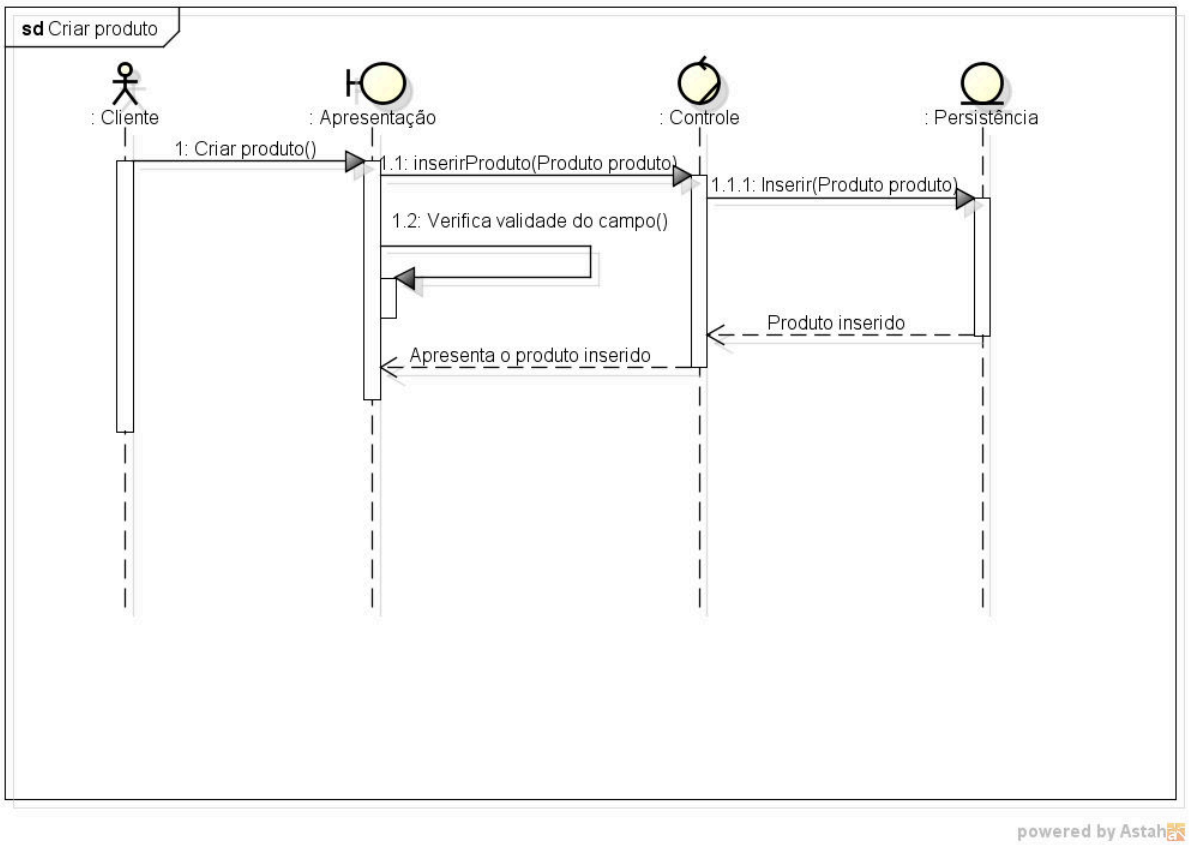
Figura 2 - DIAGRAMA DE SEQUÊNCIA: CRIAÇÃO DE UMA VENDA



Fonte: autoria própria (2021).

Para criar um produto, o cliente acessa uma página da camada de apresentação contendo um formulário, tal como ocorre em um processo manual de catalogação. Durante o preenchimento do formulário, verifica-se a integridade dos dados inseridos de modo a garantir a consistência da base de dados. Caso os dados inseridos não apresentem nenhuma irregularidade, são estruturados e encaminhados à camada de persistência de dados. Considerando que não haja uma falha de comunicação com o banco de dados, o produto é adicionado ao catálogo e persiste na base de dados, conforme apresentado na Figura 3.

Figura 3 - DIAGRAMA DE SEQUÊNCIA: CRIAÇÃO DE UM PRODUTO



Fonte: autoria própria (2021).

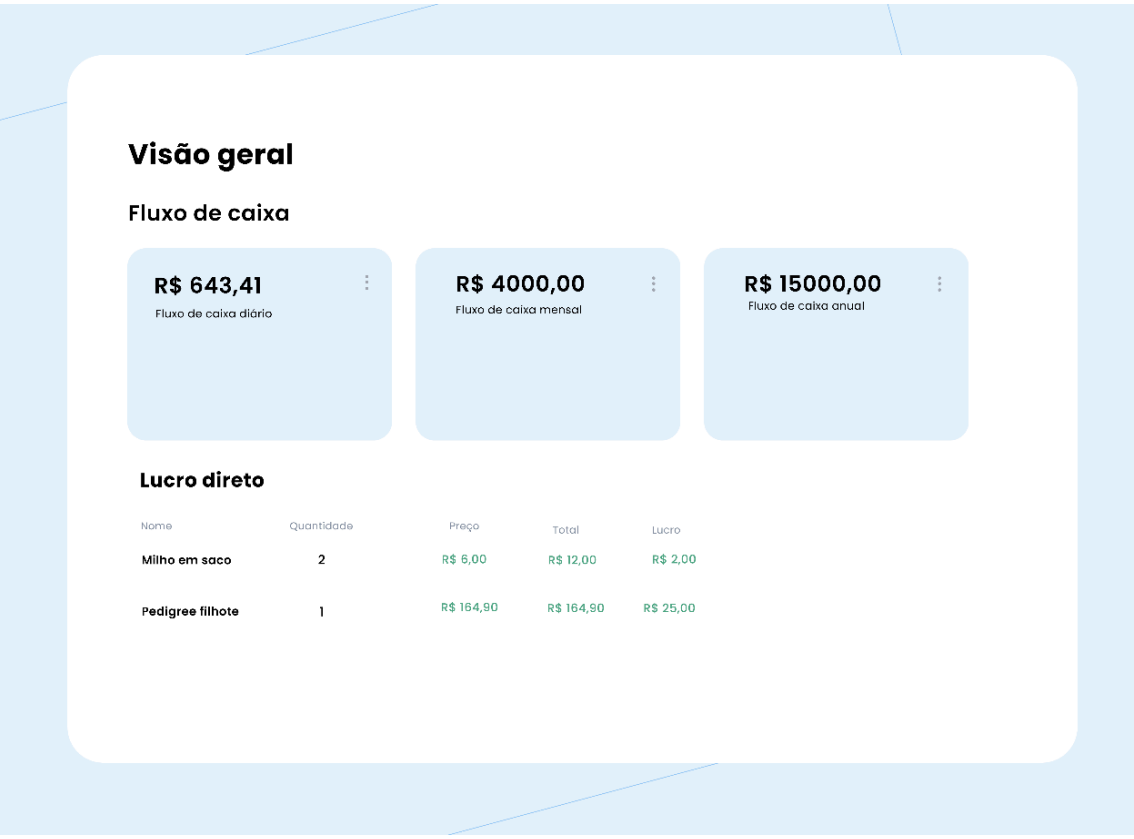
3.6. Desenvolvimento e validação do protótipo de interface

