WELLINGTON RODRIGUES PEREIRA

DESENVOLVIMENTO DE APLICAÇÃO MÓVEL PARA VENDEDORES EXTERNOS DE UMA EMPRESA ATACADISTA DO SETOR DE PESCA E NÁUTICA

LAVRAS – MG

2021

WELLINGTON RODRIGUES PEREIRA

DESENVOLVIMENTO DE APLICAÇÃO MÓVEL PARA VENDEDORES EXTERNOS DE UMA EMPRESA ATACADISTA DO SETOR DE PESCA E NÁUTICA

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do Programa de Graduação em Sistemas de Informação, para a obtenção do título de Bacharel.

Prof. DSc. Antônio Maria Pereira de Resende

Orientador

LAVRAS – MG

2021

WELLINGTON RODRIGUES PEREIRA

DESENVOLVIMENTO DE APLICAÇÃO MÓVEL PARA VENDEDORES EXTERNOS DE UMA EMPRESA ATACADISTA DO SETOR DE PESCA E NÁUTICA

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do Programa de Graduação em Sistemas de Informação, para a obtenção do título de Bacharel.

APROVADA em 20 de maio de 2021.

Prof. DSc. Heitor Augustus Xavier Costa UFLA
Prof. DSc. Paulo Afonso Parreira Junior UFLA

Prof. DSc. Antônio Maria Pereira de Resende
Orientador

LAVRAS – MG
2021

Pela presença e apoio em todos os momentos, dedico este trabalho aos meus pais.

AGRADECIMENTOS

À Universidade Federal de Lavras, seus professores e corpo técnico por propiciarem um ambiente de muitas oportunidades e crescimento. Ao professor e orientador Antônio Maria Pereira de Resende, por se mostrar sempre disposto a contribuir.

No futuro, os computadores podem pesar não mais que 1.5 toneladas.
(Popular Mechanics, 1949)

RESUMO

A HP Pesca, Moto e Náutica é uma empresa atacadista do setor de pesca e náutica, sediada na cidade de Varginha, MG, e com filial na cidade de São Paulo, SP. Uma parte considerável das vendas da empresa são provenientes de vendedores externos e representantes, que prospectam clientes e realizam vendas com o apoio de um catálogo impresso de produtos. Devido ao fato da empresa possuir um catálogo muito extenso, há constantes alterações de preços e disponibilidade dos produtos, o que torna os catálogos impressos constantemente defasados. Sob este contexto, buscou-se nesse trabalho encontrar uma solução que permitisse ao vendedor externo e ao representante uma forma de consultar as informações dos produtos de forma rápida e fácil, estando essas informações atualizadas e consistentes. Optou-se pela construção de um aplicativo móvel que, se alimentando de uma API por meio da internet, permita a consulta de produtos e o cadastro de pedidos de forma digital. O aplicativo e a API foram modelados seguindo o padrão de notação UML e desenvolvidos com as ferramentas PHP e React Native, principalmente. A utilização do aplicativo, inicialmente por um número limitado de vendedores, apresentou bons resultados, diminuindo drasticamente a ocorrência de problemas relacionados a defasagem de informações.

Palavras-chave: Modelagem de Software; UML; Desenvolvimento de Sistemas; Desenvolvimento Móvel. Teste de Software.

ABSTRACT

HP Pesca, Moto e Náutica is a wholesale company in the fishing and nautical sector based in Varginha, MG, with a second branch in Sao Paulo, SP. A considerable part of the company's sales come from external salesmen and representatives, who prospect customers and make sales with the support of a printed product catalogue. Due to the fact that the company has a very extensive catalogue, there are constant changes in prices and availability of products, which makes printed catalogs constantly outdated. Therefore, the present work sought to find a solution that would allow external salesmen and representatives to look up products' up-to-date and consistent information, quickly and easily. The chosen solution was to build a mobile application that, feeding on an API through the internet, allows the search for products information and the placement of orders digitally. The application and API were modeled following UML notation and developed with PHP and React Native, mainly. The use of the app, initially by a limited number of salesmen, brought good results, drastically reducing the occurrence of issues related to information inconsistency.

Keywords: Software Design; UML; Systems Development; Mobile Development; Software Testing.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este documento descreve a modelagem e desenvolvimento de um sistema construído sob a responsabilidade do autor deste trabalho para atender a necessidade da empresa HP Pesca, Moto e Náutica Yamaha, localizada em Varginha, Minas Gerais. O autor é funcionário da empresa desde 2017 onde atua na parte de e-commerce, tecnologia da informação e desenvolvimento de sistemas.

A motivação para o desenvolvimento deste sistema emergiu, principalmente, devido a três fatores: (1) a grande quantidade de vendedores externos, (2) um extenso catálogo com mais de 2000 produtos e (3) oscilação constante de preços e disponibilidade destes produtos.

