CENTRO UNIVERSITÁRIO UNIDADE DE ENSINO SUPERIOR DOM BOSCO
ENGENHARIA DE SOFTWARE

DIOGO MONDEGO DOS SANTOS

ANÁLISE DE VIABILIDADE DE UMA APLICAÇÃO MOBILE: estudo de caso dos processos correlatos à elaboração de um *software e-commerce* para uma papelaria em São Luís, MA.

São Luís
2023

DIOGO MONDEGO DOS SANTOS

ANÁLISE DE VIABILIDADE DE UMA APLICAÇÃO MOBILE: estudo de caso dos processos correlatos à elaboração de um *software e-commerce* para uma papelaria em São Luís, MA.

Monografia apresentada ao Curso de Engenharia de Software do Centro Universitário Unidade de Ensino Superior Dom Bosco como requisito parcial para obtenção do grau de Bacharel em Engenharia de Software. Orientador: Prof. Esp. Daniel Herrera de Oliveira Lemos.

São Luís

2023

Dados Internacionais de Catalogação na Publicação (CIP)
Centro Universitário - UNDB / Biblioteca

Santos, Diogo Mondego dos

Análise de viabilidade de uma aplicação mobile: estudo de caso dos processos correlatos à elaboração de um software e-commerce para uma papelaria em São Luís, MA. / Diogo Mondego dos Santos. __ São Luís, 2023.

68 f.

Orientador: Prof. Esp. Daniel Herrera de Oliveira Lemos.

Monografia (Graduação em Engenharia de Software) – Curso de Engenharia de Software - Centro Universitário Unidade de Ensino Superior Dom Bosco – UNDB, 2023.

1. Papelaria. 2. Interface. 3. Design. 4. Vendas.
5. Desenvolvimento - Programação - Software. I. Título.

CDU 004.05(812.1)

DIOGO MONDEGO DOS SANTOS

ANÁLISE DE VIABILIDADE DE UMA APLICAÇÃO MOBILE: estudo de caso dos processos correlatos à elaboração de um *software e-commerce* para uma papelaria em São Luís, MA.

Monografia apresentada ao Curso de Engenharia de Software do Centro Universitário Unidade de Ensino Superior Dom Bosco como requisito parcial para obtenção do grau de Bacharel em Engenharia de Software. Orientador: Prof. Esp. Daniel Herrera de Oliveira Lemos.

Aprovada em: _____/____/_____.

BANCA EXAMINADORA:

_____

Prof. Esp. Daniel Herrera de Oliveira Lemos (Orientador)

Especialista em Ciência de Dados

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

_____

Prof. Me. Allisson Jorge Silva Almeida

Mestre em Inteligência Artificial

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

_____

Prof. Esp. Alysson Marquezelli Simeão Almeida

Especialista em Liderança e Gestão Empresarial

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

Dedico de coração este estudo à minha mãe, meu pai e toda a minha família.

AGRADECIMENTOS

Gostaria de expressar meus sinceros agradecimentos a todos que contribuíram para o sucesso deste trabalho de conclusão de curso. Cada um de vocês desempenhou um papel fundamental e merece ser reconhecido.

Primeiramente, agradeço a Deus por ter me concedido sabedoria e força ao longo desta jornada acadêmica. Sua orientação divina tem sido uma fonte de inspiração e motivação constantes.

À minha família, expresso minha gratidão profunda. Vocês estiveram ao meu lado em todos os momentos, oferecendo apoio incondicional, encorajamento e amor. Sem a presença e o suporte de vocês, esta conquista não seria possível. Seu apoio significou tudo para mim e sou eternamente grato.

Não posso deixar de mencionar meu orientador, Daniel, que desempenhou um papel essencial na minha jornada acadêmica. Sua orientação experta, conhecimento profundo e paciência infinita foram inestimáveis. Suas contribuições críticas e valiosas moldaram este trabalho e me ajudaram a alcançar um nível superior de qualidade. Sou grato por ter você como meu guia e mentor.

Também gostaria de agradecer a todos os meus colegas pelo companheirismo e apoio ao longo dessa caminhada. As discussões, trocas de ideias e colaboração mútua foram fundamentais para o meu crescimento pessoal e acadêmico. Obrigado por compartilharem suas perspectivas e conhecimentos, tornando esta experiência ainda mais enriquecedora.

Por fim, quero expressar minha gratidão a todos os demais professores, amigos e entes queridos que contribuíram de alguma forma para o meu sucesso neste trabalho de conclusão de curso. Suas palavras de encorajamento, críticas construtivas e apoio contínuo foram inestimáveis.

Este trabalho é o resultado de uma jornada coletiva, e agradeço do fundo do meu coração a todos que estiveram ao meu lado, direta ou indiretamente. O conhecimento adquirido, as amizades cultivadas e as lições aprendidas permanecerão comigo para sempre. Sou grato por ter sido abençoado com uma rede de apoio tão incrível.

“O sucesso de uma aplicação mobile não reside apenas em sua funcionalidade, mas também na compreensão profunda dos processos de desenvolvimento e nas decisões estratégicas tomadas ao longo do caminho.”

(PRESSMAN, 2014).

RESUMO

O objetivo deste estudo consiste em analisar as possibilidades de aplicação das técnicas de Engenharia de Software nos processos relacionados ao desenvolvimento de um *software* de papelaria digital, a fim de melhorar a experiência do usuário e aumentar a conversão de vendas. O intuito é aprimorar a experiência do usuário por meio da utilização de modernas técnicas de UI (*user interface*), UX (*user experience*) e desenvolvimento, contemplando temas como a componentização dos elementos, usabilidade, acessibilidade e personalização da jornada do usuário. A gestão do projeto , a fim de alcançar todos os objetivos, também constitui parte importante da elaboração do estudo e fora devidamente considerada. Com o alcançar desses objetivos, serão adotadas práticas de *marketing* que permitam a personalização da jornada do usuário, combinando essas técnicas com conhecimentos de psicologia e programação. Além da pesquisa bibliográfica, será trabalhada a prototipação de forma experimental a fim de ilustrar técnicas e preceitos demonstrados de forma prática.

Palavras-chave: Papelaria. Interface. *Design*. Vendas. Programação.

ABSTRACT

The objective of this study is to analyze the possibilities of applying Software Engineering techniques in the processes related to the development of a digital stationery software in order to improve the user experience and increase sales conversion. The intention is to enhance the user experience through the use of modern UI (user interface), UX (user experience), and development techniques, encompassing topics such as componentization of elements, usability, accessibility, and customization of the user journey. Project management, in order to achieve all objectives, also constituted an important part of the study's development and was duly considered. In order to achieve these objectives, marketing practices will also be adopted to allow for the personalization of the user journey, combining these techniques with knowledge of psychology and programming. In addition to the literature review, experimental prototyping will be employed to illustrate techniques and principles demonstrated in a practical manner.

Palavras-chave: Stationery. Interface. Design. Sell. Programing.

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CEP	Código de Endereçamento Postal
IDE	Integrated Development Environment
PEGN	Pequenas Empresas & Grandes Negócios
PWA	Progressive Web Applications
RN	React Native
SEBRAE	Serviço Brasileiro de Apoio às Micro e Pequenas Empresas
UI	User Interface
UX	User experience

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Devido ao avanço tecnológico, os dispositivos móveis conectados à internet, como *smartphones* e *tablets*, têm se tornado cada vez mais presentes na vida das pessoas, impulsionando a criação de diversas soluções que visam tornar o dia-a-dia mais conveniente e eficiente (SILVA; ALMEIDA, 2019, p. 23).

Conforme a Comscore (2018), mais de 80% do tempo despendido em *smartphones* é com aplicativos, em um mercado que, segundo a Statista (2020), contava com mais de 3,2 bilhões de usuários conectados até o final de 2019. De acordo com a Santos (2020), o brasileiro passa em média três horas por dia utilizando aplicativos no smartphone e mantém cerca de 70 a 80 aplicativos instalados, com um uso médio real de 30 apps. A presença dos dispositivos conectados à rede mundial de computadores pode exercer grande influência na sociedade em virtude da diversidade de soluções presentes nos aplicativos.

O comércio é uma das atividades que mais se beneficia com a evolução tecnológica dos aplicativos em *smartphones*. O uso dessa ferramenta para compra e venda tem se tornado uma tendência cada vez mais comum e bem recebida pelo público, constituindo o chamado *e-commerce*. Segundo Albertin (2004), o comércio eletrônico é uma atividade de compra e venda realizada com o auxílio de recursos eletrônicos.

Esse tipo de mercado vem se consolidando como uma ferramenta estratégica para gestores, sobretudo a partir dos anos 2000, quando se tornou uma fonte de inovação e de melhores resultados para as empresas. O uso de sites e aplicativos tem se tornado uma grande aliada dos varejistas tanto em relação aos processos de gestão quanto na facilitação da comercialização de produtos e serviços (SANTOS, 2013, p.43).

Entretanto, os gestores empresariais muitas vezes não conseguiram antecipar a magnitude da expansão do mercado *on-line* (BARONE; BONFANTE; SCHNEIDER, 2012). Embora muitas empresas tenham reconhecido a necessidade de ingressar nesse segmento, muitas delas o fizeram sem planejamento adequado, sem considerar fatores como escopo, custos e requisitos, presumindo que sua experiência com lojas físicas seria suficiente para garantir o sucesso no *e-commerce*.

Essa abordagem muitas vezes resulta na perda de clientes potenciais que esperam uma experiência de compra mais ágil e conveniente (BRITO; GIOVANINI, 2012, p. 24), isto pois criar um *e-commerce* envolve uma série de questões que vão muito além de “criar um site” ou “mudar o negócio para a internet”, a experiência do usuário se faz fundamental não só para a conversão de vendas, mas também para a fidelização e, por lógica, a manutenção do *e-commerce*. Portanto, a complexidade persiste no amplo conjunto de elementos que compõem toda a experiência do usuário.

Um exemplo recente que mostrou de forma patente o potencial não explorado quando falamos em *e-commerces*, foi o advento da Pandemia Global da COVID-19, diversos negócios foram forçados a migrar suas operações para meios estritamente digitais. Estima-se ainda que, mesmo após o fim das restrições de circulação, o *e-commerce* ainda tem um grande potencial a ser explorado, tornando as empresas do setor favoritas entre os investidores (BANCO DO BRASIL, 2023).

No Brasil, o crescimento do comércio eletrônico em 2020 foi de 68% (sessenta e oito por cento) em relação ao ano anterior, segundo a Associação Brasileira de Comércio Eletrônico (ABCOMM) em parceria com a Neotrust. Com um salto no faturamento, o comércio eletrônico fez com que as empresas se reinventassem para manter o ritmo de vendas mesmo com a pandemia. Isso mostra como a pandemia acelerou o crescimento do *e-commerce* e como é importante para as empresas investirem em tecnologia e inovação para se manterem competitivas neste mercado.

De acordo com dados do Serviço Brasileiro de Apoio às Micro e Pequenas Empresas (SEBRAE), os pequenos negócios representam cerca de 98% (noventa e oito por cento) das empresas registradas no país (SEBRAE, 2021). Isso evidencia a importância desses empreendimentos para a economia brasileira, uma vez que geram empregos e contribuem para a diversificação do mercado (FERRARI, 2020).

A importância dos pequenos negócios na economia também está relacionada à sua capacidade de adaptação. Segundo um artigo publicado na Revista Pequenas Empresas & Grandes Negócios (PEGN), os pequenos negócios têm uma estrutura mais enxuta, o que lhes permite serem ágeis e flexíveis em relação às mudanças do mercado (DIAS, 2021). Dessa forma, os pequenos

negócios conseguem oferecer soluções customizadas aos seus clientes e se diferenciar da concorrência (FERRARI, 2020).

Nesse contexto, a digitalização dos pequenos negócios é uma estratégia essencial para a sua sobrevivência e sucesso. Segundo uma pesquisa do SEBRAE, o número de micro e pequenas empresas que adotaram a internet como canal de venda aumentou 21,6% (vinte e um por cento) durante a pandemia da COVID-19 (SEBRAE, 2020). Isso reforça a importância da presença *on-line* para esses empreendimentos, uma vez que a internet se tornou uma das principais formas de acesso aos produtos e serviços (JORNAL DO COMMERCIO, 2021).

Por fim, é fundamental que os empreendedores invistam em estratégias de *marketing* digital e presença *on-line* para ampliar as oportunidades de negócio. Como afirma um artigo publicado na Forbes Brasil, a digitalização permite que os pequenos negócios alcancem um público maior e diversificado, além de possibilitar a criação de relacionamentos mais duradouros com os clientes (FREITAS, 2020).

Sendo assim, a digitalização pode ser vista como uma ferramenta para o crescimento e sucesso dos pequenos negócios no mercado atual. Com o intuito de proporcionar maior comodidade ao consumidor desta papelaria, sendo uma solução tecnológica foi concebida, visando assim atender a um público que valoriza a praticidade e conveniência na aquisição de produtos e serviços (TAVARES, 2014, p.15).

A criação de um software de e-commerce específico para uma papelaria em São Luís, MA, busca aproveitar essa oportunidade e fornecer uma solução eficiente e acessível para os clientes, permitindo que eles adquiram produtos de maneira segura e conveniente, mesmo em momentos de restrição de mobilidade.

Para atingir tal fim, a metodologia adotada fora a revisão bibliográfica, que propositadamente identifica vantagens e desvantagens do comércio eletrônico para a papelaria, analisa e identifica os requisitos do projeto, sempre percorrendo um rol das técnicas envolvidas e consagradas pelos autores consagrados da Engenharia de Software, adaptando tais conhecimentos coletados com o intuito de abranger o máximo de questões possíveis que se relacionam com o escopo definido.

1.1 OBJETIVOS

1.1.1 Geral

Analisar e, preliminarmente, trabalhar o processo de prototipação e desenvolvimento e uma solução mobile de um *e-commerce* de uma papelaria, sendo esta situada em São Luís - MA, verificando os requisitos do projeto e visando a eficácia da solução desenvolvida.

1.1.2 Específicos

- Evidenciar vantagens concretas na utilização de um Processo de Software adequado às necessidades do projeto.
- Evidenciar as principais demandas no que tange aspectos de psicologia e marketing atreladas a boas práticas de *User Interface* e *User Experience* no desenvolvimento do e-commerce da papelaria.
- Descrever quais decisões se mostraram mais promissoras ante aos requisitos levantados.

2 REFERENCIAL TEÓRICO

A Engenharia de Software é uma disciplina que abrange todos os aspectos da produção de software, desde a especificação inicial do sistema até a sua manutenção após o uso. Ela busca aplicar teorias, métodos e ferramentas para encontrar soluções para os problemas encontrados durante o processo de desenvolvimento, enquanto trabalha dentro das restrições organizacionais e financeiras (Sommerville, 2011, p.5). Além disso, a Engenharia de Software também envolve atividades correlatas, como gerenciamento de projetos e desenvolvimento de ferramentas, métodos e teorias para apoiar a produção de software.

Para atingir seus objetivos, a Engenharia de Software faz uso de diversas técnicas e ferramentas, como a elicitação de requisitos, a documentação do software e rotinas de testes de software (PRESSMAN, 2011, p.13). Essas técnicas permitem mensurar os resultados obtidos durante o processo de desenvolvimento de uma solução tecnológica através de métricas e indicadores objetivos (SOMMERVILLE, 2011, p.36). Isso permite que os engenheiros de software executem o processo de desenvolvimento em sua plenitude, tornando-o cada vez mais eficiente e eficaz.