A interface desenvolvida foi adaptada de um modelo preexistente encontrado em um banco de interfaces - um local que agrega criações de terceiros, sem necessariamente possuir relação direta com seu criador. O banco de interfaces oferece a seus usuários uma coleção de modelos, além de ferramentas adicionais para auxiliar o desenvolvimento de interfaces próprias.

O banco de interfaces não hospeda nenhum modelo em seu domínio. Cumprindo com o papel de agregador, ele direciona seus usuários aos criadores do produto original, muitas vezes apontando para plataformas externas, como a Dribbble.

A página do autor não situa o visitante sobre a licença e permissões de uso da interface. Todavia, o agregador nos informa de sua gratuidade, fato depreensível pelo acesso irrestrito ao arquivo editável da interface fornecido por seu proprietário. O arquivo obtido foi modificado utilizando o editor gráfico Figma, tal como disposto na Figura 4.

Figura 4 - PROTÓTIPO DE INTERFACE APRESENTADO AO CLIENTE



Fonte: adaptado da interface desenvolvida por Niqox (2021)[7].

O modelo desenvolvido e disponibilizado tratava-se originalmente de uma interface para exibição de uma carteira para acompanhar o saldo e a valorização de criptomoedas. Tal modelo também contava com uma barra para pesquisa e um menu de usuário. Naturalmente, o presente trabalho não possui semelhante finalidade, de modo que não há nada que justifique manter a interface tal qual aquela que foi tomada como modelo. Para esse trabalho, foram preservadas tipografia, paleta de cores e disposição dos quadros que indicavam os valores correspondentes ao fluxo de caixa da microempresa.

3.7. Desenvolvimento do protótipo da aplicação

Conforme Neto (2016, p. 24), o ciclo de vida de um *software* segue um padrão linear bem definido. Para o autor, esse ciclo compreende cinco etapas distintas, descritas adiante: a elicitação, o planejamento do projeto, a implementação do produto, os testes do sistema e a manutenção.

Destarte, destacamos as etapas do ciclo de vida de um *software* tal qual definidas pelo autor: a elicitação de requisitos, um momento dedicado ao entendimento das necessidades do cliente em relação ao produto desenvolvido (NETO, 2016, p. 23); o planejamento do projeto,

[7] A referência original encontra-se no seguinte endereço eletrônico: <https://dribbble.com/shots/15270128-Crypto-Wallet-UI-design>. Acesso em: 29 de jul. de 2021.

que preocupa-se em como atender as necessidades do cliente[8] (NETO, 2016, p. 23); a implementação do produto, onde "o projeto é transformado em linguagem de programação para que, de fato, o produto passe a ser executável" (NETO, 2016, p. 24); os testes do sistema, ou seja, "a execução do programa em busca de defeitos" (NETO, 2016, p. 25); pelo entendimento de que o *software* pode se encontrar em permanente evolução, a manutenção caracteriza a etapa final do ciclo de vida (NETO, 2016, p. 25).

Durante a etapa de implementação, a criação de um protótipo — uma "versão com funcionalidades apenas de tela para proporcionar entendimento das entradas e saídas do software" (NETO, 2016, p. 24) — pode ser útil para a validação de requisitos (NETO, 2016, p. 24; SOMMERVILLE, 2011, p. 35). Adicionalmente, um protótipo também pode servir para que o usuário compreenda melhor a interface (SOMMERVILLE, 2011, p. 35).

Lançando mão dos requisitos levantados junto ao cliente, foi construído um protótipo navegável de alta fidelidade. Para tanto, utilizou-se a ferramenta de edição gráfica Figma, que habilita a prototipação ao atribuir ações aos componentes que fazem parte da tela. Há de se notar, porém, que a prototipação possibilitada pela ferramenta pode se tornar um trabalho demorado e complexo, pois para cada interação do usuário com os componentes deve ser feita uma nova tela que represente o resultado de sua ação.