Devido a estes fatores, somado ao fato de que até o momento a empresa trabalhava com tabelas e catálogos impressos, boa parte dos vendedores estavam frequentemente com informações defasadas, tanto de preços, características e até mesmo disponibilidade dos produtos oferecidos. Surgiu, assim, a necessidade de centralizar e acessar de forma fácil e rápida essas informações, de forma digital.

Inicialmente foi discutido o formato da solução, se seria uma aplicação executada no computador do vendedor, uma aplicação web acessada por meio de um navegador, ou uma aplicação móvel instalada no *smartphone* do vendedor. Optou-se pela última devido ao fato de que, comumente, vendedores e representantes frequentam feiras de exposição de produtos, onde possuem a necessidade de locomoção constante. Soma-se também a essa escolha a constatação de que todos os vendedores externos e representantes da empresa já possuem um *smartphone*.

Após determinado o formato da solução, buscou-se no mercado aplicações que pudessem suprir satisfatoriamente as expectativas. As soluções avaliadas, contudo, se mostraram mais complexas que o esperado, o que não se julgou apropriado para um cenário em que o público alvo da aplicação possui níveis muito diferentes de habilidade com tecnologia. Concluiu-se, portanto, que a melhor abordagem seria o desenvolvimento de uma aplicação própria, totalmente alinhada com o negócio e que fornecesse tão somente as funções necessárias.

Neste contexto, identificou-se a necessidade de desenvolver um aplicativo móvel que, se alimentando das informações concedidas por uma API (*Application Programming Interface*), permitisse ao vendedor externo consultar instantaneamente o catálogo de produtos da empresa e ter acesso às informações necessárias, incluindo preços, disponibilidade, condições de pagamento e características técnicas, além de imagens dos produtos. Além disso, o sistema deveria propiciar ao vendedor a capacidade de realizar o cadastro de pedidos

que, após finalizados, pudessem ser acessados na sede da empresa, a qual iniciaria os processos de cobrança, fiscal e almoxarifado

Assim, o objetivo deste trabalho foi desenvolver um aplicativo móvel e uma API que permita o vendedor consultar o catálogo de produtos e realizar pedidos.

Este trabalho está organizado da forma que se segue. No capítulo 2, é apresentado o referencial teórico, contemplando os tópicos de modelagem, UML (*Unified Modeling Language*), diagrama de casos de uso, diagrama de classe, diagrama de sequência, diagrama de pacotes, diagrama de implantação, MER (Modelo Entidade Relacionamento) e teste de software, respectivamente. Em seguida, no capítulo 3, é apresentado o processo de desenvolvimento empregado neste trabalho. No capítulo 4, o desenvolvimento do trabalho é detalhado. Os resultados alcançados são descritos no capítulo 5. Por fim, o capítulo 6 contém as conclusões deste trabalho, seguido das referências bibliográficas.

2 REFERENCIAL TEÓRICO

A modelagem de sistemas de software é parte essencial do desenvolvimento de uma aplicação. Nela, se criam modelos que são analisados antes da implementação do sistema. Estes modelos direcionam a posterior implementação (GOMAA, 2011).

Nas palavras de (BOOCH; RUMBAUGH; JACOBSON, 2006), "projetos de software mal sucedidos falham em relação a aspectos únicos e específicos de cada projeto, mas todos os projetos bem sucedidos são semelhantes em diversos aspectos" e, para os autores, a modelagem de software é um destes aspectos em comum dos projetos bem sucedidos.

Ainda de acordo com (BOOCH; RUMBAUGH; JACOBSON, 2006), são quatro os objetivos alcançados pelo emprego da modelagem de sistemas:

1. Os modelos ajudam a visualizar o sistema como ele é ou como desejamos que ele seja;
2. Os modelos permitem especificar a estrutura e/ou o comportamento de um sistema;
3. Os modelos proporcionam um guia para a construção do sistema;
4. Os modelos documentam as decisões tomadas.

Na Engenharia de Software, uma das formas de se fazer a modelagem é por meio do padrão de notação UML, criado na década de 1990 para padronizar as diferentes notações existentes na época. Da mesma forma que um resistor tem o mesmo símbolo numa planta de circuito eletrônico no Japão, EUA, China, Brasil etc, foi necessário padronizar mundialmente a notação utilizada na construção de plantas de software.

2.1 UML

A UML (*Unified Modeling Language*) pode ser definida como uma família de notações gráficas que auxilia na descrição e no projeto de sistemas de software (FOWLER; KOBRYN; ONLINE, 2004). Como explicam (BOOCH; RUMBAUGH; JACOBSON, 2006), essa linguagem é utilizada para as seguintes ações no desenvolvimento de software: a) visualizar; b) especificar; c) construir; e d) documentar um sistema de software, detalhadas a seguir.