No contexto proposto por este estudo, ao planejar-se uma aplicação móvel de e-commerce, é de extrema importância a clareza proporcionada por todas as técnicas de Engenharia de Software de que se pode dispor, pois é, fundamentado nas especificações iniciais que virão a ser coletadas e no trato para com elas, que o alicerce da aplicação será desenvolvido (SOMMERVILLE, 2011, p.36).

É somente com o conhecimento de tais necessidades e objetivos que a parte prática do projeto poderá ser desempenhada em sua plenitude, como será melhor abordado daqui em diante, é através dos processos de software — como é o caso da elicitação *a priori,* e da implementação a *posteriori* —, é que requisitos como a pesquisa de produtos, o carrinho de compras, e opções de pagamento diversificadas, poderão tornar-se tangíveis. Assim como tais práticas nortearão escolhas como a plataforma, linguagem, testes, bibliotecas, frameworks e diversos outros aspectos correlatos.

Portanto, a Engenharia de Software fornece uma abordagem sistemática e disciplinada para o desenvolvimento da aplicação de e-commerce de uma papelaria, garantindo que os requisitos sejam atendidos, as restrições sejam respeitadas e que a aplicação seja de qualidade, segura e eficiente.

2.1 Processos de Software

São definidos como processos de *software* uma série de atividades inter-relacionadas que visam o desenvolvimento de *software* de alta qualidade, dentro de um prazo determinado. Eles englobam desde a análise dos requisitos, até a manutenção/evolução. A utilização de um processo de *software* adequado é fundamental para garantir a qualidade do produto e para reduzir o risco de falhas que possam gerar prejuízos financeiros e de reputação para a empresa, como diz (PRESSMAN, 2016, p.18), "o processo de *software* é um conjunto de atividades que leva à produção de um produto de *software*".

Sommerville (2011, p.18), por sua vez, oferece-nos a percepção de que os "processos de *software* são os meios pelos quais podemos gerenciar a complexidade do desenvolvimento de *software*", ou seja, ele destaca que os Processos de Software são um conjunto de ferramentas gerenciais que nos proporcionam meios de lidar, de forma inteligente e prática, com as demandas e requisitos de um projeto.

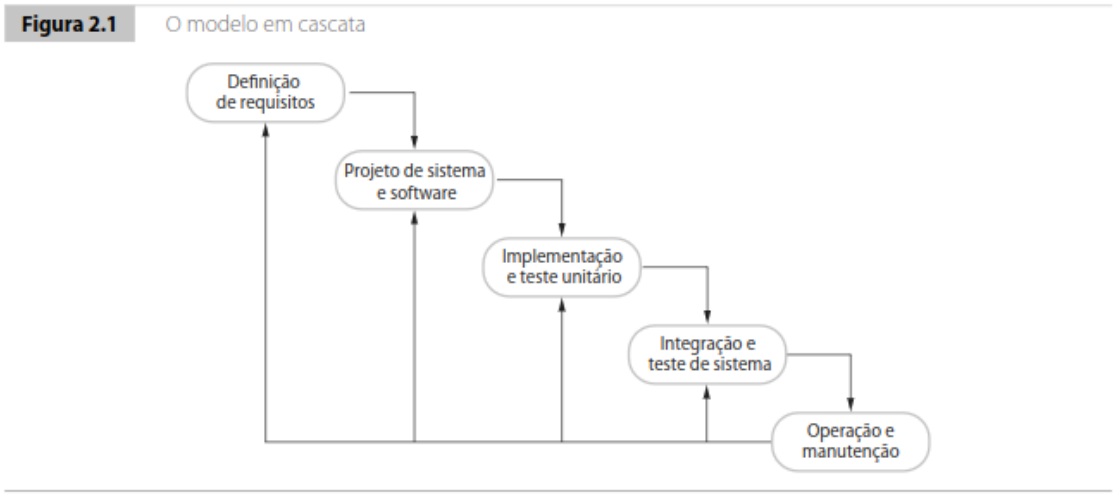
O modelo cascata é um modelo clássico de desenvolvimento de software que segue uma abordagem sequencial e linear, dividindo o processo em fases distintas. Como destacado por Pressman (2016) em seu livro "Engenharia de Software", o modelo cascata é caracterizado pela sequência fixa de fases, começando pela análise de requisitos, passando pelo *design*, implementação, testes e finalizando com a manutenção.

Essa abordagem proporciona clareza na definição dos requisitos e permite um planejamento detalhado. No entanto, sua rigidez pode dificultar a adaptação a mudanças durante o desenvolvimento. Ou seja, se mostra clara e evidente a imperativa necessidade de as empresas optarem pela adoção de um processo de *software* devidamente adequado para obter sucesso em um projeto.

Em resumo, a utilização de um processo de *software* adequado e a escolha de um modelo de desenvolvimento apropriado, nos fornecem ferramentas de gestão que são essenciais para garantir a qualidade, o prazo e o sucesso de um *e-commerce* de papelaria.

2.1.1 Modelo de Processo em Cascata

Figura 01 - Modelo em Cascata.



Fonte: SOMMERVILLE,(2011, p. 20).

O Modelo de Processo em cascata é, por muitas vezes, denominado vulgarmente como “antigo” e “engessado”, adjetivos que — metaforicamente falando — fazem sim sentido para ilustrar para leigos o seu funcionamento em comparação com metodologias ágeis, mas que, de certa forma, cometem uma injustiça com este referido modelo, sobretudo se considerada a demanda que ele visa atender.

Como ilustra Pressman (2016, p. 37), "o modelo de processo em Cascata é um dos modelos mais antigos e amplamente utilizados no desenvolvimento de *software*" — entretanto, é preciso recordar que, um método ser novo ou antigo nada nos diz com relação à sua eficiência e eficácia. O autor detalha ainda que "[o Modelo de Cascata] é um modelo sequencial, ou seja, cada fase é executada em ordem e as atividades de uma fase são concluídas antes de passar para a próxima fase" (PRESSMAN, 2016, p. 38).

Esta dinâmica pode sim dar uma impressão de lentidão nos processos, afinal, as etapas estão sempre em uma relação dependência, por outro lado, tal relação entre etapas pode representar uma melhor estruturação do projeto como um todo, proporcionando maior clareza e controle, o que pode trazer impactos realmente significativos em termos de manutenção e evolução do *software*.

De forma mais detalhada, temos em Sommerville (2011, p. 38) que, "o modelo de processo em cascata é dividido em várias fases, sendo a primeira delas a

definição de requisitos, onde os requisitos do sistema são identificados e documentados". O autor continua sua enumeração de etapas afirmando-nos que "a segunda fase é a fase de projeto, onde é desenvolvido um projeto detalhado do sistema, que define a arquitetura, interfaces e componentes do *software*" (SOMMERVILLE, 2011, p. 38).

Na terceira fase, segundo o autor, temos a fase de implementação, nela o código-fonte é escrito de acordo com as especificações definidas nas fases anteriores. A quarta e penúltima fase, afirma Sommerville (2011, p. 38), é a fase de teste, onde o *software* é testado para garantir que ele atenda aos requisitos do sistema. Mas, assim como qualquer escolha a ser feita em um projeto, é preciso ponderar as vantagens e desvantagens, e com o modelo de cascata isto não se faz diferente, por exemplo, no campo das desvantagens Pressman (2016, p. 44) afirma que,

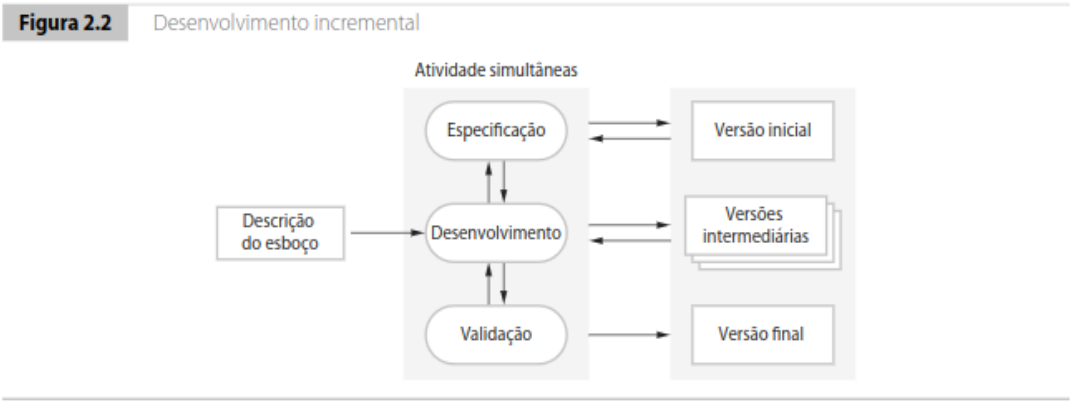
> [...] embora o modelo de processo em Cascata tenha sido amplamente utilizado no passado, ele apresenta algumas limitações. Uma das principais desvantagens é que ele não permite a revisão e ajuste dos requisitos à medida que o processo de desenvolvimento progride. [...] isso significa que se houver algum problema ou mudança de requisitos, a fase de definição de requisitos deve ser reiniciada, o que pode levar a atrasos no projeto e aumento de custos. Outra desvantagem é que esse modelo não permite muita flexibilidade e adaptabilidade, uma vez que cada fase depende da fase anterior, tornando difícil voltar atrás em caso de mudanças. (PRESSMAN, 2016, p. 44-45).

Apesar das desvantagens apresentadas, o Modelo de Processo em Cascata continua sendo utilizado em muitos projetos de desenvolvimento de *software*, principalmente em projetos de pequeno e médio porte, onde os requisitos são relativamente simples e bem definidos (PRESSMAN, 2016, p. 44-45).

Embora o Modelo Cascata apresente limitações, como a falta de flexibilidade para lidar com mudanças de requisitos, ele ainda pode ser aplicado em projetos de *e-commerce* de papelaria de menor escala, onde os requisitos são relativamente simples e bem definidos. No entanto, é importante considerar outras abordagens, como metodologias ágeis, para projetos mais complexos ou sujeitos a mudanças frequentes nos requisitos.

2.1.2 Modelo de Processo Incremental

Figura 02 - Modelo Incremental.



Fonte: SOMMERVILLE,(2011, p. 22).

Uma das principais vantagens do modelo incremental é que ele permite que o *software* seja desenvolvido em partes menores o que permite que o cliente tenha uma visão mais clara do progresso do projeto e pode fornecer *feedbacks* mais cedo no processo de desenvolvimento (SOMMERVILLE, 2011, p. 32).

Sommerville afirma ainda que:

> [...] o modelo incremental é um modelo de processo de *software* que se baseia em uma abordagem iterativa e incremental para o desenvolvimento de *software*. Nesse modelo, o *software* é desenvolvido e entregue em partes, ou seja, em incrementos. Cada incremento é uma nova versão do *software* que adiciona novas funcionalidades ou melhora as funcionalidades já existentes. (SOMERVILLE, 2011, p .32)

Isso significa que o *software* pode ser desenvolvido de forma mais flexível e adaptativa, levando em consideração empecilhos e oportunidades de melhoria que surgirem durante o intercurso do projeto, o que é especialmente útil em projetos em que os requisitos possuem incerteza ou que estejam sujeitos a muitas mudanças drásticas (SOMERVILLE, 2011, p. 32).

De acordo com Pressman (2016, p. 106),

> [...] o modelo incremental divide o processo de desenvolvimento em várias iterações. Cada iteração corresponde a um incremento do *software* e envolve atividades de planejamento, análise, projeto, implementação e testes. A cada iteração, o *software* é testado e validado para garantir que ele atenda aos requisitos estabelecidos.

Baseado nisto, é seguro afirmar que o modelo incremental permite que os desenvolvedores possam corrigir problemas no processo de desenvolvimento, reduzindo custos e o tempo gasto na correção de problemas (SOMMERVILLE, 2011, p. 32).

> [...] Modelos de processos sequenciais, como o modelo cascata, não são capazes de lidar com mudanças frequentes, imprevisibilidade e incerteza, além de terem uma visão bastante restrita da qualidade de *software*. Por outro lado, modelos ágeis, como o Scrum, são indicados para projetos em que a incerteza é alta, as mudanças são frequentes e a colaboração é fundamental. (SOUZA, 2020, p. 24).

O modelo incremental também apresenta desvantagens, em projetos de escala menor, como a criação de um e-commerce de papelaria, o modelo incremental pode não ser a melhor escolha. Os requisitos geralmente são simples e bem definidos, tornando a abordagem sequencial e linear do modelo cascata mais adequada. Além disso, a flexibilidade e adaptação oferecidas pelo modelo incremental podem não ser necessárias. Portanto, para projetos menores, a simplicidade e a previsibilidade do modelo cascata são mais indicadas.

2.2 Engenharia de Requisitos

2.2.1 Requisitos de Usuário

Segundo LAUREANO et al. (2019, p. 63), "[...] os requisitos de usuário descrevem as necessidades e expectativas do cliente em relação ao *software* que está sendo desenvolvido". Esses requisitos são obtidos por meio da comunicação direta com os usuários finais do *software*, que podem incluir tanto usuários internos quanto externos. É essencial envolver os usuários finais do software na definição dos requisitos de usuário para garantir que suas necessidades sejam contempladas e para assegurar que a equipe de desenvolvimento possua habilidades de comunicação e compreensão dos objetivos do projeto (CARVALHO; RIBEIRO, 2020, p. 32).

Pressman (2016, p. 38) define os requisitos de usuário são obtidos a partir de entrevistas, questionários e *workshops* com os usuários finais e demais partes envolvidas ou com algum interesse quanto ao projeto (*stakeholders*), isto se faz necessário para entender cada necessidade e expectativas relacionadas quanto ao sistema que está sendo desenvolvido e aquilo que ele deve alcançar.

Por sua vez, Sommerville (2015, p. 40) reforça que é importante que esses requisitos sejam obtidos por meio de uma comunicação efetiva com os *stakeholders*, a fim de garantir que as necessidades e expectativas dos usuários sejam contempladas no sistema que está sendo desenvolvido.

O autor também destaca a importância de validar os requisitos de usuário, para garantir que eles sejam precisos e atendam às necessidades dos usuários finais, ou seja, os requisitos precisam ser validados não só por testes, mas também por seus requisitos levantados, baseando-se nas expectativas dos usuários finais. Portanto, conclui-se que os requisitos de usuário são fundamentais para o desenvolvimento de *software* bem-sucedido.

Para o nosso caso abordado, um *e-commerce* de uma papelaria, existem diversos exemplos de requisitos de usuário. Estes podem ser o registro de usuários, catálogo de produtos, carrinho de compras, pagamento seguro, rastreamento de pedidos, avaliações e comentários, integração com redes sociais e atendimento ao cliente.

2.2.2 Requisitos de Software (ou Sistema)

Os requisitos de *software* descrevem as funcionalidades e características que o sistema deve ter para atender às necessidades do cliente e dos usuários finais. Esses requisitos são baseados nas necessidades do próprio *software*, incluindo aspectos técnicos e de negócio. Alguns exemplos de requisitos de *software* podem incluir suporte a múltiplas plataformas, escalabilidade, modularidade etc.