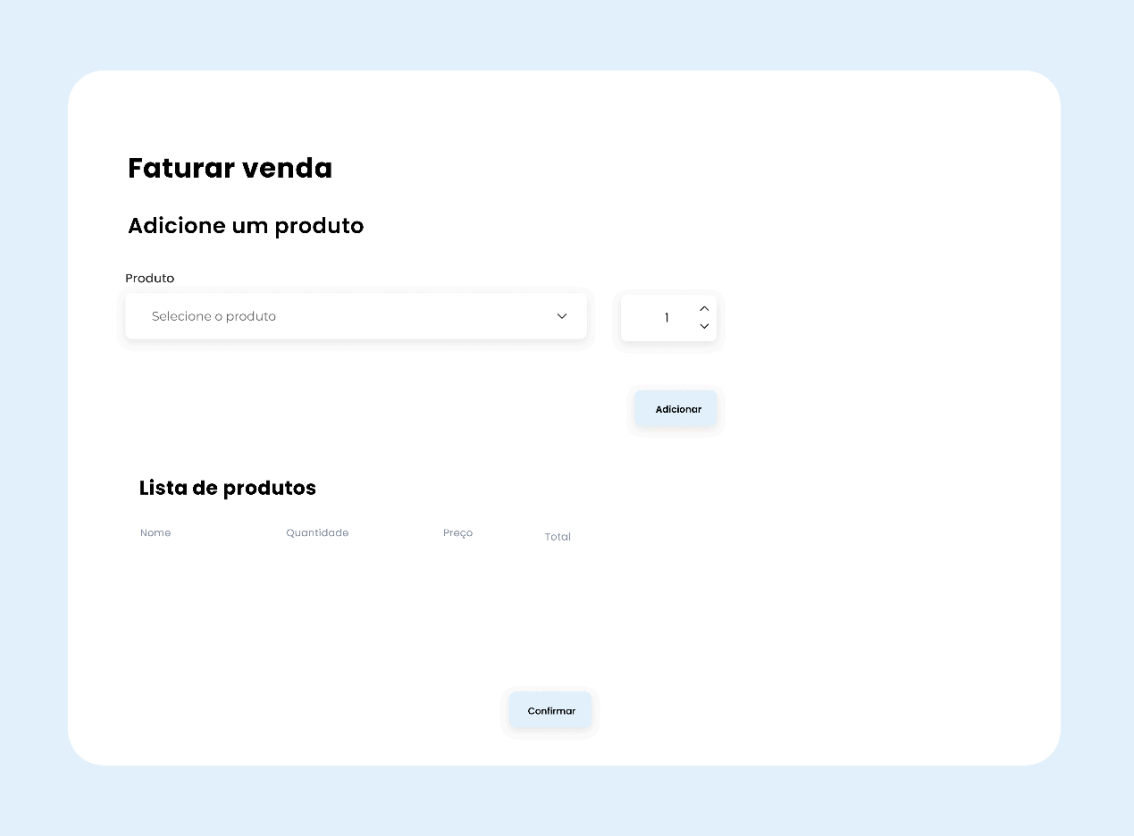
O protótipo navegável de alta fidelidade, embora não represente a versão final do sistema com todas as suas funcionalidades, apresenta ao cliente a materialização de seus pedidos em relação ao produto, coletados na forma de requisitos. Dessa forma, de modo a garantir que os requisitos foram corretamente compreendidos, um protótipo foi submetido à aprovação do cliente, o qual implementou a funcionalidade correspondente à criação de uma venda.

A tela inicial do protótipo representa a página que possibilita a seleção de um produto referente à venda em curso, como nota-se na Figura 5. A partir dessa tela o usuário pode selecionar um produto do menu e sua quantidade. Ao clicar no botão "Adicionar", logo abaixo do componente que indica a quantidade do produto, o usuário adiciona o referido produto a uma lista, posteriormente utilizada para cadastrar a venda em curso[9].

[8] Neto (2016, p. 24) nos diz que se os requisitos nos mostram "o que" o sistema deverá fazer, o projeto deve refletir "como" ele o fará.

[9] Vale ressaltar que esse processo não é otimizado nem tampouco desejável em um cenário com alto fluxo de caixa, porém, dada a natureza deste trabalho que é automatizar uma funcionalidade da administração financeira de uma microempresa por meio de uma solução de *software*, não houve a preocupação, a princípio, em fornecer uma solução célere para essa tarefa.

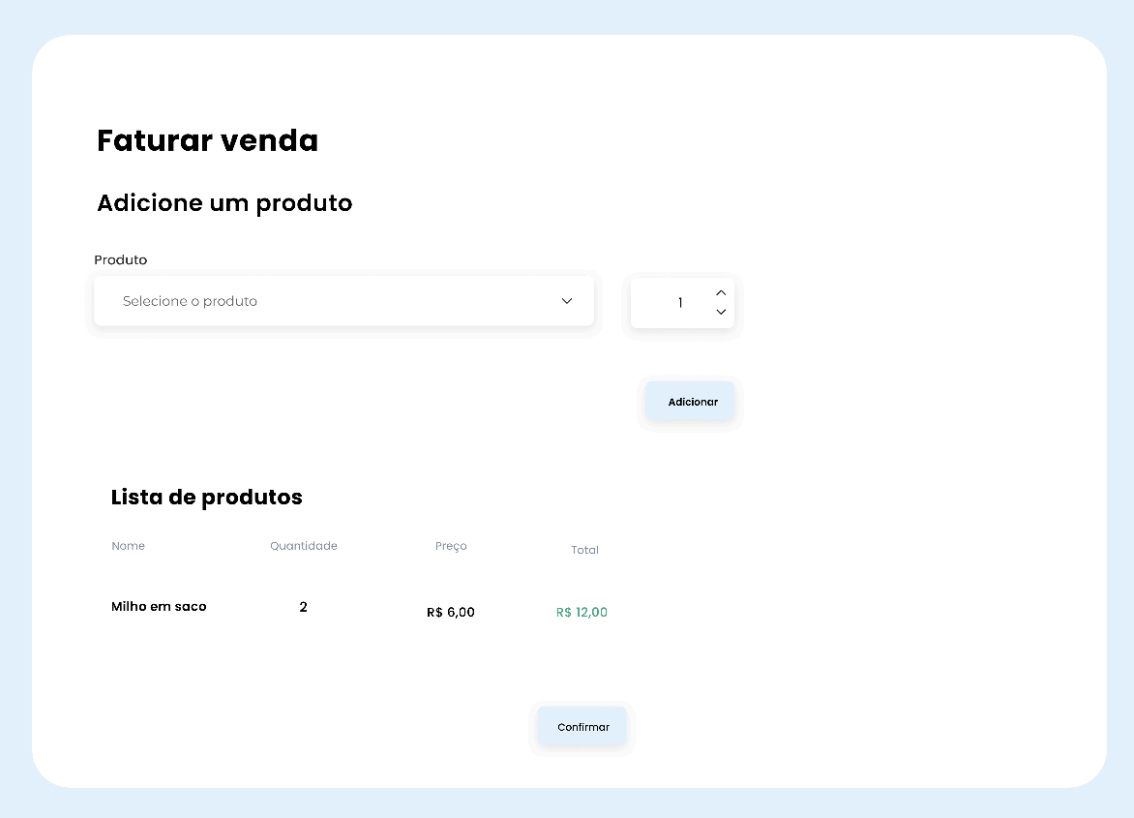
Figura 5 - TELA INICIAL DO PROTÓTIPO



Fonte: autoria própria (2021).

O produto adicionado é exibido na parte inferior da tela, contendo nome, quantidade, preço e valor total, como nota-se através da Figura 6. Quando o usuário certifica-se que adicionou todos os produtos referentes à venda em decurso, pode prosseguir para a tela de cadastro de vendas ao clicar no botão “Confirmar”. É importante notar que o botão “Confirmar” faz-se presente na tela anterior, conquanto não lhe foi atribuído nenhum comportamento pelo entendimento de que uma transação desse tipo não pode acontecer sem um produto a ser vendido.