Quanto à ação de visualizar, o emprego da UML oferece grande apoio para a compreensão de um sistema complexo. Não só para para aqueles diretamente envolvidos no seu desenvolvimento, mas também

para outros que eventualmente tenham a necessidade da compreensão do sistema, ou de parte dele. Para (BOOCH; RUMBAUGH; JACOBSON, 2006), isso se deve ao fato da UML possuir uma semântica bem definida, o que suprime a ocorrência de ambiguidades.

Quanto à ação de especificar, a UML permite elaborar entendimento mais alinhado entre os interessados, eliminando contratempos que ocorreriam por entendimentos errôneos do escopo do projeto (RIBEIRO, 2013). Nas palavras de (BOOCH; RUMBAUGH; JACOBSON, 2006), "especificar significa construir modelos precisos, sem ambiguidades e completos". São esses modelos que servirão como base para a implementação do sistema.

Quanto à ação de construir, os modelos desenvolvidos durante a fase de especificação servem como guia para que se desenvolvam os elementos da aplicação. Modelos obtidos por meio da UML podem ser facilmente mapeados para uma linguagem de programação, especialmente aquelas sob o paradigma da orientação a objetos (FOWLER; KOBRYN; ONLINE, 2004). Esse mapeamento permite que o código seja produzido com base nos modelos preestabelecidos.

Quanto à ação de documentar, os diagramas elaborados por meio da UML fornecem uma ampla visão do sistema. Neles, representa-se o comportamento do sistema, suas funcionalidades e, indo mais a fundo, os requisitos identificados nas fases preliminares de concepção do software e como foram satisfeitos. Conforme enfatizam (BOOCH; RUMBAUGH; JACOBSON, 2006), a produção de artefatos, além do código-fonte, é de grande importância para que se chegue a um sistema de software saudável.

Em suma, como apresentam (ANDA et al., 2006), a utilização da UML no processo de construção de um sistema de software se mostra eficaz, capaz de promover melhorias no entendimento e no acompanhamento dos requisitos do projeto, na implementação do código e no desenvolvimento de casos de teste, além de melhorar a comunicação e a documentação do sistema.

A versão mais atual da UML é a 2.5.1 e ela possui os seguintes diagramas: a) Diagrama de Classes; b) Diagrama de Componentes; c) Diagrama de Implantação; d) Diagrama de Objetos; e) Diagrama de Pacotes; f) Diagrama de Perfil; g) Diagrama de Estrutura Composta; h) Diagrama de Casos de Uso; i) Diagrama de Atividade; j) Diagrama de Transição de Estados; l) Diagrama de Sequência; m) Diagrama de Comunicação; n) Diagrama de Interação e o) Diagrama de Tempo (ABOUT. . . ,). Nas seções seguintes serão detalhados os diagramas utilizados neste trabalho.

2.2 Diagrama de Casos de Uso

O objetivo do diagrama de caso de uso é definir os casos em que o sistema de software será usado e por qual ator. Além disso, define em alto nível detalhes de cada caso de uso.

Um caso de uso pode ser definido como "um conjunto de sequência de ações, inclusive variantes, que um sistema executa para produzir um resultado" (BOOCH; RUMBAUGH; JACOBSON, 2006). Para cada uma dessas sequências, dá-se o nome de cenário. Conforme explica (LARMAN, 2000), um cenário é uma sequência específica de interações entre o sistema e os atores, e que produz um determinado resultado.

É importante ressaltar que um cenário não necessariamente se refere a casos de sucesso de utilização do sistema (LARMAN, 2000). Também são consideradas cenários as sequências de ações que não levam a conclusão da ação, como por exemplo, mas não somente, nos casos onde atores apresentam dados incompletos ou incorretos.

Outra observação relevante é que os atores dos casos de uso não são necessariamente usuários humanos. (LARMAN, 2000) define um ator como algo com comportamento. De acordo com os autores, isso pode representar uma pessoa, um sistema ou uma organização, por exemplo.

No diagrama UML de casos de uso, cada ator é representado por um desenho simplificado de uma pessoa, e os casos de uso são representados por seus nomes, circulados por uma elipse. As associações são representadas por uma reta ligando atores a casos de uso, e casos de uso a outros casos de uso.