Segundo Audy e Brodbeck (2019, p. 42), os requisitos de sistema são responsáveis por definir as funcionalidades e características específicas que o *software* deve ter para atender às necessidades e expectativas do cliente. Para que o processo de desenvolvimento seja bem-sucedido, é fundamental que esses requisitos sejam claros, completos e precisos, garantindo a satisfação do cliente.

De acordo com Shamieh (2011, p. 24), os requisitos de sistema “são os requisitos que definem o que o sistema deve fazer. Eles iniciam em um alto nível do sistema e são analisados e decompostos para produzir os requisitos dos subsistemas de nível mais baixo”.

Pressman (2016, p. 145), por sua vez, define os requisitos de *software* como "uma descrição detalhada e precisa das funções, serviços e restrições que o sistema deve satisfazer". Segundo o autor, esses requisitos são obtidos por meio de

um processo de elicitação, que envolve a identificação das necessidades e expectativas dos *stakeholders*, incluindo usuários finais, clientes e especialistas no domínio do problema.

Os requisitos de software necessários para o desenvolvimento e funcionamento de um e-commerce de papelaria incluem:

A. Desempenho adequado para lidar com alto tráfego de usuários (disponibilidade);
B. Segurança robusta para proteger dados sensíveis (confiabilidade);
C. Integração com sistemas de pagamento confiáveis (confiabilidade);
D. Escalabilidade para acomodar crescimento gradual (disponibilidade);
E. Gerenciamento eficiente de estoque (confiabilidade);
F. Personalização de pedidos;
G. Compatibilidade com dispositivos móveis (multiplataforma);
H. Integração com mídias sociais;
I. Acompanhamento de pedidos e rastreamento de entregas e
J. Recursos de relatórios e análises para obter insights sobre as vendas.

Estes requisitos visam garantir uma experiência de compra fluida, segura e personalizada, bem como o bom gerenciamento das operações de uma papelaria online.

2.2.3 Requisitos funcionais

De acordo com Shamieh (2011, p. 24-25), os requisitos funcionais devem ser utilizados para “descrever o que o sistema deve fazer, de acordo com certas informações”. Segundo Sommerville (2015, p. 48), os requisitos funcionais são as especificações das funcionalidades que o sistema deve realizar para atender aos objetivos do usuário, usando suas próprias palavras, "as funções que o sistema deve realizar, incluindo suas entradas, processamento e saídas". O autor destaca que esses requisitos são importantes porque definem o que o sistema deve fazer e como ele deve fazer, sendo a base para a implementação do sistema.

Já Pressman (2016, p. 147) complementa que os requisitos funcionais são as funções específicas que o *software* deve executar, geralmente expressas em termos de entrada, processamento e saída de dados, usando sua própria definição, os requisitos funcionais são "declarações das funções que o sistema deve fornecer,

de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações".

No contexto ao qual estamos inseridos para este estudo, esses requisitos são responsáveis por descrever as funcionalidades e serviços que o sistema deve oferecer, como exibir informações em tempo real, permitir a configuração de preferências pessoais dos usuários, tais como produtos de seu interesse, e realizar backups automáticos de dados, como por exemplo, dados das compras realizadas. Portanto, são essenciais para garantir que o software atenda às necessidades do cliente e dos usuários finais, além de funções do sistema.

Pois, como diz Silva e Fernandes (2019, p. 27), "os requisitos funcionais descrevem as funcionalidades e serviços que o *software* deve oferecer para atender às necessidades do cliente e do usuário final". Esses requisitos estão relacionados a processos de negócios e podem incluir exibir informações em tempo real, fazer *backup* automático de dados e permitir a configuração de preferências pessoais dos usuários etc. Diante disso, pode-se concluir que os requisitos funcionais são fundamentais para o desenvolvimento de um *software*, pois definem e corroboram todas as funcionalidades que o sistema deve possuir para atender às necessidades do usuário.

2.2.4 Requisitos não-funcionais

Os requisitos não-funcionais descrevem características do *software* que não estão diretamente relacionadas às suas funcionalidades específicas. Sommerville (2015, p. 46) define requisitos não-funcionais como "restrições sobre o comportamento ou a implementação do sistema". Esses requisitos podem incluir aspectos como desempenho, segurança, usabilidade, manutenibilidade, entre outros.

Pressman (2016, p. 142) complementa que os requisitos não-funcionais podem ser tão importantes quanto os requisitos funcionais, pois podem influenciar diretamente na experiência do usuário com o *software*. Para o autor, esses requisitos podem ser obtidos a partir de padrões de qualidade ou requisitos regulatórios impostos pela indústria ou governo.

Já Shamieh (2011, p. 27) destaca que os requisitos não-funcionais estão diretamente atrelados à ideia de performance e/ou qualidade, além disto existe uma

outra função declarativa destes requisitos, que é limitar o *design* do sistema, através de restrições, aos exemplos citados pelo autor envolvem velocidade, capacidade, confiabilidade, peso, uso e escalabilidade. Baggio (2017, p. 44) complementa que os requisitos não-funcionais podem ser expressos em termos de características, desempenho, segurança, usabilidade, disponibilidade, confiabilidade, entre outros aspectos que são importantes para o cliente.

Exemplos de requisitos não-funcionais podem incluir requisitos de desempenho, como tempo de resposta, de carregamento, escalabilidade do sistema; proteção de dados, autenticação de usuários e prevenção de ataques cibernéticos; intuitividade e acessibilidade, tempo de funcionamento e capacidade de recuperação de falhas (PRESSMAN, 2016, p. 142).

Em termos da elaboração de um software de e-commerce para uma papelaria, podemos destacar que estes requisitos assumem um papel de protagonistas, pois a usabilidade, estabilidade, confiabilidade etc., se relacionam direto com a experiência do usuário, estes são os diferenciais a serem oferecidos ante a concorrência, pois é a experiência do usuário durante sua jornada que implicará na fidelização deste como um cliente.

2.2.5 Elicitação de Requisitos

A elicitação de requisitos é um processo crucial no desenvolvimento de *software*, que tem como objetivo identificar e definir as necessidades e expectativas do cliente em relação ao sistema que está sendo desenvolvido, é nesta etapa que todos os requisitos, sejam estes de *software*, de usuário, funcionais ou não-funcionais são documentados e catalogados. Segundo Sommerville (2015, p. 46), a elicitação de requisitos é um processo complexo e pode envolver diferentes técnicas, como entrevistas, questionários, grupos de foco e prototipagem, que visam entender as necessidades do cliente e definir os requisitos do sistema.

Pressman (2016, p. 142) destaca a importância da comunicação durante o processo de elicitação de requisitos, afirmando que "o processo de comunicação é o componente mais crítico do processo de elicitação de requisitos". Para o autor, a equipe de desenvolvimento deve estabelecer uma comunicação efetiva com o cliente, a fim de entender suas necessidades e expectativas em relação ao sistema que está sendo desenvolvido.

Baggio (2017, p. 44-45) complementa que a elicitação de requisitos deve ser uma atividade contínua e iterativa, que envolve tanto o cliente quanto a equipe de desenvolvimento. Segundo o autor, é importante que a equipe de desenvolvimento esteja preparada para mudanças durante o processo de elicitação de requisitos, sobretudo em se tratando em processos incrementais, visando atender às necessidades do cliente de forma efetiva.

No contexto da criação de um e-commerce de papelaria, a elicitação de requisitos desempenha um papel fundamental. Durante esse processo, é essencial identificar e compreender as necessidades e expectativas dos clientes em relação ao sistema. Isso pode envolver requisitos de software, como funcionalidades específicas do site, facilidade de navegação, opções de pagamento e integração com sistemas de gestão de estoque.

Além disso, requisitos de usuário, como preferências de personalização, opções de busca avançada e acesso. Ao realizar a elicitação de requisitos para um e-commerce de papelaria, é fundamental utilizar técnicas apropriadas, como entrevistas com os clientes, análise de concorrência, pesquisa de mercado e prototipagem.

3 ANÁLISE DA ARQUITETURA DE SOFTWARE

A arquitetura de *software* é um dos elementos fundamentais na construção de sistemas de *software* robustos e confiáveis. Ela representa a estrutura do sistema, descrevendo os componentes, suas interações e responsabilidades. Segundo Bass, Clements e Kazman (2013), "[...] a arquitetura é a organização fundamental de um sistema, incorporando seus componentes, seus relacionamentos entre si e com o ambiente, e os princípios que orientam seu projeto e evolução".

A definição da arquitetura de *software* é um processo complexo e deve ser realizado com cuidado e atenção. Ela deve ser capaz de guiar todo o processo de desenvolvimento do *software*, garantindo que o sistema seja construído de forma eficiente e eficaz. Além disso, ela deve ser capaz de lidar com as mudanças e evoluções que o *software* pode sofrer ao longo do tempo. Como afirma Pressman (2011), "A arquitetura do *software* é a base sobre a qual se constrói um *software* robusto e bem projetado. Ela é importante porque afeta diretamente a qualidade do *software* e sua capacidade de satisfazer as necessidades do usuário final".

Ao criar a arquitetura do e-commerce de papelaria, é preciso considerar os requisitos específicos do negócio, como a exibição e organização dos produtos, o processamento de pedidos, o gerenciamento de estoque, o pagamento online e a interação com os clientes. A arquitetura deve ser projetada para lidar com o grande volume de produtos e pedidos que um e-commerce de papelaria pode ter em demandas específicas e sazonais, como por exemplo, no período pré-volta às aulas.

A arquitetura também envolve criação de interfaces amigáveis, navegação fácil e rápida, filtros de pesquisa eficientes e uma estrutura de categorias bem organizada para facilitar a localização e seleção de produtos. Em resumo, a arquitetura é responsável por definir a estrutura e as interações do sistema, garantindo a robustez, confiabilidade e eficiência do sistema (SANTOS, 2023, p. 87).

3.1 Desenvolvimento nativo

O desenvolvimento nativo para iOS e Android é uma das principais formas de criar aplicativos para dispositivos móveis. Ele permite que os aplicativos sejam criados especificamente para cada sistema operacional, aproveitando suas características e recursos específicos. Isso proporciona um melhor desempenho e experiência para o usuário (Singh, 2017). O desenvolvimento nativo envolve a

utilização de ferramentas e linguagens de programação específicas para cada plataforma, como Objective-C ou Swift para iOS e Java, Kotlin ou C++ para Android.

Embora haja pelo menos 3 (três) linguagens disponíveis para desenvolvimento Android, o Google optou por mudar do Java para o Kotlin como sua "linguagem oficial" devido à sua interoperabilidade com o Java (Singh, 2017). O desenvolvimento nativo do Android oferece recursos avançados, como integração com hardware específico do dispositivo, proporcionando uma melhor experiência do usuário. Outra vantagem do desenvolvimento nativo é a possibilidade de otimização do desempenho do aplicativo.

Segundo Fling (2016, p. 62), o uso do código nativo permite que o aplicativo seja executado diretamente pelo *hardware* do dispositivo, proporcionando um desempenho mais rápido e uma melhor resposta ao usuário. Além disso, a otimização do código pode ser feita de forma específica para cada plataforma, o que garante que o aplicativo tenha um desempenho otimizado em cada sistema operacional (FLING, 2016, p. 62-63).

De acordo com Mendes e Colombo (2019, p. 25),

> [...] o desenvolvimento de aplicativos móveis pode ser realizado de diversas formas, entre elas o desenvolvimento nativo e o desenvolvimento híbrido. O desenvolvimento nativo é feito para cada plataforma de maneira específica, utilizando as linguagens de programação e ferramentas nativas do sistema operacional, garantindo melhor performance e usabilidade. Já o desenvolvimento híbrido utiliza tecnologias web e uma camada de abstração para tornar possível o uso em diferentes plataformas. No entanto, o desenvolvimento híbrido pode apresentar algumas limitações em relação à performance e à qualidade de interface do usuário.

O desenvolvimento nativo para o sistema operacional Android pode enfrentar desafios devido à fragmentação da plataforma. Quiles et al. (2018) destacam que a fragmentação do sistema operacional Android é uma das maiores barreiras para o desenvolvimento de aplicativos nativos para essa plataforma. A presença de diversos fabricantes de dispositivos móveis com diferentes especificações de hardware dificulta a garantia de compatibilidade do aplicativo em todos os dispositivos.

Além disso, as variações nas versões do sistema operacional em cada dispositivo podem afetar o comportamento do aplicativo, demandando esforço adicional no desenvolvimento. Segundo um estudo da Opensignal (2016), a fragmentação do Android é uma das principais razões pelas quais os

desenvolvedores têm dificuldade em fornecer experiências consistentes em todos os dispositivos. Essa situação leva alguns desenvolvedores a preferirem criar aplicativos para iOS, onde a fragmentação de dispositivos não é um problema.

Para minimizar essas desvantagens, uma alternativa é o desenvolvimento híbrido, que utiliza tecnologias como o React Native e o Xamarin para criar aplicativos para iOS e Android de forma mais eficiente. De acordo com Varella (2018), o desenvolvimento híbrido de aplicativos móveis é uma técnica que permite a criação de um único código, que pode ser executado em diversas plataformas móveis, reduzindo os custos e o tempo de desenvolvimento.

Portanto, se a papelaria tiver recursos específicos que precisam ser integrados ao aplicativo, como leitor de código de barras para produtos ou uso de sensores do dispositivo, o desenvolvimento nativo pode ser a melhor opção, como trata-se de uma aplicação inteiramente mobile, voltada para vendas, não se faz necessário. Se a papelaria tiver como objetivo alcançar um amplo público e fornecer uma experiência consistente em diferentes dispositivos como é o caso, o desenvolvimento nativo pode apresentar desafios adicionais.

Outro aspecto importante é a manutenção do aplicativo. Com o desenvolvimento híbrido, as atualizações e correções podem ser aplicadas de forma mais eficiente, uma vez que elas são feitas em um único código-base. Considerados todos os pontos, é inegável que o desenvolvimento híbrido se mostra como a melhor opção para nossa demanda, sobretudo para fornecer uma experiência sólida e unificada.

3.2 Desenvolvimento híbrido

3.2.1 Progressive Web Apps (PWA)

As *Progressive Web Applications* (PWA) são uma alternativa ao desenvolvimento nativo para dispositivos móveis, permitindo que os desenvolvedores criem aplicativos acessíveis a partir de websites. Segundo Zhang e Zhao (2019), as PWA combinam as melhores práticas do desenvolvimento *web* e *mobile*, utilizando recursos como *Service Workers*, *Web App Manifest* e outras tecnologias para oferecer uma experiência de usuário similar à de um aplicativo nativo.

Uma das vantagens das PWA é a sua compatibilidade com dispositivos móveis e desktops, já que são acessíveis a partir de um *website*. Segundo Rossmann (2020), isso permite que os usuários acessem o aplicativo a partir de qualquer dispositivo, sem a necessidade de instalar o aplicativo em seus dispositivos. Além disso, as PWA podem ser facilmente compartilhadas e vinculadas a partir de outras páginas web.

Outra vantagem das PWA é a sua capacidade de ser portada de outros modos de desenvolvimento, como aplicativos nativos e web apps. Segundo Souza et al. (2020), as PWA podem ser criadas a partir de um *website* existente, utilizando tecnologias *web*, como HTML, CSS e Javascript, o que reduz o tempo e os custos de desenvolvimento. Além disso, as PWA podem ser facilmente transformadas em aplicativos nativos a partir de ferramentas como o React Native, Capacitor ou o Ionic.