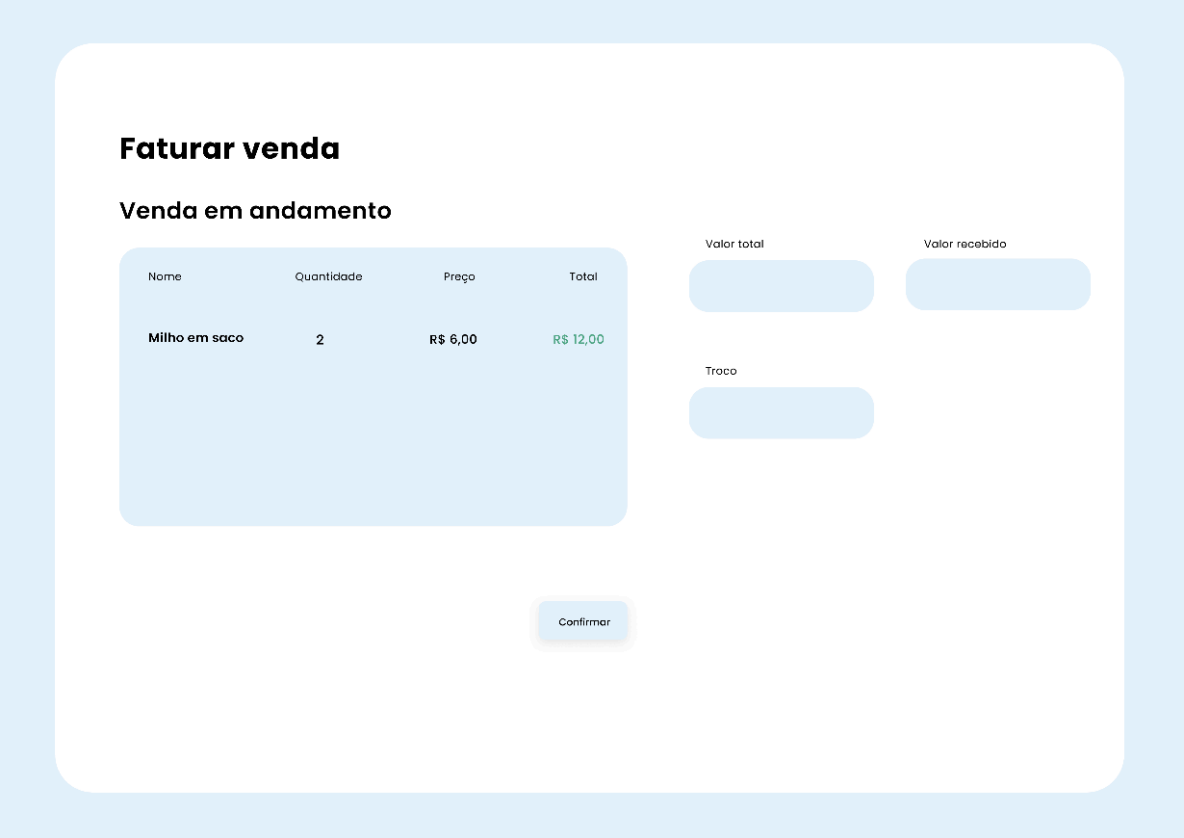
Figura 6 - INSERÇÃO DE UM PRODUTO NA LISTA



Fonte: autoria própria (2021).

Caso haja ao menos um produto presente na lista de produtos, ao clicar no botão “Confirmar” o usuário é encaminhado para a tela de confirmação de venda, como apresenta a Figura 7. A tela contém uma lista de produtos, adicionados pelo usuário em operações anteriores, e campos correspondentes à soma total do valor dos produtos listados, um indicativo de quanto foi recebido para quitar a despesa e a diferença entre essa importância e o valor total dos produtos. Para finalizar essa operação e faturar a venda, o usuário pode clicar no botão “confirmar”, presente abaixo da lista de produtos.

Figura 7 - FATURAR VENDA



Fonte: autoria própria (2021).

O final do procedimento apresenta a página de resumo da venda. O resumo contém um número de identificação único para cada venda realizada, a data em que sucedeu e o preço total, como evidencia a Figura 8. Adicionalmente, apresenta ao usuário uma lista com todos os produtos vendidos durante essa operação.

Figura 8 - RESUMO DA VENDA

Fonte: autoria própria (2021).

3.8. Desenvolvimento das funcionalidades

A solução de *software* integra a interface do usuário diretamente com o código da aplicação em uma arquitetura monolítica. A arquitetura resultante, com todo o *software* concentrado em um único lugar, pode gerar dificuldade de manutenção futura.

A ferramenta de design e prototipação foi essencial para a criação de uma interface limpa e agradável. A partir dessa ferramenta, foi possível obter arquivos CSS correspondentes aos elementos que compõem as páginas do protótipo de interface apresentado ao cliente. Todavia, a estrutura das folhas de estilo obtidas era de difícil compreensão, além de que, feitas sob medida para um tamanho específico de tela, não carregavam medidas que garantissem a portabilidade pretendida aos mais diversos dispositivos.

Para a interface do usuário, HTML e CSS dão forma à apresentação do sistema. Percebeu-se que aplicar puramente CSS para gerenciar a parte visual do documento é uma abordagem pouco produtiva, embora garanta ao programador liberdade e domínio sobre o comportamento visual da página.

Uma possível alternativa para esse entrave e uma abordagem consideravelmente superior encontra-se no reuso de componentes prontos e altamente customizáveis. Assumindo que desenvolver uma interface gráfica genérica a partir de componentes reutilizáveis requer

uma quantidade significantemente menor de esforço, ao mesmo tempo que produz resultados satisfatórios, é seguro inferir que há um ganho significativo na produtividade.

Diante do exposto, é preferível aderir a uma ferramenta naturalmente responsiva, com grande variedade de componentes reutilizáveis e compatibilidade entre múltiplos navegadores. Nesse sentido, é justificada a escolha de uma ferramenta como Bootstrap que, além de gratuita e de código aberto, supre todas as necessidades mencionadas.