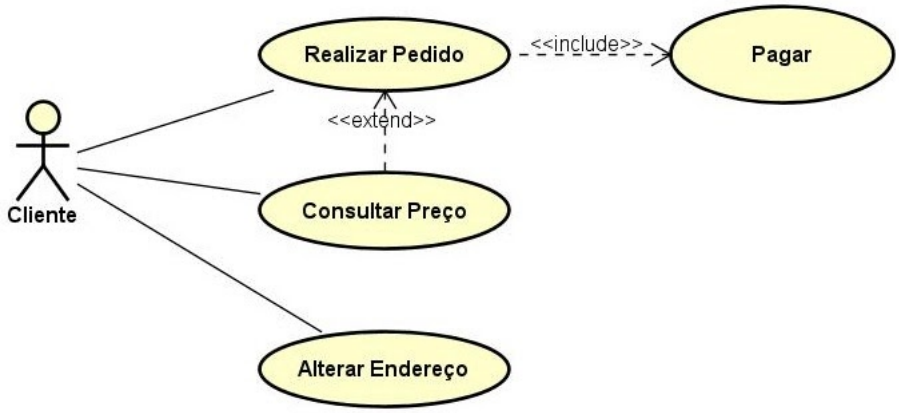
As associações entre os casos de uso podem ser do tipo de inclusão ou extensão. No caso de associação de inclusão, um caso de uso é parte obrigatória do processo de outro. Este tipo de associação é representada no diagrama pela notação «include».No caso da extensão, um caso de uso é parte opcional do processo de outro. Este tipo de associação é representada no diagrama pela notação «extend».

Na figura 2.1, observa-se um exemplo de diagrama de casos de uso.

No exemplo citado, o ator Cliente pode executar os casos de uso Realizar Pedido, Consultar Preço e Alterar Endereço. Para o caso de uso Realizar Pedido, há também a ação obrigatória de Pagar, ou seja, é obrigatório a realização do pagamento para que se possa realizar o pedido. Isso está representado do diagrama pela notação «include».

Além disso, conforme apresenta o diagrama, a ação de Consultar Preço é uma ação opcional da ação Realizar Pedido. Por este motivo, durante o processo de realização do pedido, o cliente pode optar por consultar o preço de algum produto. Esta ação, porém, não é obrigatória. Isso está representado no diagrama pela notação «extend».

Figura 2.1 – Exemplo de Diagrama de Casos de Uso



Fonte: Elaborada pelo autor.

Para cada caso de uso especificado no diagrama, deve-se fazer sua descrição. Esta descrição irá apresentar em mais detalhes os passos necessários para que o sistema possa executar o caso de uso.

A descrição de um caso de uso apresenta uma visão mais abrangente sobre como aquele caso de uso é executado e segue a estrutura exemplificada acima. A descrição se inicia com o nome do caso de uso e de um breve resumo, seguidos da pré-condição e a pós-condição da execução do caso de uso.

A pré-condição define pré-requisitos que precisam estar satisfeitos para que o caso de uso seja executado. No exemplo apresentado, o usuário estar autenticado no sistema é uma pré-condição. A pós-condição define o que ocorre após a execução do caso de uso. No exemplo apresentado, o sistema atualiza a contagem de quantas vezes aquele produto específico teve seu preço consultado.

Após essas definições, a descrição define o fluxo de execução do caso de uso. Inicialmente, o seu fluxo principal, seguido dos fluxos alternativos.

O fluxo principal descreve o fluxo natural de execução do caso de uso. No exemplo citado, o fluxo principal descrito é aquele onde o usuário informa o código correto do produto, o sistema encontra suas informações no banco de dados e apresenta o preço ao cliente, conforme esperado.

Em relação ao fluxo alternativo, não há um consenso na academia. Alguns autores fazem uma distinção entre a) fluxo alternativo, como uma forma diferente de realizar o caso de uso e b) exceção, como um fluxo de execução para o caso de ocorrência de erros, inconsistência de informações ou casos similares. Outros autores, como (SOMÉ, 2009) por exemplo, consideram ambos os casos como fluxo alternativo. Essa última abordagem será empregada neste trabalho.

Há um fluxo alternativo descrito no exemplo citado. Nele descreve-se o fluxo de execução para o caso em que não foi possível localizar o produto a partir do código informado.

A figura 2.2 apresenta o detalhamento do caso de uso Consultar Preço, supracitado.

Figura 2.2 – Exemplo de Descrição de um Caso de Uso

Nome do Caso de Uso:	Consultar Preço	
Resumo:	O cliente realiza a consulta de preço de um produto específico	
Pré-condição:	O cliente precisa se autenticar no sistema	
Pós-condição:	O sistema aumenta a contagem de vezes que o produto foi consultado	
Fluxo Principal:	Cliente: 1. Informa o código do produto desejado;	Sistema: 2. Consulta no banco de dados o preço do produto; 3. Exibe o preço para o cliente.
Fluxo Alternativo 1:	Cliente:	Sistema: 2. Se o produto não foi encontrado, exibe a mensagem "Produto não encontrado".

Fonte: Elaborada pelo autor.

2.3 Diagrama de Classes

Em sistemas orientados a objetos, a classe é um dos elementos mais importantes. (BOOCH; RUMBAUGH; JACOBSON, 2006) definem classe como “uma descrição de um conjunto de objetos que compartilham os mesmo atributos, operações, relacionamentos e semântica. Para os autores, ter classes bem estruturadas é uma forma de distribuir as responsabilidades de forma eficiente e equilibrada em um sistema.