No entanto, as PWA também apresentam desvantagens. Uma das principais é a sua dependência de recursos *web*, limitando os dispositivos. Segundo Malaquias et al. (2021), as PWA não têm acesso a recursos nativos do dispositivo, como câmera, contatos e armazenamento local. Além disso, as PWA podem apresentar desempenho inferior ao de aplicativos nativos, especialmente em dispositivos mais antigos.

Apesar das desvantagens, as PWA são uma alternativa interessante, especialmente para empresas que desejam disponibilizar seus aplicativos em várias plataformas sem os custos e a complexidade do desenvolvimento nativo. Segundo Rossmann (2020), as PWA são uma tendência em crescimento, especialmente em setores como o comércio eletrônico.

3.2.2 Flutter

Flutter é um *framework* de desenvolvimento de aplicativos móveis, criado pela Google em 2017. Uma das principais características do Flutter é a utilização da linguagem Dart, que foi criada pela própria Google em 2011. A Dart é uma linguagem multiparadigma, fortemente tipada e que possui recursos modernos como modo assíncrono, tipagem estática e *garbage collector* (gerenciamento de memória). A linguagem também oferece suporte a bibliotecas nativas do sistema, como a API (*Application Programming Interface*) de animações do iOS e Android (DART LANG 2023).

Uma das principais vantagens do Flutter é a sua capacidade de criar aplicativos para iOS e Android a partir de um único código base. Isso reduz significativamente o tempo e os custos de desenvolvimento, além de permitir que os desenvolvedores entreguem aplicativos com aparência e desempenho consistentes em ambas as plataformas. O Flutter também oferece um *hot-reload*, que permite que os desenvolvedores visualizem as mudanças em tempo real durante o processo de desenvolvimento, sem a necessidade de reiniciar o aplicativo, sendo, portanto, bem ágil (FLUTTER, 2023).

No entanto, o Flutter também apresenta algumas desvantagens. Como é uma tecnologia relativamente nova, ainda há uma curva de aprendizado para os desenvolvedores, especialmente para aqueles que não estão familiarizados com a linguagem Dart. Além disso, o Flutter ainda não tem uma comunidade tão grande quanto outras tecnologias de desenvolvimento *mobile* (SINGH, 2017).

De acordo com Singh (2017), "[...] o Flutter é uma opção atraente para desenvolvedores que buscam criar aplicativos móveis com uma interface de usuário moderna e de alto desempenho para iOS e Android. O uso de uma única base de código para ambas as plataformas é uma vantagem significativa, reduzindo o tempo e o esforço necessários para o desenvolvimento de aplicativos móveis."

3.2.3 React Native

O React Native é um *framework* de desenvolvimento de aplicativos mobile criado pelo Facebook em 2015, baseado no popular React – para desenvolvimento *web*. Segundo o Facebook (2023), o React Native é "a maneira mais rápida de criar aplicativos móveis para iOS e Android". O desenvolvimento com o React Native é feito utilizando linguagens *web*, como HTML, CSS e Javascript. Conforme afirmado por Stanger et al. (2018), o React Native oferece uma solução para desenvolvimento mobile com a utilização de uma única linguagem de programação para ambas as plataformas, o que permite um desenvolvimento mais rápido e fácil de manutenção.

O React Native permite que os desenvolvedores criem aplicativos para iOS e Android a partir de uma única base de código, economizando tempo e recursos. Para o desenvolvimento no iOS, é necessário utilizar o Xcode, a IDE oficial da Apple, enquanto para o desenvolvimento no Android, é utilizado o Android Studio. De acordo com Helmy e Abdelrazek (2018), o React Native tem a capacidade de "compilar o código para ambas as plataformas, sem a necessidade de escrever

código nativo separado para cada plataforma". Isso significa que o desenvolvimento com o React Native pode ser mais eficiente do que o desenvolvimento nativo para cada plataforma.

Uma das ferramentas mais populares para o desenvolvimento com React Native é o Expo. O Expo é uma plataforma que fornece uma série de recursos para ajudar no desenvolvimento, como uma ferramenta de linha de comando, um aplicativo para visualização do projeto em tempo real, e um conjunto de APIs para o acesso a recursos do dispositivo, como a câmera e o GPS.

Como afirma Miller e Lawless (2020), "o Expo reduz a curva de aprendizado e a complexidade do desenvolvimento React Native, oferecendo aos desenvolvedores uma maneira rápida e fácil de criar aplicativos". No entanto, o uso do Expo pode limitar algumas funcionalidades do React Native, e pode não ser a melhor opção para projetos maiores ou mais complexos.

Outra opção para o desenvolvimento com React Native é o React Native CLI. O React Native CLI é uma ferramenta de linha de comando que permite o desenvolvimento sem o uso do Expo. Como afirmado por Patel e Patel (2018), "o uso do React Native CLI é a escolha mais adequada para desenvolvedores experientes que desejam ter mais controle sobre o processo de desenvolvimento".

Com o React Native CLI (FACEBOOK, 2023), os desenvolvedores têm acesso a um conjunto completo de ferramentas de desenvolvimento, mas também precisam lidar com mais complexidade no processo de desenvolvimento.

Recentemente, o React Native passou a oferecer suporte para a linguagem Typescript. Typescript é uma linguagem que adiciona tipagem estática ao Javascript, tornando o código mais seguro e fácil de ser mantido e evitando erros em tempo de desenvolvimento, ao invés de tempo de execução. De acordo com Davis (2020), "o Typescript é uma ótima opção para desenvolvedores que desejam uma maior segurança no código, e para projetos maiores, onde a manutenção do código pode se tornar um problema". O uso do Typescript no React Native pode trazer diversas vantagens, como a redução de erros de codificação e uma melhor documentação do código.

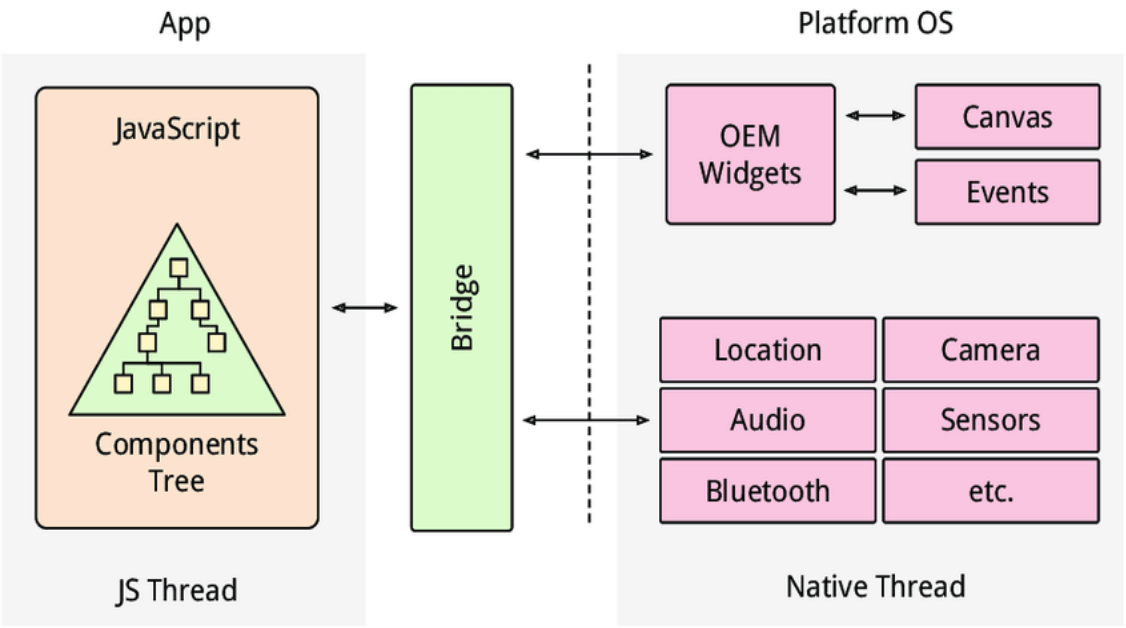
Além disso, o React Native também é conhecido por sua excelente performance, principalmente quando comparado a outras tecnologias híbridas. De acordo com Santos (2019, p. 92), é possível afirmar que o React Native permite a criação de aplicativos móveis tão eficientes e rápidos quanto os aplicativos nativos,

além de ser uma opção mais acessível em termos de tempo e recursos financeiros necessários para o desenvolvimento. Essa eficiência é alcançada devido ao acesso direto às API's nativas do dispositivo, o que permite uma maior integração com os recursos de *hardware*, como a câmera, o acelerômetro, geoposicionamento, entre outros.

No entanto, assim como qualquer tecnologia, o React Native também possui suas desvantagens. Uma das principais críticas em relação ao React Native é sua curva de aprendizado, que pode ser considerada íngreme para desenvolvedores que não estão acostumados com o desenvolvimento em Javascript.

Segundo Skovyra (2019), "o desenvolvimento com React Native exige um conhecimento sólido em Javascript e uma familiaridade com o ecossistema React". Além disso, apesar da grande comunidade em torno do React Native, ainda é possível que alguns recursos ou funcionalidades não estejam disponíveis, o que pode exigir o desenvolvimento de soluções personalizadas ou o uso de bibliotecas de terceiros.

Figura 03 - Bridge de uma aplicação React Native.



Fonte: GIORLAMI; BARSOCCHI; 2021, p.10.

Além disso, o suporte ao Typescript oferece uma vantagem adicional em termos de robustez e escalabilidade. Sua integração com APIs nativas do dispositivo e a possibilidade de compartilhamento de código entre plataformas tornam o React

Native uma opção atraente para empresas e desenvolvedores individuais. No entanto, é importante estar ciente das desvantagens, como a curva de aprendizado íngreme e a possibilidade de recursos limitados. Com uma comunidade ativa e em constante crescimento, o React Native é uma tecnologia promissora para o futuro do desenvolvimento de aplicativos móveis (SKOVYRA, 2019)

De acordo com ROCHA e OLIVEIRA (2020), embora o React Native ofereça diversas vantagens, como o desenvolvimento para múltiplas plataformas a partir de uma única base de código e a utilização de tecnologias *web* conhecidas, é preciso considerar as desvantagens, como a possibilidade de surgirem *bugs* relacionados ao desempenho devido à renderização assíncrona de componentes. Além disso, a curva de aprendizado pode ser um pouco longa, especialmente para aqueles que não estão familiarizados com a estrutura do React. Porém, a comunidade tem trabalhado para solucionar essas questões e oferecer soluções práticas para os desenvolvedores.

Em suma, o React Native é uma ferramenta poderosa para o desenvolvimento de aplicativos móveis, permitindo que os desenvolvedores criem aplicativos para iOS e Android a partir de uma única base de código.

De acordo com Coelho et al. (2021), embora possa haver algumas desvantagens a serem consideradas, como a curva de aprendizado e possíveis problemas de compatibilidade, as vantagens oferecidas, como a possibilidade de reutilização de código e a familiaridade com as tecnologias *web*, fazem dele uma opção cada vez mais popular para o desenvolvimento *mobile*.

Com a crescente comunidade de desenvolvedores e a constante evolução da tecnologia, seja em termos de documentação, suporte, ou novos recursos implementados, é provável que o React Native continue a se destacar como uma ferramenta importante no mundo do desenvolvimento *mobile*, sobretudo valendo-se da popularidade do Javascript, neste caso, do seu *superset*, o Typescript, ou ainda da comodidade de uma linguagem declarativa, como é o caso do HTML.

3.2.4 Ponderações acerca da escolha

O React Native tem sido uma escolha cada vez mais popular para o desenvolvimento de aplicativos móveis, especialmente para aqueles que precisam ser lançados em várias plataformas. Se você está procurando uma maneira eficiente

de construir um aplicativo de *e-commerce* que não precise de um desempenho tão preciso quanto uma aplicação nativa, o React Native pode ser a escolha mais adequada para você (NASCIMENTO, 2020, p. 32).

Uma das principais vantagens do React Native é que ele permite que você crie aplicativos para iOS e Android usando uma única base de código, o que economiza tempo e esforço. Além disso, se você já tem conhecimentos prévios de HTML, CSS, Javascript, Typescript e React, o aprendizado do React Native será mais fácil, já que ele usa uma sintaxe semelhante à do React e de outros *frameworks* baseados em Javascript.

Outra vantagem particular do desenvolvimento com React Native é a facilidade de manutenção e evolução do *software*. Isso se deve ao fato de que a estrutura do React Native é modular e "componentizada", o que torna a reutilização de código muito mais fácil. Além disso, o React Native possui uma documentação extensa e uma grande comunidade. Outro ponto importante é que as atualizações do React Native são feitas de forma incremental (SKOVYRA, 2019).

Além disso, uma vantagem da facilidade de manutenção e evolução de softwares desenvolvidos com React Native é que "o código pode ser atualizado em tempo real, sem a necessidade de atualizar o aplicativo na App Store ou no Google Play, o que economiza tempo e dinheiro" (NASCIMENTO, 2020, p. 42). Isso é possível graças ao recurso de *hot-reloading*, que permite a visualização imediata das alterações feitas no código.

Outra vantagem de utilizar o React Native em um aplicativo de e-commerce é a facilidade de implementação de recursos e requisitos devido à disponibilidade de bibliotecas na comunidade. Conforme mencionado por J. Rodrigues em seu artigo "Desenvolvimento de Aplicativos *Cross-platform* com *React Native*", o *React Native* oferece uma ampla variedade de componentes prontos para uso que agilizam o processo de desenvolvimento. Além disso, existem diversos frameworks de CSS disponíveis, como o NativeBase e o *React Native Elements*, que auxiliam na melhoria do design e da aparência do aplicativo.

No entanto, é importante lembrar que o *React Native* também tem suas limitações e que, embora o *React Native* possa fornecer uma experiência de usuário semelhante à de um aplicativo nativo, ele pode não se mostrar tão rápido quanto uma aplicação nativa, especialmente em aplicativos que exigem uma alta

performance, mas, para nossa sorte, este não é o caso de uma aplicação de *e-commerce* (NASCIMENTO, 2020, p. 56).

Para além disto, aplicações valendo-se de *frameworks* Javascript, são de uso consagrado no desenvolvimento de soluções para *e-commerces*, sobretudo no que tange o desenvolvimento *mobile*, isto porque tais aplicações não demandam recursos muito complexos, bastando para seu pleno funcionamento, o acesso às funções básicas, como armazenamento e cache de dados, e, de forma muito esporádica, o acesso a dados da câmera ou posicionamento global do cliente (FACEBOOK, 2023).

Uma das escolhas refere-se à utilização de Javascript ou Typescript, que, como visto anteriormente, trata-se de uma versão turbinada do próprio Javascript e que fornece uma tipagem estática para a aplicação. Assim como definir qual será a organização da estrutura visual dos elementos, se será usado CSS puro ou algum tipo de *framework*, considerada ainda a possibilidade de elementos já prontos (STANGER, 2018).