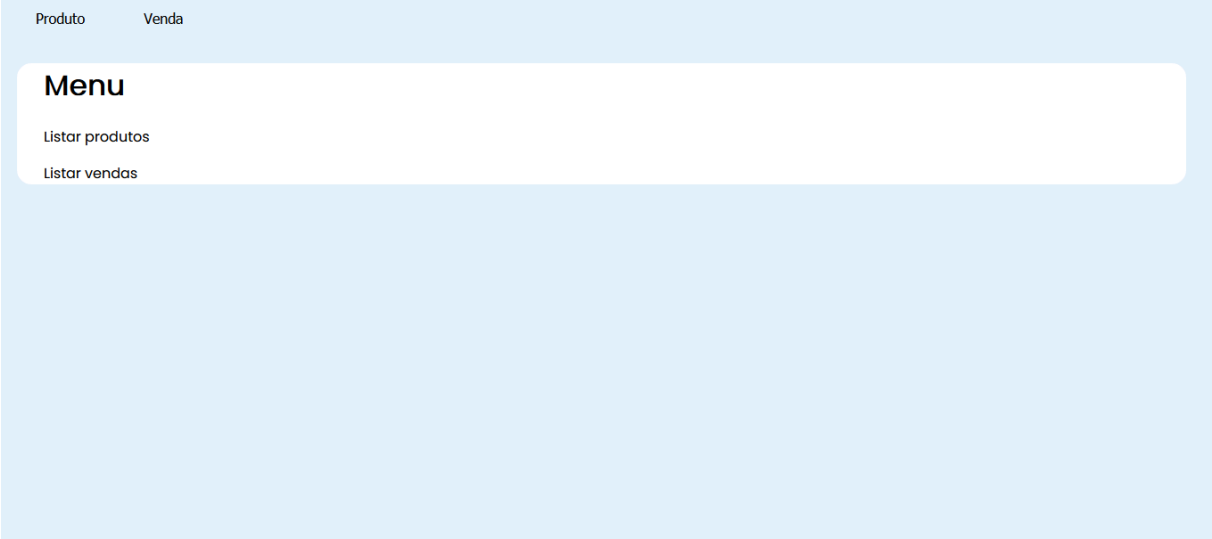
Sabe-se que a funcionalidade desenvolvida para o acompanhamento dos lucros sobre a venda de produtos é tão somente um conjunto de operações comuns a todos os sistemas computacionais: criar, ler, atualizar e deletar dados. Nesse sentido, elementos do ecossistema Spring auxiliam o programador em tarefas corriqueiras provendo interfaces com a assinatura de métodos utilitários, os quais implementam consultas básicas, mas necessárias, a qualquer sistema que comunique-se com um banco de dados.

Para além do acesso a dados, outros elementos do ecossistema Spring facilitam a execução do projeto localmente, o tráfego entre páginas e requisições. Em contrapartida, justamente por tratar muitos aspectos da programação internamente, pode causar certo receio quanto ao número de anotações, configurações e dependências necessárias para a criação de um projeto Spring.

A aplicação foi implantada com sucesso por meio do serviço oferecido gratuitamente pelo Heroku. Dessa condição advém a necessidade de aderir à política de uso da plataforma em relação à gratuidade, em outras palavras, cada aplicativo recebe uma quantidade de horas gratuitas (PUNDSACK, 2013), consumidas mediante tráfego externo; caso o aplicativo não seja requisitado em um período de trinta (30) minutos, ficará inativo (HEROKU, 2020).

A página inicial da aplicação funciona como ponto de chegada (*entry-point*) para o usuário, conforme indica a Figura 9. A partir da tela inicial, o usuário pode navegar para a seção Produto ou Venda, também acessível por meio do menu.

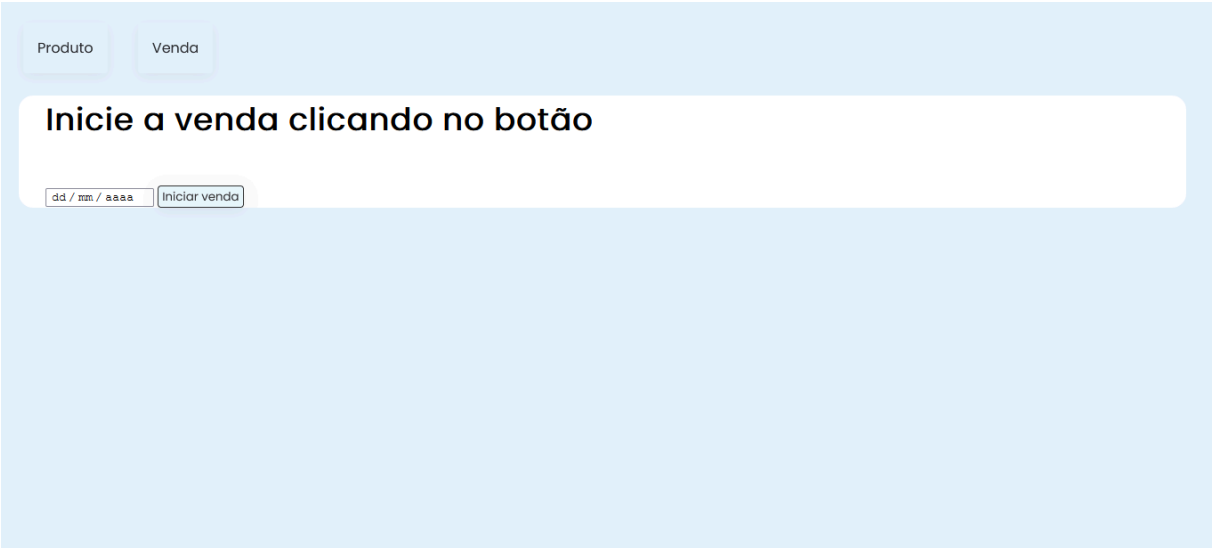
Figura 9 - PÁGINA INICIAL DA APLICAÇÃO



Fonte: autoria própria (2021).

A página responsável por faturar a venda contém um campo para que o usuário possa indicar a data em que fora realizada, como denota a Figura 10. Ao clicar em "Iniciar venda" com uma data válida selecionada, o cliente indica que deseja armazenar uma venda referente a aquele momento — é então criada uma instância referente a essa venda na base de dados.