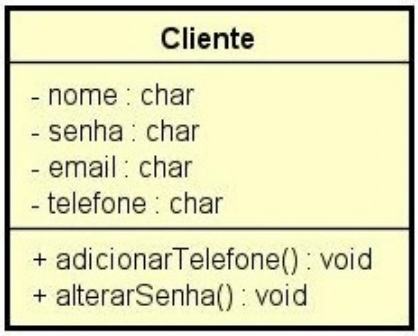
Isso evidencia a importância de desenvolver um bom diagrama de classes antes do desenvolvimento do sistema. Para (GENERO; PIATTINI; CALERO, 2002), a qualidade do diagrama de classes pode ser determinante para a qualidade final do software.

O diagrama de classes é uma representação das classes do software, e seus relacionamentos. Conforme (BOOCH; RUMBAUGH; JACOBSON, 2006), o primeiro passo para sua elaboração é a identificação dos itens importantes do sistema.

No diagrama de classes, cada classe é representada por um retângulo e deve possuir, basicamente, um nome, seus atributos e sua operações. Na figura 2.3 observa-se um exemplo de representação de uma classe.

No exemplo apresentado, a classe de nome Cliente possui os atributos nome, senha, email e telefone, e as operações adicionarTelefone e alterarSenha.

Figura 2.3 – Exemplo de Representação de uma Classe



Fonte: Elaborada pelo autor.

O diagrama de classes, além de apresentar as classes e seus atributos e operações, também aponta os relacionamentos entre elas. De acordo com (BOOCH; RUMBAUGH; JACOBSON, 2006), os relacionamentos dos tipos de associação e generalização estão entre os principais tipos de relacionamentos entre classes.

Nos relacionamentos do tipo de associação, representam-se ligações entre classes. No diagrama UML, este tipo de relacionamento é representado por uma reta ligando as duas classes que se relacionam.

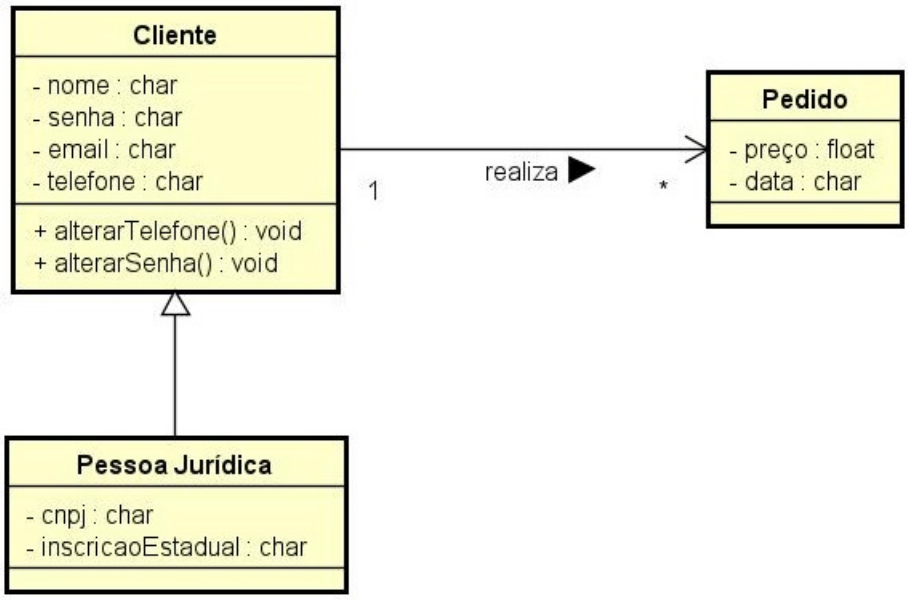
Os relacionamentos de generalização definem os casos de classes gerais e suas especializações. Em outras palavras, um relacionamento de generalização ocorre quando uma classe possui todas as características de outra, mas também elementos próprios. No diagrama de classes, representa-se este tipo de relacionamento com uma linha sólida seguida de uma seta triangular que aponta para a classe geral (também chamada de classe-mãe, ou superclasse).

Na Figura 2.4, observa-se a representação de relacionamentos de generalização e associação em um diagrama de classes.

No exemplo supracitado, a classe Pessoa Jurídica é uma subclasse de Cliente. Dessa forma, a classe Pessoa Jurídica possui todos os atributos e operações da classe Cliente, além de seus atributos próprios: cnpj e inscricaoEstadual.