Em conclusão, o React Native é uma escolha adequada para o desenvolvimento de um aplicativo de *e-commerce* que não precisa de um desempenho tão preciso quanto de uma aplicação nativa. A possibilidade de desenvolver para múltiplas plataformas a partir de uma única base de código, as bibliotecas disponíveis na comunidade e a facilidade de implementação de recursos e requisitos constituem uma grande vantagem. Porém, é importante ter em mente que o React Native pode não ser adequado para todas as situações, e que a escolha da melhor abordagem de desenvolvimento deve ser baseada nas necessidades específicas do projeto.

Assim, a escolha do React Native para a criação de um e-commerce proporciona uma economia de tempo e esforço no desenvolvimento, facilidade de manutenção e evolução, agilidade nas atualizações e uma vasta biblioteca de componentes disponíveis. Esses benefícios, aliados à popularidade e à familiaridade com o JavaScript, tornam o React Native uma escolha sólida para o desenvolvimento de um e-commerce eficiente e de qualidade.

3.3 Arquitetura de software para aplicativos React Native

3.3.1 Linguagem

De acordo com Oliveira e Ferreira (2018, p. 38-45), o avanço do Javascript possibilitou sua utilização no desenvolvimento mobile através do React Native. Essa tecnologia permite criar aplicativos para iOS e Android a partir de uma única base de código, aproveitando habilidades adquiridas no desenvolvimento *web* (TYPESCRIPT, 2023).

Com o aumento da popularidade do React Native, a comunidade tem trabalhado para resolver desafios como compatibilidade com bibliotecas nativas e melhorias de desempenho. No entanto, o Javascript evoluiu ao longo dos anos e agora é capaz de realizar uma ampla variedade de tarefas. Conforme Flanagan (2012, p. 3), "o Javascript é adequado para diversas aplicações, desde pequenos programas em páginas web até programas de servidor altamente interativos e escaláveis".

O Typescript, desenvolvido pela Microsoft, é uma linguagem de programação de código aberto que estende o Javascript com recursos opcionais de tipagem estática e outras funcionalidades, tornando o código mais robusto e fácil de manter. Como um superset do Javascript, todo código Javascript é válido no Typescript. Com tipagem estática, é possível especificar os tipos de dados das variáveis, evitando erros comuns de programação (TYPESCRIPT, 2023).

Uma vantagem significativa do Typescript é sua capacidade de identificar erros de programação em tempo de compilação, antes da execução do código. Isso ocorre porque o Typescript verifica o uso adequado das variáveis e a consistência na manipulação dos tipos de dados em todo o código. Isso reduz consideravelmente o tempo gasto na depuração e melhora a qualidade do código (ALMEIDA, 2021).

Figura 04 - Componente de botão em React Native.

```tsx
import React from 'react';
import { View, Button, StyleSheet } from 'react-native';

const App = () => {
  return (
    <View style={styles.container}>
      <Button style={styles.button} title="Comprar"/>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  button: {
    backgroundColor: '#A46758',
    width: 124,
    height: 24,
    borderRadius: 4,
    border:"none",
    color: "white",
    fontFamily: 'Inter',
    fontStyle: 'normal',
    fontWeight: '700',
    fontSize: 18,
    lineHeight: 22,

  },
});

export default App;
```

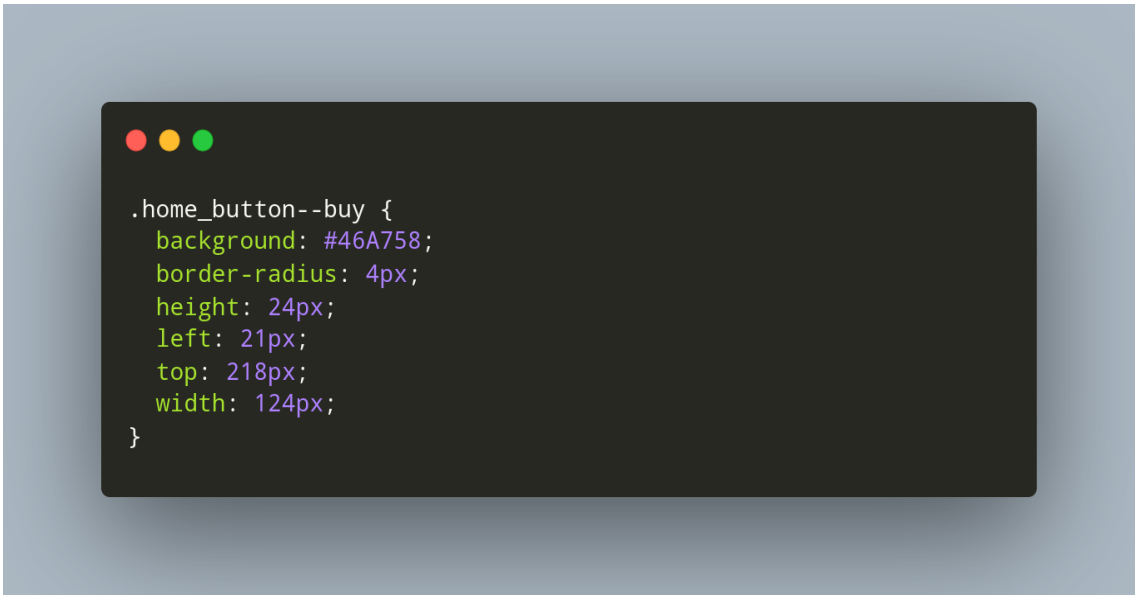
Fonte: O autor (2023).

A integração do Typescript com o React Native é popular entre desenvolvedores devido às vantagens que oferece em relação ao Javascript, como a verificação de tipos estática e o código mais claro e legível. O Typescript é compatível com o React Native desde a versão 0.62, o que torna a integração uma opção viável para projetos atuais e futuros (CANELO, 2021). Outra vantagem é a possibilidade de usar classes em vez de funções para criar componentes. Isso melhora a organização e modularidade do código, permitindo que os desenvolvedores criem componentes mais reutilizáveis.

3.3.2 *Framework* CSS

De acordo com Ferreira (2021), o TailwindCSS é uma opção viável para estilização de componentes em React Native devido à sua compatibilidade com a linguagem. Embora tenha sido originalmente projetado para HTML e CSS, é possível usar as classes de estilo do TailwindCSS diretamente nos componentes React Native, aproveitando o recurso de estilos em cascata. Essa abordagem facilita a criação de interfaces visualmente atraentes e consistentes, economizando tempo e esforço. O TailwindCSS oferece uma ampla variedade de classes de estilo pré-definidas, permitindo que os desenvolvedores estilizem seus componentes de maneira rápida e eficiente.

Figura 05 - CSS do botão comprar.



Fonte: O autor (2023).

Segundo Silva (2021), embora o TailwindCSS tenha sido projetado originalmente para HTML e CSS, ele também é compatível com o React Native. Isso é possível devido à capacidade do React Native de estilizar componentes usando estilos em cascata, semelhantes às classes de estilo no HTML. Com o TailwindCSS, os desenvolvedores podem criar interfaces visualmente atraentes e consistentes em aplicativos da Web e móveis. A modularidade do TailwindCSS facilita a criação de interfaces personalizadas para atender às necessidades específicas de cada plataforma suportada pelo React Native, incluindo iOS, Android e web (CANELO, 2021).

Figura 06 - Componente Button em RN refatorado com TailwindCSS.

```jsx
import React from 'react';
import { View, StyleSheet } from 'react-native';
import { Button } from 'react-native-elements';
import 'tailwindcss/tailwind.css';

export default function App() {
  return (
    <View style={styles.container}>
      <Button
        title="Comprar"
        buttonStyle="bg-green-600 w-28 h-10 rounded-md"
        titleStyle="text-white text-lg font-bold"
      />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
});
```

Fonte: O autor (2023).

3.3.3 Bibliotecas

O desenvolvimento em React e React Native envolve o uso de bibliotecas para tornar o processo mais eficiente e produtivo. Essas bibliotecas aceleram o desenvolvimento e melhoram a qualidade do código. Além disso, permitem a reutilização de código e a colaboração entre desenvolvedores (FERREIRA, 2019).

As bibliotecas de JavaScript para React Native são fundamentais para o desenvolvimento de aplicações móveis. Um exemplo é o Axios, que permite o envio de requisições HTTP no React Native. Segundo sua documentação, o Axios é um "cliente HTTP baseado em Promises para fazer requisições a servidores *web*" (AXIOS, 2023). Ele facilita tarefas como gerenciamento de erros, definição de cabeçalhos e manipulação de dados (PAVANI, 2019).

Outra biblioteca útil é o React Native Tanstack Query, que simplifica o gerenciamento de dados remotos em aplicativos React Native. Ele utiliza o Axios como uma de suas opções de cliente HTTP e oferece recursos como cache de requisições, atualização otimista de dados e gestão de estados de carregamento (GARCIA, 2022).

Para o gerenciamento de rotas, destaca-se o React Navigation, uma biblioteca popular que ajuda na navegação entre telas em aplicações React Native. Com ele, é possível gerenciar diversas opções de navegação, como abas, empilhamento e gaveta (React Navigation, documentação oficial).

Em um e-commerce, a biblioteca de rotas desempenha um papel fundamental ao lidar com a navegação entre os diferentes produtos disponíveis no site. Cada produto em um e-commerce geralmente possui uma página individual dedicada a ele, contendo informações detalhadas, imagens, opções de compra, comentários de clientes, entre outros elementos relevantes. Cada produto pode ter uma rota específica associada a ele, que é ativada quando o usuário clica em um link ou realiza uma pesquisa. Ao utilizar uma rota dedicada para cada produto, é possível garantir uma experiência de usuário fluida e intuitiva.

Com a utilização de rotas específicas para cada produto, torna-se possível compartilhar links diretos para produtos específicos. Isso é útil em campanhas de marketing, mídias sociais ou quando os usuários desejam compartilhar um produto com amigos ou familiares. Ao clicar em um link compartilhado, a biblioteca de rotas permite que o aplicativo navegue diretamente para a página do produto referenciado, melhorando a experiência do usuário.

A biblioteca de rotas também permite implementar roteamento condicional, onde determinadas rotas ou ações de navegação são habilitadas ou desabilitadas com base nessas condições. Isso permite personalizar a experiência do usuário com base no contexto específico, garantindo que apenas rotas relevantes estejam acessíveis para cada usuário.

O gerenciamento de estado é importante em projetos de React Native, e o Jotai é uma opção minimalista para essa tarefa. Ele permite criar e compartilhar estados de forma simples, sem a necessidade de configurações complexas (MANTILLA, 2021).

Com a biblioteca Jotai, é possível, por exemplo, centralizar o estado do carrinho de compras, a autenticação do usuário, o estado do pedido, os filtros e a ordenação, além das notificações. Isso possibilita a interação fluida entre os diferentes componentes do aplicativo, garantindo uma experiência de compra agradável e personalizada. O uso adequado do gerenciamento de estados com o Jotai simplifica o desenvolvimento, a manutenção e a expansibilidade do e-commerce, melhorando sua qualidade e desempenho (JOTAI, 2023).

A biblioteca Helmet é uma escolha popular para gerenciar metadados, pois facilita a definição de títulos, tags, descrições e outros elementos importantes para a indexação e exibição dos produtos nos mecanismos de busca. Com o uso adequado do Helmet, é possível garantir que cada produto tenha metadados únicos e relevantes, melhorando a visibilidade e o posicionamento do e-commerce nos resultados de pesquisa. (KOENIG, 2018).

Apesar da comodidade ao se usar bibliotecas, é importante considerar a segurança e a confiabilidade das mesmas. A manutenção regular e a atualização das bibliotecas são fundamentais para evitar vulnerabilidades conhecidas (XIONG et al., 2021). É necessário avaliar cuidadosamente a necessidade de cada biblioteca, considerando a segurança, manutenção e complexidade do projeto. O uso excessivo de bibliotecas pode resultar em uma solução complexa e de difícil manutenção (BALAGUER et al., 2018).

3.3.4 Otimizações por plataforma

O *React Native* oferece aos desenvolvedores a flexibilidade necessária para otimizar e personalizar o desenvolvimento de aplicativos iOS e Android. Isso é evidente, por exemplo, no gerenciamento de permissões. Com o uso de bibliotecas como '*react-native-permissions*', é possível solicitar permissões específicas do sistema operacional, como acesso à câmera, localização ou notificações.

Conforme mencionado por M. Rahman em seu livro "*React Native in Action*", o *React Native* simplifica a solicitação e o tratamento de permissões, proporcionando uma experiência mais fluida para o usuário final. Além disso, o *React Native* oferece recursos para aprimorar o desempenho e a funcionalidade das aplicações.

Uma maneira de alcançar isso é por meio da utilização de bibliotecas de otimização, como 'react-native-fast-image'. Essas bibliotecas permitem o carregamento e a exibição eficiente de imagens, reduzindo o consumo de recursos e melhorando o tempo de resposta da aplicação. De acordo com A. Tariq et al. em seu estudo "Performance Analysis of React Native for Mobile Application Development", o uso dessas bibliotecas pode levar a uma experiência de usuário mais rápida e responsiva.

O React Native possibilita a criação de componentes reutilizáveis que se adaptam automaticamente a várias telas e resoluções. Dessa forma, é possível criar interfaces consistentes para diferentes tamanhos de tela. Conforme mencionado por S. B. Al-Bassam et al. em seu artigo "A Study on Developing Cross-platform Mobile Applications Using React Native", essa abordagem economiza tempo de desenvolvimento, pois evita a necessidade de criar layouts diferentes para cada dispositivo.

O React Native permite a criação de módulos nativos personalizados, escritos em Java (para Android) ou Objective-C/Swift (para iOS), que podem ser integrados. Essa capacidade de personalização nativa permite aos desenvolvedores aproveitar recursos específicos de cada plataforma, garantindo uma experiência completa. Conforme observado por A. B. Freitas et al. em seu trabalho "Building mobile applications with React Native: An experience report", essa flexibilidade é uma vantagem significativa para o desenvolvimento de aplicativos móveis.

3.3.5 Boas práticas de UI

As boas práticas de UI (User Interface) são essenciais para criar experiências digitais eficientes, intuitivas e agradáveis. Essas práticas abrangem uma variedade de aspectos, desde a organização visual dos elementos na tela até a consistência do *design* em toda a aplicação. De acordo com Treder em seu livro "*Designing Interfaces: Principles of UI Design*", a adoção de boas práticas de UI contribui para uma melhor usabilidade e satisfação do usuário.