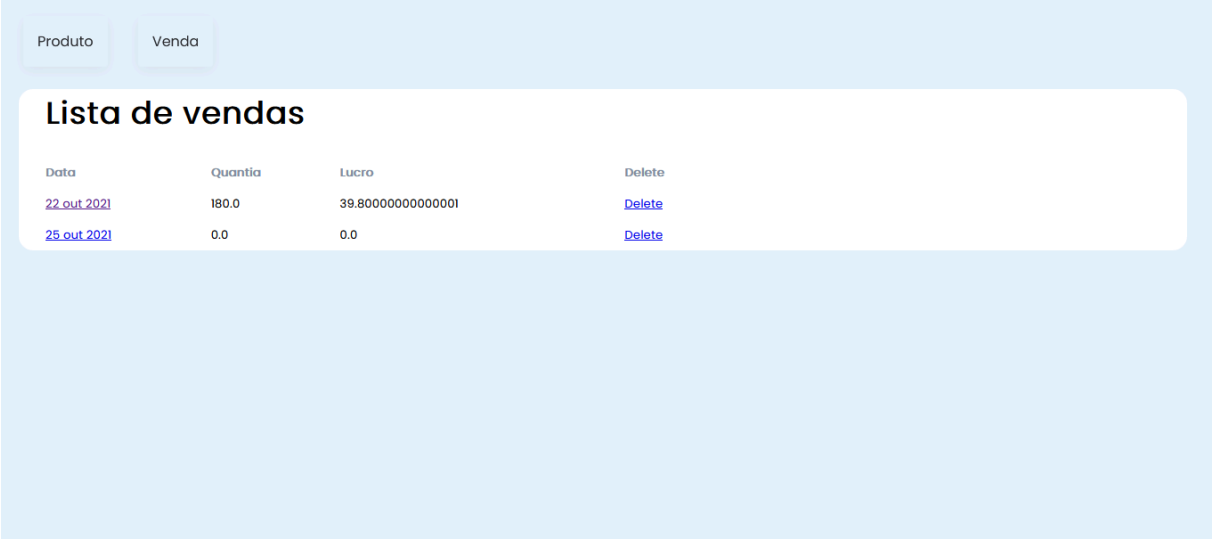
Figura 10 - FATURAR VENDA



Fonte: autoria própria (2021).

O usuário pode visualizar todas as vendas realizadas e devidamente registradas, como ilustra a Figura 11. Por padrão, a venda é instanciada com a data indicada, a quantia apurada e o lucro proveniente, ambos inicializados com valor zero, tendo em vista que ainda não foram relacionados produtos à venda.

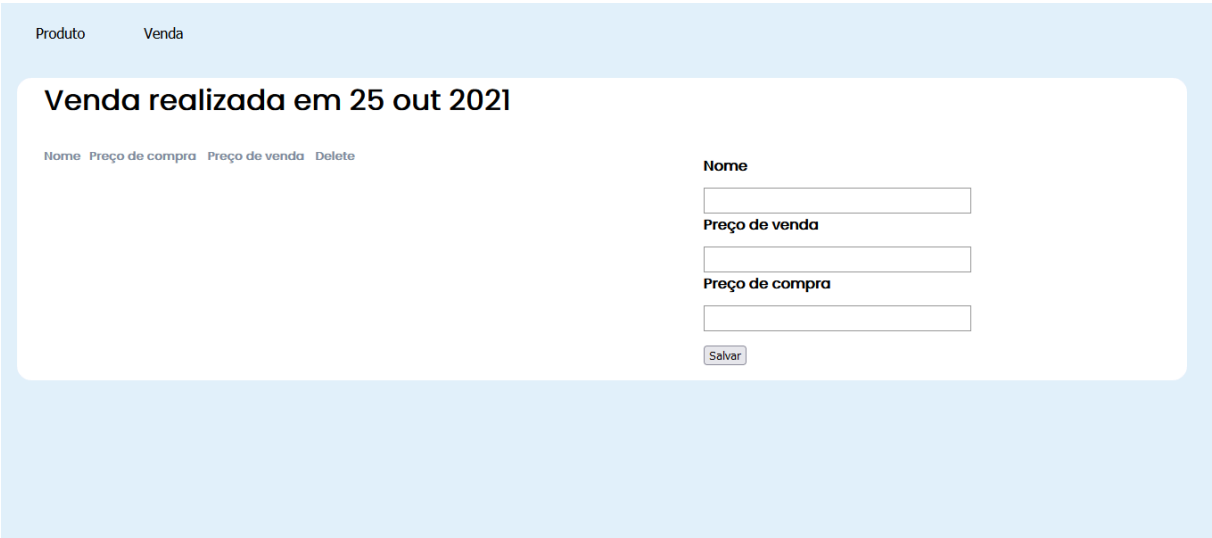
Figura 11 - LISTA DE VENDAS



Fonte: autoria própria (2021).

O usuário deve clicar no campo que indica a data em que fora realizada a venda para que possa relacionar um produto a uma venda. A partir desse campo, o usuário tem acesso a um formulário para que possa adicionar o produto vendido na ocasião, conforme nota-se pela Figura 12, fá-lo ao clicar no botão “Salvar”. A operação resulta na atualização da quantia apurada com a venda, do lucro que a sucede e da tabela onde são dispostas informações como nome, preço de compra e venda referentes ao produto.

Figura 12 - RELACIONANDO PRODUTOS À VENDA



Fonte: autoria própria (2021).

Por meio do menu, o usuário pode solicitar a visualização de todos os produtos registrados, onde cada qual representa uma unidade composta por nome, preço de compra e preço de venda, conforme demonstra a Figura 13.

Figura 13 - LISTA DE PRODUTOS VENDIDOS

Fonte: autoria própria (2021).

Para facilitar o manuseio, o gerenciamento e a interação do programador com a base de dados do sistema já em fase de produção, foi utilizada uma instância local do sistema de gerenciamento de banco de dados PostgreSQL. A partir da variável de ambiente criada no momento de inicialização do projeto na plataforma Heroku, foi possível extrair uma cadeia de caracteres com informações pertinentes para a conexão com o banco de dados em produção (a *String* de conexão completa), feita localmente via pgAdmin, a interface gráfica fornecida pelo PostgreSQL.

3.9. Débito técnico

Cunningham (2011) cunhou a metáfora do débito técnico para traduzir um problema enfrentado no desenvolvimento de *software*. A metáfora faz referência ao momento em que é necessária a refatoração ou evolução do *software*, mas a solução não foi implementada de maneira ideal — quer seja por falta de visão à época ou ausência de conhecimentos técnicos suficientes — e representa empecilhos que dificultam a tarefa.