Há também, no exemplo citado, um relacionamento do tipo de associação entre as classes Cliente e Pedido. Conforme representado no diagrama, o cliente realiza o pedido. Desta forma, as instâncias da classe Pedido estão vinculadas a instâncias da classe Cliente, por meio do atributo cliente presente na classe Pedido.

Figura 2.4 – Exemplo de Representação de Relacionamentos Entre Classes



Fonte: Elaborada pelo autor.

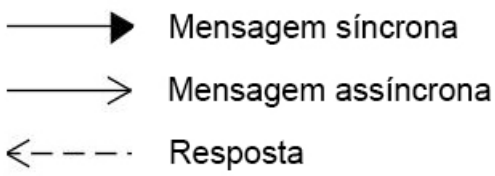
2.4 Diagrama de Sequência

O objetivo do diagrama de sequência é representar as interações entre objetos no sistema de software para cada caso de uso criado. Conforme mostra (BELL, 2004), o diagrama apresenta essas interações de maneira sequencial em um cenário. Por este motivo, esse diagrama é construído a partir do diagrama de casos de uso.

As mensagens trocadas entre os elementos da aplicação são representadas de maneira diferente no diagrama de sequência, dependendo de seu tipo. O envio de uma mensagem é representado por uma linha reta, enquanto que a resposta é representada por uma linha pontilhada. No caso de mensagens enviadas, há ainda a diferenciação entre mensagem síncrona e mensagem assícrona. No caso de uma mensagem síncrona, a aplicação interrompe sua execução e aguarda o recebimento de uma resposta para, então, prosseguir. No caso de uma mensagem assíncrona, a aplicação segue o seu fluxo de execução sem a necessidade de aguardar a resposta para continuar. No diagrama, mensagens síncronas são representadas por uma seta de ponta sólida, enquanto que mensagens assíncronas são representadas por uma seta de ponta aberta. Na Figura 2.5, observam-se as representações mencionadas.

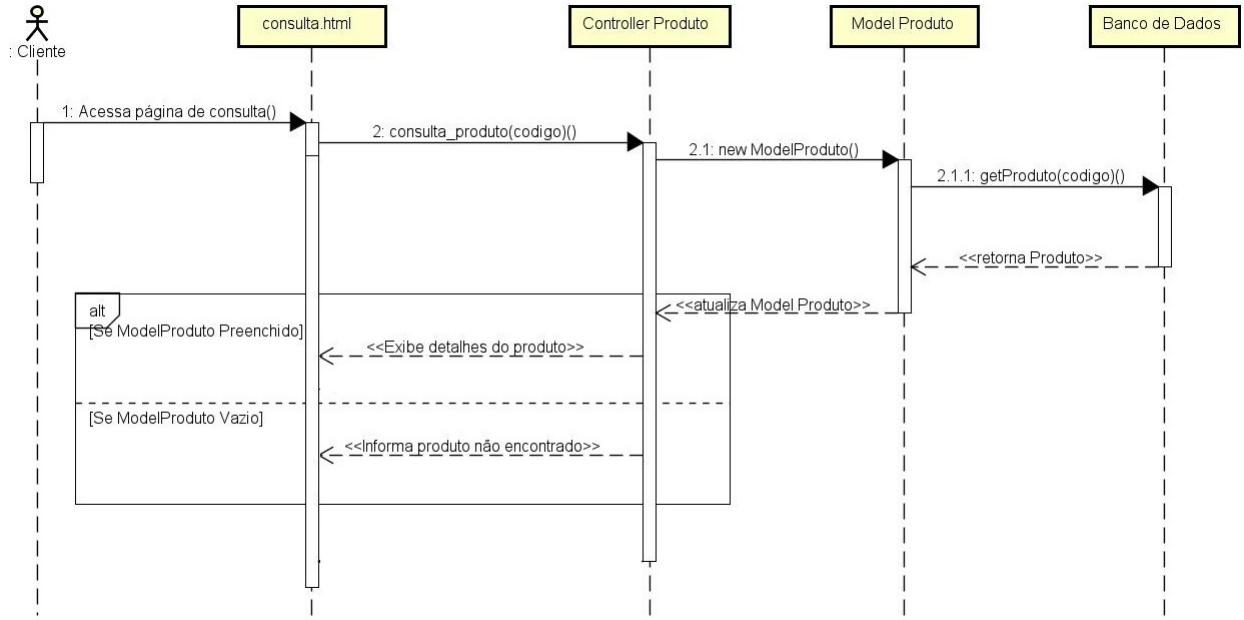
O diagrama de sequência representado na figura 2.6 exemplifica um processo de consulta de produto em um sistema web sob o padrão MVC (Modelo-Visão-Controle).

Figura 2.5 – Representação de Mensagens em um Diagrama de Sequência



Fonte: Elaborada pelo autor.

Figura 2.6 – Exemplo de um Diagrama de Sequência



Fonte: Elaborada pelo autor.