Uma boa prática de UI é manter a simplicidade e a clareza visual. Isso significa evitar o excesso de informações e elementos desnecessários na tela. Como mencionado por Tufte em seu livro "T*he Visual Display of Quantitative Information*", a simplificação da interface torna mais fácil para os usuários compreenderem a informação e interagirem com ela de forma eficiente. Utilizar espaçamento

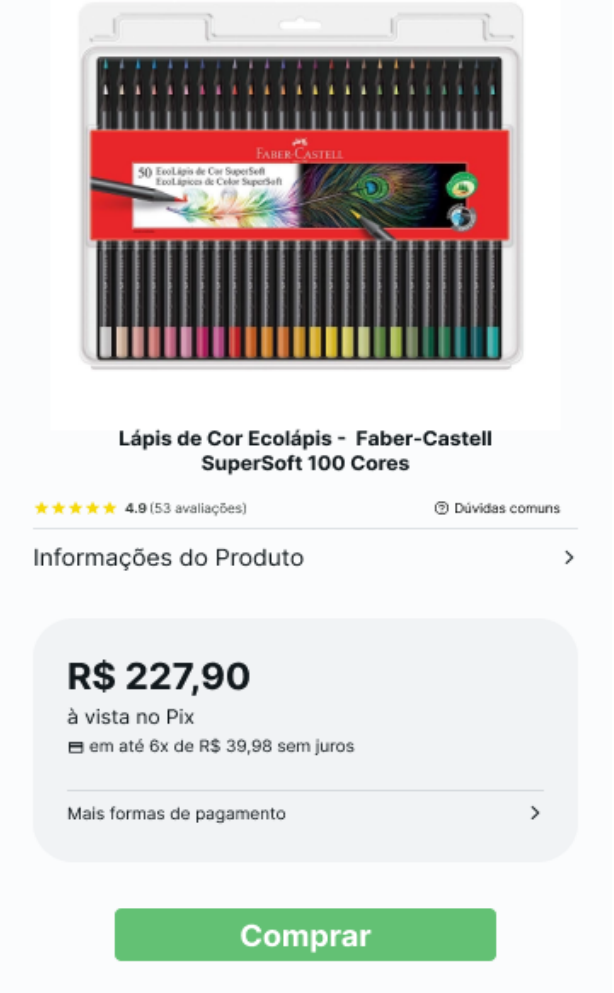
adequado, tipografia legível e cores contrastantes são elementos fundamentais para criar uma interface visualmente agradável e de fácil compreensão.

Figura 07 - Informações do Produto simplificadas para o cliente.

Fonte: O autor (2023).

Outra boa prática é adotar uma hierarquia visual adequada. Isso implica em destacar elementos importantes e reduzir a importância visual de elementos menos relevantes. Conforme sugerido por Cooper et al. em seu livro "*About Face: The Essentials of Interaction Design*", o uso de tamanho, cor e contraste adequados ajuda a direcionar a atenção do usuário para os elementos mais importantes da interface.

Figura 08 - Informações do Produto com Hierarquia trabalhada.



Fonte: O autor (2023).

Consistência é fundamental no design de UI, pois proporciona uma experiência mais coesa e familiar para o usuário. Nielsen destaca em seu livro "*Usability Engineering*" que a consistência ajuda os usuários a aprenderem e se familiarizarem com a aplicação mais rapidamente, aumentando a eficiência e reduzindo erros. Para garantir a consistência visual e comportamental, é importante seguir as diretrizes de design da plataforma (iOS ou Android) e aproveitar os componentes nativos do React Native (Nielsen, Usability Engineering).

A usabilidade é uma prática que deve ser priorizada no *design* de UI. A aplicação deve ser fácil de usar, com fluxos de navegação intuitivos e elementos interativos bem identificados. Conforme afirmado por Biazus et al. (2017) em seu livro "Design de Interação e Interfaces para Dispositivos Móveis", realizar testes de usabilidade e coletar *feedback* dos usuários é essencial para identificar problemas e realizar melhorias.

Além disso, a responsividade é uma prática essencial no *design* de UI para aplicativos React Native. A interface deve se adaptar de forma adequada a diferentes tamanhos de tela e orientações. Como afirmado por Krug (2014) em seu livro "*Don't Make Me Think*", garantir que a aplicação seja visualmente atraente e funcional em dispositivos de diferentes resoluções e proporções é fundamental.

Por fim, é importante realizar testes e iterações constantes ao longo do processo de *design* de UI. A coleta de *feedback* dos usuários e a análise dos dados de uso permitem identificar problemas e oportunidades de melhoria. Como mencionado por Nielsen e Molich em seu artigo "*Heuristic Evaluation of User Interfaces*", a aplicação de heurísticas de usabilidade e a realização de testes com usuários ajudam a identificar problemas de interface e a validar soluções propostas.

Em resumo, a adoção de boas práticas de UI no desenvolvimento de aplicativos React Native contribui para uma experiência de usuário mais eficiente, intuitiva e agradável. Manter a simplicidade visual, adotar uma hierarquia adequada, garantir a consistência, priorizar a usabilidade e a acessibilidade, buscar a responsividade e realizar testes e iterações são fundamentais para criar interfaces de alta qualidade.

3.3.6 Boas práticas de UX

As boas práticas de UX (*User Experience)* desempenham um papel crucial no desenvolvimento de aplicativos React Native, visando proporcionar uma experiência fluida e satisfatória aos usuários. A UX abrange diversos aspectos, desde a usabilidade e eficiência do aplicativo até a emoção e engajamento do usuário. Conforme mencionado por Norman em seu livro "*The Design of Everyday Things*", a adoção de boas práticas de UX é fundamental para criar aplicativos que atendam às necessidades e expectativas dos usuários.

Uma das principais boas práticas de UX é a compreensão do usuário e a criação de personas. Isso envolve a pesquisa e análise dos usuários-alvo, identificando suas características, necessidades, motivações e comportamentos. Conforme afirmado por Zeldman em seu livro "*Designing with Web Standards*", a criação de personas permite que os desenvolvedores tenham uma compreensão mais profunda dos usuários, orientando as decisões de *design* e funcionalidade do aplicativo.

A arquitetura da informação é outra prática essencial no *design* de UX. Isso envolve a organização e estruturação das informações de forma lógica e intuitiva. Conforme mencionado por Morville e Rosenfeld em seu livro "*Information Architecture for the World Wide Web*", uma boa arquitetura de informação facilita a localização e o acesso às informações desejadas, garantindo uma experiência de usuário mais eficiente e agradável.

A personalização é uma prática cada vez mais valorizada na UX. Permitir que os usuários personalizem a interface, como a escolha de temas, configurações de notificações e preferências de exibição, oferece uma experiência mais personalizada e envolvente.

A consistência também desempenha um papel crucial na UX. Manter uma experiência consistente em toda a aplicação, desde a navegação até o *design* visual, cria familiaridade e reduz a curva de aprendizado para os usuários. Conforme mencionado por Nielsen em seu livro "*Usability Engineering*", a consistência garante que os usuários possam transferir seu conhecimento de uma parte da aplicação para outra, facilitando a compreensão e a interação.

A acessibilidade é uma prática fundamental no *design* de UX. Garantir que o aplicativo seja acessível a todos os usuários, incluindo pessoas com deficiências, é uma prioridade. Como mencionado por W3C em suas diretrizes de acessibilidade, é necessário considerar aspectos como contraste adequado, navegação por teclado, suporte a leitores de tela e alternativas para conteúdo multimídia. O foco na acessibilidade proporciona uma experiência inclusiva e garante que todos os usuários possam utilizar o aplicativo de forma efetiva.

Figura 09 - Componente refatorado com atributos de acessibilidade.

```jsx
import React from 'react';
import { View, StyleSheet } from 'react-native';
import { Button } from 'react-native-elements';
import 'tailwindcss/tailwind.css';

export default function App() {
  return (
    <View style={styles.container}>
      <Button
        title="Comprar"
        buttonStyle="bg-green-600 w-28 h-10 rounded-md"
        titleStyle="text-white text-lg font-bold"
        accessibilityLabel="Botão de compra"
        accessibilityRole="button"
        accessibilityHint="Clique para comprar o item"
      />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
});
```

Fonte: O autor (2023).

Garantir que a aplicação seja acessível a todos os usuários, incluindo pessoas com deficiências, é uma obrigação ética e, por muitas vezes, legal. Como mencionado por Crumlish e Malone (2009) em seu livro "*Designing Social Interfaces*", é necessário considerar requisitos de acessibilidade, como o uso de contraste adequado, legibilidade de texto, suporte a leitores de tela e controle por teclado. O React Native oferece recursos nativos de acessibilidade que podem ser utilizados para melhorar a experiência de usuários com deficiência, como a inclusão de rótulos descritivos para elementos de interface e suporte a gestos personalizados.

Por fim, a importância da validação e iteração constante na UX não pode ser subestimada. Realizar testes de usabilidade, coletar *feedback* dos usuários e analisar dados de uso são aspectos essenciais para melhorar continuamente a UX de um aplicativo React Native. Como afirmado por Krug em seu livro "*Rocket Surgery Made Easy*", os testes com usuários permitem identificar problemas de usabilidade e entender as necessidades e expectativas dos usuários de forma direta. A análise dos dados de uso, por sua vez, oferece *insights* valiosos sobre o comportamento dos usuários e a eficácia das soluções implementadas.

Além disso, a otimização de desempenho é uma prática fundamental na UX de aplicativos React Native. O desempenho ágil e responsivo do aplicativo é crucial para proporcionar uma experiência de usuário fluida. Conforme mencionado por McLaughlin em seu livro "*Building React Native Applications*", otimizar o desempenho do aplicativo envolve a redução do tempo de carregamento, a otimização do consumo de recursos e a melhoria da fluidez das animações e transições.

A simplicidade é uma diretriz importante na UX. Simplificar a interação e minimizar a carga cognitiva do usuário são práticas que contribuem para uma experiência mais agradável. Como mencionado por Buley em seu livro "*The User Experience Team of One*", a eliminação de elementos e etapas desnecessárias e a priorização das funcionalidades principais resultam em um aplicativo mais intuitivo e fácil de usar.

Por fim, a empatia é uma prática fundamental na UX. Colocar-se no lugar do usuário e considerar suas necessidades, emoções e contextos ajuda a projetar uma experiência verdadeiramente centrada no usuário. Como afirmado por Cooper (2004) em seu livro "The Inmates Are Running the Asylum", a empatia permite que os desenvolvedores criem soluções que atendam às necessidades reais dos usuários e proporcionem uma experiência que os encante.

Em resumo, a adoção de boas práticas de UX no desenvolvimento de aplicativos React Native é essencial para criar experiências de usuário atraentes e satisfatórias. Compreender os usuários, estruturar a informação, oferecer personalização, garantir consistência, priorizar a acessibilidade, validar e iterar constantemente, otimizar o desempenho, buscar a simplicidade e praticar a empatia são diretrizes fundamentais para proporcionar uma experiência de usuário excepcional.

3.3.7 Prototipação

A prototipação desempenha um papel fundamental no desenvolvimento de aplicativos usando o React Native. Ela ajuda a visualizar e testar ideias de *design*, fluxos de navegação e interações antes de implementá-los completamente. Como afirmado por J. Tidwell em seu livro "*Designing Interfaces: Patterns for Effective Interaction Design*", a prototipação permite que os desenvolvedores iterem e refinem suas ideias, evitando retrabalhos custosos durante o processo de desenvolvimento.

No contexto do React Native, a prototipação é especialmente útil para definir e estruturar componentes. Ela permite que os desenvolvedores experimentem diferentes combinações de componentes e estilos visuais, garantindo a consistência e a usabilidade da aplicação. De acordo com M. D. Morais et al. em seu estudo "*A Comparative Study on Usability Evaluation Methods for Mobile Applications Developed with React Native*", a prototipação ajuda a identificar problemas de usabilidade e aprimorar a experiência do usuário antes de iniciar a codificação.

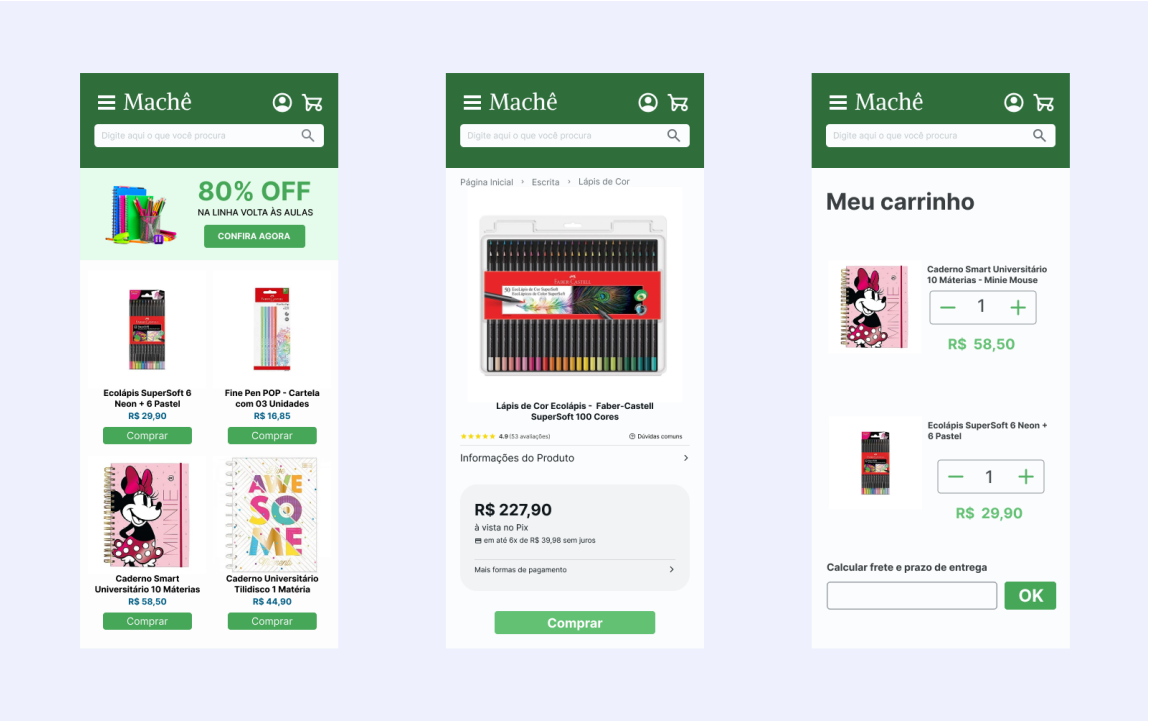
Existem diferentes tipos de protótipos que podem ser usados no desenvolvimento de aplicativos React Native. Entre eles, destacam-se os protótipos de baixa fidelidade, como esboços em papel ou mockups estáticos, e os protótipos de alta fidelidade, que se aproximam visualmente da aparência final da aplicação. Cada tipo de protótipo tem seu propósito e nível de detalhamento. Conforme mencionado por J. Gonzalez em seu artigo "*The Role of Prototyping in UX Design*", a escolha do tipo de protótipo depende da fase do projeto e do objetivo específico de cada iteração.

No estágio inicial do design, os wireframes são frequentemente utilizados como protótipos de baixa fidelidade. Eles representam esboços simplificados da estrutura e do layout da aplicação, destacando a disposição dos elementos na tela. Os wireframes ajudam a definir a arquitetura da informação e a organizar os componentes principais. Segundo A. Cooper et al. em seu livro "*About Face: The Essentials of Interaction Design*", os wireframes são uma ferramenta valiosa para capturar e comunicar conceitos de design de forma rápida e eficiente.

Existem diferentes tipos de *wireframes* que podem ser criados no contexto do desenvolvimento de aplicativos React Native. Os *wireframes* de baixa fidelidade geralmente consistem em esboços simples, usando linhas e caixas para

representar os elementos da interface. Por outro lado, os *wireframes* de alta fidelidade podem incluir elementos visuais mais detalhados, como cores, tipografia e ícones. De acordo com Snyder em seu livro "*Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces*", a escolha do tipo de *wireframe* depende do nível de detalhamento necessário para comunicar adequadamente as ideias de *design*.

Figura 10 - Wireframe de média/alta fidelidade feito no FIGMA.



Fonte: O autor (2023).