Segundo Fowler, todo *software* está sujeito a contrair um débito técnico. O autor esclarece que o débito técnico é formado por "deficiências na qualidade interna (do *software*) que tornam as modificações e expansões do sistema mais difíceis que o comum" (FOWLER, 2019, grifo nosso, tradução livre). Adicionalmente, Kerievsky (2004, p. 59) sugere que o

débito técnico[10] é consequência da inconsistência na remoção de códigos duplicados, da escrita de códigos desnecessariamente complexos ou inteligíveis.

Martin propõe que o débito técnico consiste nas "decisões que as partes interessadas e os desenvolvedores de *software* precisam tomar a fim de cumprir os prazos e atender as expectativas do cliente", onde muitas vezes opta-se por soluções menos elegantes em vista dos prazos estipulados (MARTIN, 2009, tradução livre). Vale destacar que, para o autor, essas decisões devem ser tomadas racionalmente, de modo que excluem-se os casos onde não há ciência de como aplicar boas práticas (MARTIN, 2009).

Para Fowler (2009), questionar se uma falha configura ou não um débito técnico a partir de sua natureza é um equívoco - se havia ciência ou não de como aplicar boas práticas -, tão logo o propósito da metáfora é "ser útil para pensar em como lidar com problemas estruturais e como transmitir esses pensamentos", de modo que as decisões tomadas de forma impensada são apenas débitos imprudentes (FOWLER, 2009, tradução livre). Dessarte, o autor classifica o débito técnico em quatro (4) categorias, são elas: deliberado e imprudente, deliberado e prudente, inadvertido e imprudente e, finalmente, inadvertido e prudente[11] (FOWLER, 2009).

Adquirir um débito técnico, seja ele deliberado ou não, não é por si um sinal de ausência de zelo, apesar do que pode implicar a nomenclatura. Para além de como o débito foi contraído, é imperativo identificá-lo para futura correção — aqui encontra-se a aplicação real da metáfora, ou seja, uma forma de refletir sobre como um dado programa foi implementado e como torná-lo melhor. Adiante elenca-se o débito técnico identificado ao final da etapa de desenvolvimento da aplicação.

Primeiro, a dissonância ou descompasso entre a documentação e a efetiva implementação do projeto tornou nítido o débito técnico em relação ao faturamento da venda. Uma visita ao diagrama de sequência que descreve o fluxo da operação indica que, em comparação ao produto final, não há emissão de nota fiscal. Além disso, a visualização do lucro proveniente da venda, embora de fácil depreensão por meio da interface, não pode ser filtrada pelo usuário de acordo com sua preferência temporal.

Segundo, o desalinho entre o protótipo de alta fidelidade e a implementação final do produto. Nota-se que o uso de um protótipo de alta fidelidade pode transmitir a ideia de um

[10] O autor escolhe outra nomenclatura para o termo, adotando "*design debt*" em detrimento de "*technical debt*", termo adotado por Fowler (2019), Cunningham (2011) e Martin (2009). Para fins de consistência deste trabalho, seguimos utilizando a tradução livre para o termo "*technical debt*", débito técnico.
[11] Originalmente listados como "*deliberate reckless*", "*deliberate prudent*", "*inadvertent reckless*", "*inadvertent prudent*", respectivamente.

sistema finalizado, posto que demonstra o comportamento pretendido das funcionalidades requisitadas em um ambiente próximo ao real. No entanto, faz-se saber que essa abordagem pode gerar descontentamento quando a implementação final do sistema difere daquela indicada pelo protótipo, tendo em vista que tanto sua aparência quanto seus comportamentos foram previamente aprovados.

Por último, o fluxo da aplicação oferece pouca praticidade ao usuário, conquanto não há problema em relação à funcionalidade implementada. Pouco otimizado, faz com que seja necessário dirigir-se a uma venda específica, em outra página, para prosseguir com a operação. Quase simetricamente, faz com que seja preciso retornar à página de listagem de vendas para consultar o valor total da venda e sua margem de lucro.

O débito que se enseja é, portanto, de revisitar e melhorar a interface com o usuário, procurando reunir duas ou três telas em apenas uma, mais robusta e com mais informações para o cliente.

4. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Nesta seção são apresentados os resultados obtidos ao final do desenvolvimento da aplicação e elencados trabalhos futuros que almejam a evolução do *software*.

4.1. Resultados obtidos

Por fim, ante tudo o que foi apresentado, é possível afirmar que o presente trabalho foi bem sucedido em sua empreitada. A documentação provê uma visão detalhada do comportamento da aplicação, além disso, o produto final foi desenvolvido em harmonia com a documentação, salvo leve débito técnico descrito na seção 3.9.

A aplicação foi implantada com êxito, tanto localmente quanto em produção, como visto nas seções 3.7 e 3.8. Ainda, a instância local do sistema de gerenciamento de banco de dados conectada à base de dados do ambiente em produção confere celeridade aos processos administrativos, posto que remove a necessidade de autenticação prévia com a plataforma Heroku para utilização do serviço.

A validação da aplicação foi feita junto ao cliente durante o decorrer da implementação, prontamente documentada na seção 3. Entretanto, é impraticável falar em validação de *software* submetendo-o ao crivo de uma única pessoa. Alerto o leitor para o fato de que, no presente trabalho, a validação assume uma acepção simplificada do termo, onde diz-se validado o *software* aprovado pelo cliente.

Ao final de cada etapa do desenvolvimento, conforme consta no Quadro 1, do protótipo da interface à aplicação em ambiente de produção, o produto final era submetido à apreciação do cliente. Essa abordagem, no entanto, não logrou o sucesso que se esperava em razão da brevidade com que o cliente exprimia-se, embora suficiente para apontar o problema de fluxo da informação dentro do projeto. Apesar desse entrave, foi constatado que a aplicação supre a necessidade do cliente em relação ao registro de vendas e acompanhamento de lucros.

4.2. Trabalhos futuros

Trata-se de um trabalho iterativo e incremental, de sorte que sua natureza é evoluir continuamente e tornar-se melhor. Com isso em mente, adiante são listados pontos para melhorias da aplicação na forma de trabalhos futuros.

Primeiro, a emissão de nota fiscal eletrônica ao final da venda. Essa funcionalidade é significativamente complexa uma vez que perpassa o ramo legal, aqui se deve ponderar sobre o momento correto em implementar tal funcionalidade ou recorrer ao uso de *software* externo.

Segundo, a seleção automática do produto durante a venda. Essa funcionalidade busca acelerar a venda ao remover o tempo de digitação necessário para informar qual produto faz parte da negociação. Para que não se repita o que se sucedeu com o registro manual em papel, a celeridade desse processo é imprescindível, de modo que o cliente não veja a solução proposta como uma mera transposição de suas atividades para o meio digital e gradualmente abandone seu uso.