No exemplo citado, está representado um processo de consulta de produto em um sistema web sob o padrão MVC (Modelo, Visão, Controle). O processo se inicia com o ator (cliente) acessando a página de consulta consulta.html. Nesta página, o ator informa o código do produto pretendido e este é enviado ao controlador Produto (Controller Produto) por meio de uma mensagem síncrona. O controlador, por sua vez, cria uma instância de um modelo do produto (Model Produto), que faz o acesso ao banco de dados para buscar as informações do produto com o código informado. Essas comunicações também ocorrem por meio de mensagens síncronas. O modelo do produto é atualizado com as informações obtidas no banco de dados e as retorna ao controlador.

Nesta etapa da execução, conforme representado no diagrama, há uma condição que determina dois diferentes caminhos para finalizar o fluxo desta tarefa.

O primeiro caminho abrange o caso em que o modelo do produto está de fato preenchido com as informações do produto buscado. Neste caso, as informações do produto são exibidas ao usuário.

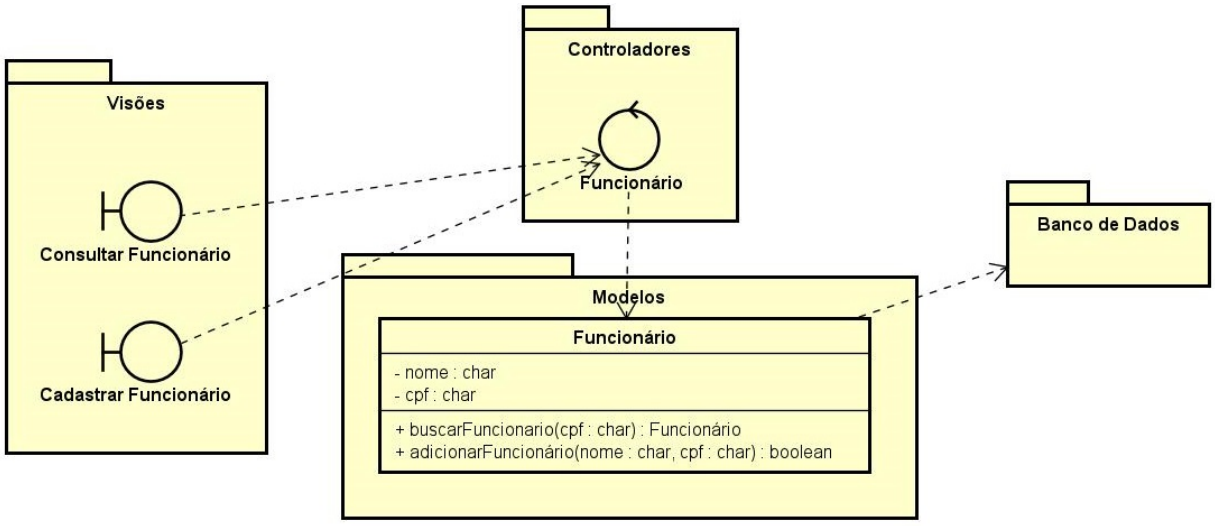
No segundo caminho, está contemplada a possibilidade do modelo do produto estar vazio. Isso indica que não foi possível ter acesso às informações do produto buscado no banco de dados, seja por um erro interno da aplicação, seja por um código incorreto informado pelo usuário. Neste caso, uma mensagem é exibida ao usuário informando que o produto não foi encontrado..

2.5 Diagrama de Pacotes

O diagrama de pacotes apresenta um agrupamento lógico das partes do sistema e seus relacionamentos. O principal benefício de utilizar diagramas de pacotes é propiciar melhor compreensão do sistema de software. Especialmente em grandes sistemas, o diagrama de pacotes facilita o entendimento da estrutura do projeto e de como os diferentes elementos atuam em cada parte do sistema, e como essas partes se relacionam.

A figura 2.7 apresenta um exemplo de um diagrama de pacotes de uma aplicação sob o modelo MVC.

Figura 2.7 – Exemplo de um Diagrama de Pacotes



Fonte: Elaborada pelo autor.