Ferramentas como o Figma são amplamente utilizadas no desenvolvimento de aplicativos React Native para criar protótipos e *wireframes*. O Figma oferece recursos de *design* colaborativo, permitindo que várias pessoas trabalhem simultaneamente no mesmo projeto. Além disso, ele possui uma ampla variedade de componentes e recursos que facilitam a criação de interfaces interativas. Conforme mencionado por Bukhari et al. em seu artigo "*Comparing Prototyping Tools for Mobile Application Design*", o Figma é uma ferramenta popular devido à sua facilidade de uso, recursos avançados e capacidade de compartilhamento e *feedback* em tempo real.

Além do Figma, existem outras ferramentas amplamente utilizadas para prototipação de aplicativos React Native. O Sketch, por exemplo, é conhecido por sua interface intuitiva e recursos específicos para *design* de interfaces. O Adobe XD é outra opção popular, oferecendo recursos de prototipação interativa e integração com outros produtos da Adobe. Como mencionado por C. N. Veerappan et al. (2021) em seu artigo "*Evaluating UI/UX Design Tools for Mobile Applications: A Comparative Study*", a escolha da ferramenta de prototipação depende das necessidades e preferências individuais do desenvolvedor.

Em suma, a prototipação permite definir e refinar componentes, testar interações e fluxos de navegação, além de identificar e corrigir problemas de usabilidade antes da implementação final.

4 IMPLEMENTAÇÃO DO SOFTWARE

A implementação de *software* em React Native requer uma combinação de ferramentas e práticas que visam facilitar o desenvolvimento e a manutenção do código. Uma IDE (*Integrated Development Environment)* adequada é essencial nesse processo. Conforme mencionado por M. Farley em seu livro "*React Native for iOS Development*", uma IDE como o Visual Studio Code ou o JetBrains WebStorm oferece recursos avançados de edição de código, depuração e gerenciamento de projetos, tornando o desenvolvimento mais eficiente.

O versionamento do código também desempenha um papel importante na implementação de software em React Native. Utilizar uma ferramenta de controle de versão, como o Git, permite gerenciar as alterações no código e trabalhar de forma colaborativa. Conforme afirmado por Chacon e Straub em seu livro "*Pro Git*", o versionamento adequado do código facilita a colaboração entre os membros da equipe de desenvolvimento, o rastreamento de alterações e a implementação de novas funcionalidades de forma organizada.

A documentação é outro elemento essencial na implementação de software em React Native. Ter uma documentação clara e abrangente ajuda os desenvolvedores a entenderem o funcionamento da aplicação e a utilização de componentes e APIs específicas. Conforme mencionado por Miller em seu livro "*React Native in Action*", a documentação é uma ferramenta valiosa para aprender e dominar as capacidades do React Native, além de permitir uma comunicação efetiva entre os membros da equipe.

Por fim, o fluxo de código é um aspecto a ser considerado na implementação de software em React Native. Seguir boas práticas de organização, como a separação em componentes reutilizáveis, a utilização de padrões de projeto e a aplicação de testes automatizados, contribui para um fluxo de código mais eficiente e sustentável. Conforme afirmado por Abramov e Lakatos em seu livro "*React in Action*", um fluxo de código estruturado e bem definido facilita a manutenção, a identificação de erros e a escalabilidade do projeto.

4.1 Fluxo de Código

O fluxo de código desempenha um papel fundamental na implementação de um software em React Native para uma papelaria. O processo começa com o usuário realizando um breve cadastro, fornecendo informações básicas como nome,

e-mail e senha. Conforme mencionado por Eisenman em seu livro "*Learning React Native*", o cadastro do usuário pode ser implementado utilizando componentes de formulário e validação de entrada para garantir a integridade dos dados fornecidos.

Após o cadastro, o software precisa autenticar o usuário utilizando uma API. Conforme afirmado por Miller em seu livro "*React Native in Action*", a autenticação pode ser realizada através do envio das credenciais do usuário para a API e a resposta da API validando essas credenciais. Isso garante que apenas usuários autenticados tenham acesso às funcionalidades do software. Uma vez autenticado, o software irá acessar a API novamente para obter os produtos disponíveis na papelaria.

Para melhorar o desempenho e a experiência do usuário, alguns dados do usuário e informações de cache podem ser armazenados localmente no dispositivo. Isso inclui dados como os itens do carrinho de compras. Conforme afirmado por Abramov em seu artigo "*Presentational and Container Components*", a separação entre componentes de apresentação e componentes de contêiner permite armazenar e sincronizar os dados localmente, proporcionando uma resposta mais rápida e off-line para o usuário.

No entanto, é importante sincronizar esses dados com a API on-line para garantir que qualquer alteração feita pelo usuário seja refletida em tempo real. Conforme mencionado por Amundsen em seu livro "*RESTful Web Clients*", essa sincronização pode ser realizada utilizando estratégias como a atualização periódica dos dados locais com os dados da API ou a sincronização sob demanda quando o usuário realizar ações específicas.

Por fim, é essencial implementar testes automatizados para verificar o fluxo de código e garantir que todas as etapas funcionem corretamente. Conforme afirmado por Lott em seu livro "*Modern Python Cookbook*", a utilização de testes unitários e testes de integração ajuda a identificar e corrigir possíveis problemas no fluxo de código, garantindo a qualidade do *software*.

4.2 Teste de software

Testes de software desempenham um papel crucial na implementação de um software de papelaria em React Native. Durante um aumento sazonal nas vendas, como feriados ou promoções especiais, é importante realizar testes de estresse para avaliar a capacidade do sistema em lidar com um grande volume de

transações. Conforme mencionado por Beck em seu livro "*Test-Driven Development*", os testes de estresse ajudam a identificar possíveis gargalos e problemas de desempenho, garantindo que o software seja capaz de suportar uma demanda intensa.

Além dos testes de estresse, os testes unitários são essenciais para a manutenção e evolução do software. Eles envolvem a verificação do comportamento de módulos individuais ou componentes isolados do aplicativo. Conforme afirmado por Martin em seu livro "*Clean Code: A Handbook of Agile Software Craftsmanship*", os testes unitários garantem que cada módulo funcione corretamente e que eventuais mudanças ou melhorias não afetem negativamente o funcionamento das partes existentes do software.

Os testes de integração também são relevantes no contexto de um software de papelaria em React Native. Eles envolvem a verificação do correto funcionamento das diferentes partes do sistema quando integradas. Conforme mencionado por Fowler em seu livro "*Patterns of Enterprise Application Architecture*", os testes de integração ajudam a identificar problemas que podem surgir devido à interação entre os componentes, como erros na comunicação com a API, inconsistências nos dados ou comportamentos inesperados.

Além dos testes de estresse, testes unitários e testes de integração, existem outros tipos de testes que podem ser adequados ao caso mencionado. Por exemplo, os testes de aceitação podem verificar se o *software* atende aos requisitos de negócio e se funciona corretamente para o usuário final. Já os testes de usabilidade permitem avaliar a facilidade de uso e a experiência do usuário do software em um ambiente real. Conforme afirmado por Krug (2014) em seu livro "*Don't Make Me Think*", os testes de usabilidade são fundamentais para identificar problemas de *design* e garantir uma experiência satisfatória para os usuários.

4.3 Manutenção e Evolução de Software

A manutenção e evolução de um *software* de papelaria em React Native são aspectos fundamentais para garantir seu bom funcionamento ao longo do tempo. Conforme mencionado por McConnell em seu livro "*Code Complete: A Practical Handbook of Software Construction*", a manutenção envolve atividades como correção de defeitos, ajustes de desempenho e atualizações de segurança. A

evolução, por sua vez, abrange a implementação de novas funcionalidades e melhorias no sistema.

Uma abordagem comum para a manutenção e evolução de um software é utilizar práticas de desenvolvimento ágil. Conforme afirmado por Cohn em seu livro "*Succeeding with Agile: Software Development Using Scrum*", o uso de metodologias ágeis permite uma abordagem iterativa e colaborativa. Conforme mencionado por Gamma et al. em seu livro "*Design Patterns: Elements of Reusable Object-Oriented Software*", a utilização de testes automatizados, como os testes unitários e de integração, ajuda a garantir que as modificações realizadas não introduzam regressões ou quebras de funcionalidade.

Além disso, é importante manter uma documentação atualizada do *software*. Conforme afirmado por Fowler em seu livro "*Refactoring: Improving the Design of Existing Code*", a documentação auxilia os desenvolvedores a entender a estrutura do sistema, facilitando a manutenção e a evolução. Isso inclui documentar as dependências, os componentes-chave, as APIs utilizadas e as decisões de *design* tomadas.

5 RESULTADOS E DISCUSSÕES

No presente Trabalho de Conclusão de Curso (TCC), foi realizado um estudo detalhado para a criação de um software de e-commerce voltado para uma papelaria localizada em São Luís, MA. O projeto envolveu conceitos fundamentais de engenharia de software, desde a definição dos requisitos até a implementação do sistema.

Para garantir uma experiência de usuário (UI) agradável e intuitiva, foram aplicados princípios de design de interface e interação, resultando em uma interface limpa, organizada e de fácil utilização. Além disso, a importância da experiência do usuário (UX) foi enfatizada durante todo o desenvolvimento, visando proporcionar aos usuários uma jornada fluida e satisfatória ao realizar suas compras.

A acessibilidade também foi considerada uma prioridade durante o processo de desenvolvimento do software de e-commerce. Foram adotadas práticas de design inclusivo, como a garantia de contraste adequado, utilização de descrições alternativas para imagens e implementação de recursos de acessibilidade para pessoas com deficiência visual ou auditiva. Isso assegurou que o sistema pudesse ser utilizado por um público diverso, cumprindo as diretrizes de acessibilidade.

Outro aspecto abordado foi a implementação de um design system consistente e modular. Um design system fornece diretrizes e componentes reutilizáveis para a criação de uma interface coesa e harmoniosa em todo o software. Isso promove a consistência visual e a eficiência no desenvolvimento, permitindo que a equipe trabalhe de maneira mais colaborativa e rápida.

Por fim, a importância dos testes de software foi destacada. Foram aplicadas diferentes técnicas de testes, como testes de unidade, integração e aceitação, para garantir a qualidade e a confiabilidade do software de e-commerce. Esses testes ajudaram a identificar e corrigir erros em diferentes etapas do desenvolvimento, contribuindo para a entrega de um produto final robusto e livre de falhas.

Além das etapas mencionadas, também foram considerados aspectos relacionados à segurança e proteção de dados dos usuários. Foram implementadas medidas de segurança, como criptografia de dados, para garantir a confidencialidade e integridade das informações pessoais dos clientes. Dessa forma,

o software de e-commerce desenvolvido para a papelaria em São Luís, MA, busca proporcionar um ambiente seguro e confiável para os usuários realizarem suas transações.

Em resumo, este TCC abordou de forma abrangente a criação de um software de e-commerce para a papelaria. Ao aplicar conceitos de engenharia de software, atenção à UI, UX, acessibilidade, design system e testes de software, o projeto foi capaz de desenvolver um sistema eficiente, intuitivo e inclusivo, proporcionando uma experiência de compra satisfatória para os usuários.

6 CONSIDERAÇÕES FINAIS

Dentre as conclusões que podemos extrair deste estudo e da prototipação nele realizada, destacam-se, em ordem cronológica, alguns eventos que foram fundamentais para alcançarmos o nível atual. Inicialmente, é importante ressaltar que a "guerra dos navegadores" desempenhou um papel crucial ao colocar o controle do design web nas mãos dos desenvolvedores, ou seja, na linha de seus códigos.

Além disso, o amadurecimento de três linguagens de programação desempenhou um papel central no desenvolvimento *web* e *mobile*. *HTML*, *CSS* (ambas de paradigma declarativo) e *JavaScript* (paradigma imperativo) representam, em conjunto, um marco importante. O amadurecimento dessas linguagens permitiu a segmentação do desenvolvimento, resultando em um aumento na qualidade das técnicas de UI e UX.

Vale ressaltar que um terceiro elemento crucial nesta progressão merece atenção: o aprimoramento dessas técnicas de UI e UX levou ao desenvolvimento de novas ferramentas para atender às crescentes demandas. Essas ferramentas incluem não apenas softwares de design, como o já mencionado e consagrado Figma, mas também frameworks de CSS, como o Tailwind CSS, e frameworks JavaScript, como React JS, React Native, Vue JS e Angular / Angular JS.

Com a abordagem de componentização, que foi aprimorada e melhor proporcionada através destes frameworks, os desenvolvedores puderam dividir a interface do usuário em partes menores e independentes, chamadas de componentes. Cada componente encapsula uma funcionalidade específica e pode ser facilmente reutilizado em diferentes partes do aplicativo ou em diferentes projetos. Isso proporcionou maior eficiência e escalabilidade no desenvolvimento, uma vez que os componentes podem ser desenvolvidos, testados e mantidos separadamente.

A componentização, como podemos ver, também abriu caminho para a aplicação de um design system consistente, garantindo, portanto, a consistência visual e de interação em um produto ou empresa. Com os componentes bem definidos e padronizados, foi possível criar um catálogo de elementos visuais e de interação que poderiam ser usados em toda a aplicação.

Esse conjunto de novas funcionalidades, proporcionadas por todos esses elos da corrente de desenvolvimento web e mobile, reduziu a necessidade de recriar elementos de design repetidamente, economizando tempo e esforço. Além disso, a componentização permitiu maior flexibilidade na personalização e extensibilidade do design system, uma vez que os componentes podem ser combinados e ajustados conforme necessário.

Por último, mas não menos importante, a partir de todas essas mudanças, foi possível atender a todas as demandas que surgiram quanto ao desenvolvimento web e mobile. Isso permitiu o aprimoramento das técnicas dos desenvolvedores em um processo que se retroalimentou de forma orgânica e virtuosa, com o objetivo principal de proporcionar uma melhor experiência aos utilizadores de uma determinada plataforma.

É na prototipação que colhemos os primeiros frutos desta evolução, pois nos é permitido conceber, desde as fases iniciais, conceitos de componentização e design system. Por exemplo, podemos criar uma variante de botão "comprar" com destaque para o usuário (CTA) e com a possibilidade de reutilizá-lo conforme a necessidade em diversos momentos, sem a necessidade de redesenhá-lo no caso da prototipação ou recodificá-lo na implementação do código.

Além disso, um mesmo botão pode assumir diferentes formas, tanto na componentização do protótipo quanto em sua codificação durante a implementação, podendo ser o mencionado botão de CTA, bem como um botão de atenção secundária, como é o caso do botão "OK" na busca pelo CEP do protótipo desenvolvido.

Portanto, apesar de os louros dessas melhorias terem início logo na prototipação, é notório que tais benefícios percorrem toda a cadeia de desenvolvimento, otimizando toda e qualquer implementação de boas práticas de UI e/ou UX.

REFERÊNCIAS

ALBERTIN, A. L. **Comercio eletrônico:** modelo, aspectos e contribuições de sua aplicação. 5. ed. São Paulo, Atlas, 2004.

ANDROID DEVELOPERS. **Android Developers**. Disponível em: <https://developer.android.com/>. Acesso em 20 de mar. de 2023.

APPLE DEVELOPER. **Apple Developer**. Disponível em: <https://developer.apple.com/>. Acesso em 20 de mar. de 2023.

ABRAMOV, D., LAKATOS, A. **React in Action.** Manning Publications, 2019.