Terceiro, a possibilidade de aplicar um determinado filtro sobre a consulta de uma venda, podendo agrupá-la por data de realização, valor total obtido e lucratividade. Essa funcionalidade anseia proporcionar ao cliente buscas mais significativas e alinhadas aos seus interesses.

Finalmente, busca-se a correção do fluxo da informação apresentado na seção 3.9 e maior alinhamento entre o protótipo de alta fidelidade e a versão final do produto.

REFERÊNCIAS

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. 2ª edição. Rio de Janeiro: Campus-Elsevier, 2006.

BRIGHAM, E. F.; EHRHARDT, M. C. **Financial management:** Theory & Practice. 16ª edição. Boston: Cengage Learning, 2019.

BROOKS, R. **Financial Management:** Core Concepts. Londres: Pearson Education, 2015.

CLASON, G. S. **The Richest Man in Babylon**. Nova Iorque: New American Library, 1926.

CUNNINGHAM, H. **Ward Explains Debt Metaphor**. Wiki, 2011. Disponível em: <http://wiki.c2.com/?WardExplainsDebtMetaphor>. Acesso em: 21 de out. de 2021.

DALTRINI, B. M.; MARTINS, L. E. G. **Utilização dos Preceitos da Teoria da Atividade na Elicitação dos Requisitos do Software**. SBES'99 (1999).

DIAMOND, J.; KHEMANI, P. **Introducing financial management information systems in developing countries**. 2005.

DRUCKER, P. F. **Post-Capitalist Society**. 3ª edição. Nova Iorque: Routledge, 2011.

_____. **Technology, Management, and Society**. Nova Iorque: Routledge, 2004.

FOWLER, M. **Technical Debt Quadrant**. Martin Fowler, 2009. Disponível em: <https://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html>. Acesso em: 23 de out. de 2021.

_____. **Technical Debt**. Martin Fowler, 2019. Disponível em: <https://www.martinfowler.com/bliki/TechnicalDebt.html>. Acesso em: 19 de out. de 2021.

_____. **UML Distilled:** A Brief Guide to the Standard Object Modeling Language. 3ª edição. Massachusetts: Addison-Wesley Professional, 2003.

HEROKU. **Free Dyno Hours**. Heroku, 2020. Disponível em: <https://devcenter.heroku.com/articles/free-dyno-hours>. Acesso em: 20 de out. de 2021.

JOHNSON, R. **Expert One-On-One:** J2EE Design and Development. 1ª edição. Birmingham: Wrox Press, 2002.

KERIEVSKY, J. **Refactoring to Patterns**. Massachusetts: Addison-Wesley Professional, 2004.

KIRMANI, M. M.; WANI, F. A.; SAIF, S. M. **Impact of ICT on effective financial management**. International Journal of Information Science and System, v. 4, n. 1, p. 1-14, 2015.

KNUDSEN, J.; NIEMEYER, P. **Learning Java**. 1ª edição. Massachusetts: O'Reilly Media, 2000.

LAKATOS, E. M.; MARCONI, M. A. **Fundamentos da Metodologia Científica. 5ª edição. São Paulo: Atlas, 2003.**

MARSHEV, V. I. **History of Management Thought:** Genesis and Development from Ancient Origin to the Present Day. Nova Iorque: Springer International Publishing, 2021.

MARTIN, R. C. **A Mess Is Not A Technical Debt**. Uncle Bob Consulting LLC, 2009. Disponível em: <https://sites.google.com/site/unclebobconsultingllc/a-mess-is-not-a-technical-debt>. Acesso em: 23 de out. de 2021.

MIDDLETON, N.; SCHNEEMAN, R. **Heroku Up and Running:** Effortless Application Deployment and Scaling. Massachusetts: O'Reilly Media, 2013.

NETO, M. R. **Engenharia de Software**. Londrina: Editora e Distribuidora Educacional S. A., 2016.

PFAFFENBERGER, B.; SCHAFER, S. M.; WHITE, C.; KAROW, B. **HTML, XHTML, and CSS Bible**. 3ª edição. Nova Jérsei: Wiley Publishing, 2004.

PRESSMAN, R. S. **Engenharia de Software:** uma abordagem profissional. 7ª edição. Porto Alegre: AMGH, 2011.

PUNDSACK, M. **App Sleeping On Heroku**. Heroku, 2013. Disponível em: <https://blog.heroku.com/app_sleeping_on_heroku>. Acesso em: 20 de out. de 2021.

RUSNOK, D. **Layers in Software Architecture That Every Software Architect Should Know**. Level Up, 2020. Disponível em: <https://levelup.gitconnected.com/layers-in-software-architecture-that-every-sofware-architect-should-know-76b2452b9d9a>. Acesso em: 30 de jul. de 2021.

SCHWABER, K. **Agile Project Management with Scrum**. Redmond: Microsoft Press, 2004.

SERRANO, D. B.; SILVA, D. M.; CAPPELLI, C.; DE ARAUJO, Renata M. 0028/2009 - Problemas na Elicitação de Requisitos: Uma visão de pesquisa/literatura. **RelaTe-DIA**, *[S. l.]*, v. 3, n. 1, 2009.

SEVERINO, A. J. **Metodologia do Trabalho Científico**. 1ª edição. São Paulo: Cortez, 2013.

SOMMERVILLE, I. **Engenharia de Software**. 9ª edição. São Paulo: Pearson Prentice Hall, 2011.

STACK OVERFLOW. **2021 Developer Survey**. Stack Overflow, 2021. Disponível em: <https://insights.stackoverflow.com/survey/2021#overview>. Acesso realizado em 16 de jul. de 2021.

WATERS, T.; WATERS, D. **Weber's Rationalism and Modern Society:** New Translations on Politics, Bureaucracy, and Social Stratification. Londres: Palgrave Macmillan, 2015.

WEBER, M. K. E. **Economia e Sociedade:** Fundamentos da Sociologia Compreensiva. Tradução de Regis Barbosa e Karen Elsabe Barbosa. Brasília: Editora Universidade de Brasília, 1999.

WEISS, R. M. Weber on bureaucracy: management consultant or political theorist?. **Academy of management review**, v. 8, n. 2, p. 242-248, 1983.

WITZEL, M. **A History of Management Thought**. Abingdon: Routledge, 2012.