ALMEIDA, R. **Introdução ao Typescript.** 2021. Disponível em: https://renatoalmeida-80270.medium.com/introdu%C3%A7%C3%A3o-ao-typescript-15a95d6eb8c6. Acesso em: 10 abr. 2023.

AMUNDSEN, S. **RESTful Web Clients.** O'Reilly Media, 2017.

AUDY, J. N, & BRODBECK, A. F. **Sistemas de Informação:** Planejamento e Alinhamento Estratégico nas Organizações. Porto Alegre: Bookman, 2003.

AXIOS. **Axios -** Documentação. Disponível em: https://axios-http.com/docs/intro. Acesso em: 13 mai. 2023.

BIAZUS, M., et al. **Design de Interação e Interfaces para Dispositivos Móveis.** Casa do Código, 2017.

BANCO DO BRASIL. **Como o e-commerce alavancou em meio à pandemia.** Blog BB. 2023. Disponível em: https://blog.bb.com.br/ecommerce-na-pandemia/. Acesso em: 28 abr. 2023.

BARONE, J. S.; BONFANTE, L. C.; SCHNEIDER, M. D. **Implantação do Ecommerce em uma empresa de implementos agrícolas**. 2012. Disponível em: https://periodicos.unesc.net/admcomex/article/download/5241/4740. Acesso em: 13 mai. 2023.

BARSOCCHI, P. GIROLAMI, M. **Detecting Proximity with Bluetooth Low Energy Beacons for Cultural Heritage**, 2021. Disponível em:<https://www.researchgate.net/publication/355670705_Detecting_Proximity_with_Bluetooth_Low_Energy_Beacons_for_Cultural_Heritage>. Acesso em: 05 de jun. de 2023.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 3 ed. Boston: Addison-Wesley Professional, 2013.

BECK, K. **Test-Driven Development:** By Example. Addison-Wesley, 2002.

BUKHARI, M., et al. . **Comparing Prototyping Tools for Mobile Application Design.** International Journal of Computer Science and Information Security, 18(8), 111-1, 2020.

BULEY, L. **The User Experience Team of One.** Rosenfeld Media, 2013.

CANELO, C**. React Native with TypeScript:** The Ultimate Guide. Medium, 2021. Disponível em: https://cristiancanelo.medium.com/react-native-with-typescript-the-ultimate-guide-2021-3a6e2c2f8a0c. Acesso em: 10 abr. 2023.

CHACON, E., STRAUB, B. **Pro Git.** Apress, 2014.

COOPER, A., et al. **About Face:** The Essentials of Interaction Design. John Wiley & Sons, 2014.

COOPER, A. **The Inmates Are Running the Asylum**. Sams, 2004.

CRUMLISH, R., MALONE, C. **Designing Social Interfaces.** O'Reilly Media, 2009.

DART. **About Dart**. 2023. Disponível em: <https://dart.dev/>. Acesso em: 23 mar. 2023.

DAVIS, S. **React Native + Typescript:** A Love Story. Medium. 2020. Disponível em:<https://medium.com/@sdavis2702/react-native-Typescript-a-love-story-b6af28b6f5fa>. Acesso em: 20 de Mar. de 2023.

DIAS, R. **Por que os pequenos negócios são tão importantes para a economia brasileira?** 2021. Disponível em: <https://revistapegn.globo.com/Dia-a-dia/noticia/2021/03/por-que-os-pequenos-negocios-sao-tao-importantes-para-economia-brasileira.html.> Acesso em: 24 abr. 2023.

EISENMAN, B. **Learning React Native.** O'Reilly Media, 2016.

FACEBOOK. (2022). **React Native**. Disponível em: <https://reactnative.dev/>. />. Acesso em: 23 mar. 2023.

FARLEY, M. **React Native for iOS Development.** Packt Publishing, 2015.

FERRARI, L. **A importância dos pequenos negócios para a economia brasileira.** 2020. Disponível em: https://grupo-gestao.com/a-importancia-dos-pequenos-negocios-para-a-economia-brasileira/. Acesso em: 24 abr. 2023.

FERREIRA, R. **Como utilizar o TailwindCSS CSS com React Native**. 2021. Disponível em: https://blog.geekhunter.com.br/como-utilizar-o-TailwindCSS-css-com-react-native/. Acesso em: 10 abr. 2023.

FERREIRA, R. **Desenvolvimento com React:** O que são bibliotecas e como utilizá-las. Medium, 24 out. 2019. Disponível em: https://medium.com/reactbrasil/desenvolvimento-com-react-o-que-s%C3%A3o-bibliotecas-e-como-utiliz%C3%A1-las-29c5b6cbf0cc. Acesso em: 13 mai. 2023.

FLANAGAN, D. **Javascript, O Guia Definitivo:** A Linguagem Mais Usada em Aplicações Web. 6ª edição. Bookman Editora, 2012.

FLING, B**. Mobile design and development:** practical concepts and techniques for creating mobile sites and web apps. Sebastopol: O'Reilly Media, 2016.

FLUTTER. **What is Flutter?** 2023. Disponível em: <https://flutter.dev/>. Acesso em: 23 mar. 2023.

GARCIA, N. **React Native Tanstack Query:** Gerenciando dados com facilidade. 2022. Disponível em: https://dev.to/nickgarciadev/react-native-tanstack-query-gerenciando-dados-com-facilidade-1bhe. Acesso em: 13 mai. 2023.

HELMY, M., & ABDELRAZEK, B. **A Comparative Study of React Native**. In 2018 11th International Conference on Developments in eSystems Engineering (DeSE).

JOTAI. **Jotai:** Documentation. Disponível em: https://github.com/pmndrs/jotai. Acesso em: 13 mai. 2023.

KOENIG, D. **React Helmet:** Gerenciamento de Metadados em Aplicações React. 2018. Disponível em: https://dev.to/danieldavidk/react-helmet-gerenciamento-de-metadados-em-aplicacoes-react-9lj. Acesso em: 13 mai. 2023.

KRUG, S. **Don't Make Me Think:** A Common Sense Approach to Web Usability. New Riders, 2014.

KRUG, S. **Rocket Surgery Made Easy**. New Riders, 2009.

LAUREANO, M.; TONINI, A.; FERREIRA, R. **Engenharia de Requisitos de Software**. São Paulo: Novatec Editora, 2019.

LOTT, S. **Modern Python Cookbook.** Packt Publishing, 2016.

MALAQUIAS, D., SILVA, G., ANDRADE, A., & SOUZA, D. (2021). **Progressive web apps:** um estudo de caso sobre as vantagens e desvantagens dessa tecnologia. Revista de Informática Aplicada.

MANTILLA, F. **Why You Should Be Using Jotai In Your React Native Projects.** Disponível em: https://betterprogramming.pub/why-you-should-be-using-jotai-in-your-react-native-projects-466e7c08ab1c. Acesso em: 13 mai. 2023.

MARTIN, R. C. **Clean Code:** A Handbook of Agile Software Craftsmanship. Prentice Hall, 2009.

MENDES, F.; COLOMBO, F. **Desenvolvimento de aplicativos móveis:** vantagens e desvantagens do desenvolvimento nativo e híbrido. In: ANAIS DO SIMPÓSIO

BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB. SBSMW, 2019, Salvador. Anais [...]. Salvador: SBSMW, 2019.

MILLER, K. S. **React Native in Action.** Manning Publications, 2017.

MORVILLE, P., ROSENFELD, L. **Information Architecture for the World Wide Web.** O'Reilly Media, 2016.

NASCIMENTO, Y. **React Native:** Tudo o que você precisa saber. São Paulo: Novatec Editora, 2020.

NIELSEN, J., MOLICH, R. **Heuristic Evaluation of User Interfaces.** Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 249-256, 1990.

NIELSEN, J. **Usability Engineering.** Academic Press, 1993.

NORMAN, D. T**he Design of Everyday Things.** Basic Books, 2013.

OLIVEIRA, G. A.; FERREIRA, E. V. **React Native:** desenvolvimento mobile com Javascript. Revista Programar, v. 2, n. 5, 2018.

OPENSIGNAL. **Fragmentação Android:** um problema para desenvolvedores de aplicativos. 2016. Disponível em: https://www.opensignal.com/reports/2016/08/android-fragmentation/. Acesso em: 25 mar. 2023.

PATEL, M., & PATEL, H. **React Native:** A Revolutionary Cross-Platform Mobile Development Technology. International Journal of Computer Sciences and Engineering, 2018.

PRESSMAN, R. S. **Engenharia de software:** uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

QUILES, R.; MARQUES, R.; MARTINS, I. et al. **Desenvolvimento nativo vs. desenvolvimento multiplataforma**: análise comparativa e estudo de caso. Revista de Sistemas e Computação (V8, 2018).

RAHMAN, M. **React Native in Action**. Boston: Manning Publications, 2022.

REACT NAVIGATION. **React Navigation:** Documentation. Disponível em: https://reactnavigation.org/docs/getting-started. Acesso em: 13 mai. 2023.

ROCHA, L. A. F., & OLIVEIRA, R. D. (2020). **O uso do React Native no desenvolvimento de aplicativos móveis:** vantagens e desvantagens. In Anais do Congresso Brasileiro de Informática na Educação (CBIE).

RODRIGUES, J. **Desenvolvimento de Aplicativos Cross-platform com React Native.** In: Anais da Conferência de Sistemas de Informação. São Paulo, 2021.

ROSSMANN, B. What Are Progressive Web Apps (PWAs): And Why Are They Revolutionizing Mobile? **Forbes**, 2020. Disponível em: <https://www.forbes.com/sites/forbestechcouncil/2020/01/07/what-are-progressive-web-apps-pwas-and-why-are-they-revolutionizing-mobile/?sh=256d194c2216>. Acesso em: 20 mar. 2023.

SANTOS, A. Brasil, segudo país onde o mercado de aplicativos mais cresce. **Terra**, 2020. Disponível em:<https://www.terra.com.br/noticias/dino/brasil-segundo-pais-onde-o-mercado-de-aplicativos-mais-cresce,1fd9d38aa995ad8ca1243f6c58080f79u2ee8tfj.html.>. Acesso em: 15 abr. de 2023.

SANTOS, I. de O. **E-commerce:** a revolução do comércio eletrônico. São Paulo: Saraiva, 2013.

SANTOS, L. **Designing an Effective Architecture for E-commerce Platforms.** In: Anais do Congresso Internacional de Sistemas de Informação. Rio de Janeiro, 2023.

SANTOS, M. A. **Desenvolvimento de aplicativos móveis com React Native**. São Paulo: Casa do Código, 2019.

SHAMIEH, C. **Engenharia de Sistemas para leigos**. Hoboken: Wiley Publishing, Inc, 2011.

SNYDER, C. **Paper Prototyping:** The Fast and Easy Way to Design and Refine User Interfaces. Morgan Kaufmann, 2003.

SILVA, A. C.; ALMEIDA, A. V. ***A importância dos dispositivos móveis conectados à internet para a sociedade contemporânea*.** In: *CONGRESSO BRASILEIRO DE CIÊNCIAS DA COMUNICAÇÃO*, 42., 2019, Belo Horizonte. Anais... São Paulo: INTERCOM, 2019. p. 21-25.

SILVA, J. **React Native:** Desenvolvimento de aplicativos móveis com Javascript e React. São Paulo: Casa do Código, 2021.

SILVA, J. P.; FERNANDES, L. S. **Engenharia de requisitos de software.** São Paulo: Érica, 2019.

SINGH, N. **Mobile App Development: Native vs Hybrid vs Web App**. 2017. Disponível em: https://www.altexsoft.com/blog/mobile/native-vs-hybrid-vs-web-apps-which-is-better-for-your-app/. Acesso em: 24 mar. 2023.

SKOVYRA, J. **React Native:** pros and cons of the framework. 2019. Disponível em:<https://www.netguru.com/blog/react-native-pros-and-cons-of-the-framework>. Acesso em: 03 de abr. de 2023.

SOMMERVILLE, I. **Engenharia de software**. 9. ed. São Paulo: Pearson, 2011.

SOUZA, Rafael. **Introdução à Metodologia Ágil:** Scrum. São Paulo: Casa do Código, 2020.

STANGER, I. et al. **React Native:** Um Estudo de Caso Comparativo com o Desenvolvimento Nativo Android. In: Anais do XVII Simpósio Brasileiro de Qualidade de Software, 2018, Fortaleza.

TANSTACK. **Tanstack Query:** Documentation. Disponível em: https://tanstack.com/query. Acesso em: 13 mai. 2023.

TAVARES, M. **Aplicativos Móveis:** Como Planejar, Desenvolver e Distribuir seu App. São Paulo: Novatec Editora, 2014.

TIDWELL, J. **Designing Interfaces:** Patterns for Effective Interaction Design. Sebastopol: O'Reilly Media, 2010.

TREDER, M. **Designing Interfaces:** Principles of UI Design. Smashing Magazine, 2011.

TUFTE, E. **The Visual Display of Quantitative Information.** 1 Ed. Graphics Press, 2001.

TYPESCRIPT. **Documentação**. Disponível em: https://www.typescriptlang.org/docs/. Acesso em: 10 abr. 2023.

VARELLA, F. **Desenvolvimento Híbrido para Dispositivos Móveis**: Tecnologias e Técnicas. São Paulo: Novatec Editora, 2018.

VEERAPPAN, C. N., et al. **Evaluating UI/UX Design Tools for Mobile Applications:** A Comparative Study. International Journal of Engineering and Advanced Technology, 10(2), 3384-3391, 2021.

W3C. Web Content Accessibility Guidelines (WCAG) 2.1. Recuperado de: https://www.w3.org/TR/WCAG21/

ZELDMAN, S. J. **Designing with Web Standards.** New Riders, 2017.

ZHANG, Y., & ZHAO, X. **Progressive Web App:** The Future of Mobile Web. In 2019 IEEE International Conference on Computational Science and Engineering (CSE), 2019.

ANEXO A - TERMO DE AUTORIZAÇÃO DE REALIZAÇÃO DE PESQUISA

PAPELARIA PAPEL MACHÊ

AUTORIZAÇÃO DE REALIZAÇÃO DE PESQUISA

Declaramos para os devidos fins, que cederemos ao/à pesquisador/a **Diogo Mondego dos Santos**, o acesso aos arquivos de **base de dados de pesquisa** para serem utilizados na pesquisa: ANÁLISE DE VIABILIDADE DE UMA APLICAÇÃO MOBILE: estudo de caso dos processos correlatos à elaboração de um software e-commerce para uma papelaria em São Luís, MA, cujo objetivo é analisar o processo de prototipação e desenvolvimento e uma solução mobile de um e-commerce de uma papelaria, sendo esta ~~hipoteticamente~~ situada em São Luís - MA, verificando os requisitos do projeto e visando a eficácia da solução desenvolvida, que está sob a orientação do/a Prof/a. Esp. Daniel Herrera de Oliveira Lemos.

Esta autorização está condicionada ao cumprimento do (a) pesquisador (a) aos requisitos das Resoluções do Conselho Nacional de Saúde e suas complementares, comprometendo-se o(a) mesmo(a) a utilizar os dados pessoais dos participantes da pesquisa, exclusivamente para os fins científicos, mantendo o sigilo e garantindo a não utilização das informações em prejuízo das pessoas e/ou das comunidades.

MB - Mondego / 05926715000 1-93

Nome/assinatura e carimbo do responsável pela
Instituição ou pessoa por ele delegada