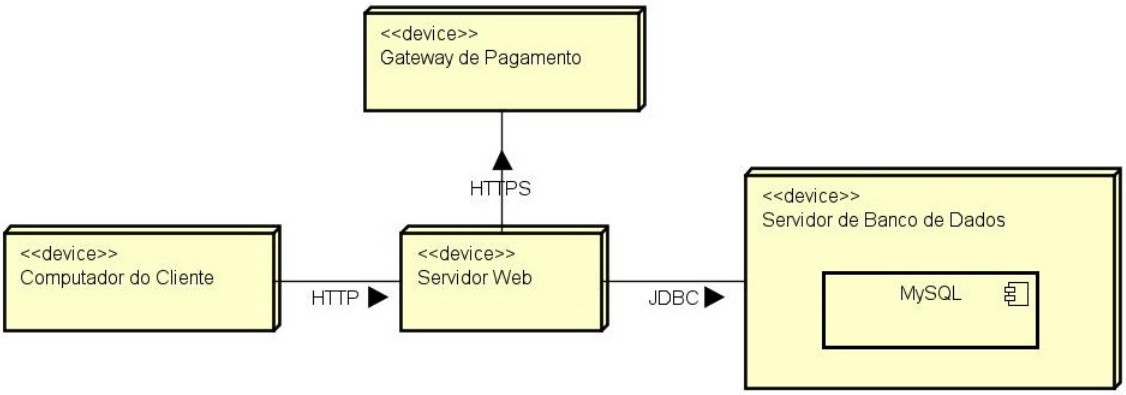
A figura citada representa um simples sistema hipotético de cadastro de funcionários. O sistema exemplificado está estruturado sob o padrão MVC e possui quatro pacotes: Modelos, Visões, Controladores e Banco de Dados.

Além de apresentar os pacotes e seus componentes, o diagrama de pacotes também representa os relacionamentos entre esses pacotes, e/ou seus componentes. No exemplo apresentado, as visões Consultar Funcionário e Cadastrar Funcionário e o modelo Funcionário são instanciados pelo controlador Funcionário. Por este motivo, está representada no diagrama uma relação de dependência entre eles. O mesmo ocorre entre o modelo Funcionário e o Banco de Dados, pois o modelo é responsável por realizar as operações no banco de dados.

2.6 Diagrama de Implantação

O objetivo de um diagrama de implantação é apresentar uma representação física do sistema. Em outras palavras, conforme explica (GUEDES, 2018), o diagrama de implantação descreve os artefatos de hardware necessários para que o sistema seja executado. O autor cita como exemplos servidores, estações, topologias e protocolos de comunicação. A figura 2.8 exemplifica um diagrama de implantação.

Figura 2.8 – Exemplo de um Diagrama de Implantação



Fonte: Elaborada pelo autor.

No exemplo acima, está representada de maneira simplificada a estrutura física de uma hipotética loja virtual. Há quatro nós no diagrama que representam os dispositivos físicos que permitem que o sistema execute: a) o computador do cliente, b) o servidor web, c) o servidor da processadora de pagamentos e d) o servidor de banco de dados.

O diagrama também apresenta as associações entre os nós, além de mostrar como sua comunicação ocorre. No exemplo apresentado, o computador do cliente se comunica com o servidor web através do protocolo HTTP (*Hypertext Transfer Protocol*), e o servidor web se comunica com o gateway de pagamento

e com o servidor de banco de dados por meio dos protocolos HTTPS (*Hypertext Transfer Protocol Secure*) e JDBC (*Java Database Connectivity*), respectivamente.

Além disso, o diagrama de implantação apresenta os eventuais componentes que podem ser utilizados pelos dispositivos para executar o sistema. No exemplo apresentado, o MySQL é um componente do servidor de banco de dados.

2.7 MER

O MER (Modelo Entidade Relacionamento) (CHEN, 1976) é um modelo de dados que representa, de forma abstrata, os componentes de um domínio de negócio, denominados entidades, e seus relacionamentos. Proposto em 1976 , o MER permite a visualização dos aspectos de interesse do sistema e é normalmente implementado como um banco de dados.

Conforme (BEYNON-DAVIES, 2004), uma entidade pode ser definida como algo com existência independente e que possa ser identificada de maneira exclusiva. Para o autor, normalmente uma entidade refere-se a um aspecto do mundo real, distinguível dos demais. É importante ressaltar que isso não necessariamente diz respeito a objetos físicos, mas conceitos abstratos, por exemplo uma reserva ou uma venda.

A figura 2.9 apresenta um exemplo de uma entidade Produto, com os atributos Código, Nome e Preço. O atributo Código é o atributo identificador da entidade.

Figura 2.9 – Exemplo de uma Entidade no MER

Fonte: Elaborada pelo autor.

Os relacionamentos entre as entidades no modelo entidade relacionamento podem ser categorizados em três tipos: a) um para um; b) um para muitos; e c) muitos para muitos.

O relacionamento um para um indica os casos em que uma instância da entidade A está associada a no máximo uma instância da entidade B, e vice-versa. Por exemplo, considerando-se as entidades Gerente e Departamento sob um relacionamento do tipo um para um, estaria definido que um gerente só pode gerenciar no máximo um departamento e que um departamento só pode ser gerenciado por no máximo um gerente.

O relacionamento um para muitos descreve os casos em que uma instância da entidade A pode ser associada e mais de uma instância da entidade B, porém uma instância da entidade B só se associa com no máximo uma instância da entidade A. Como exemplo, ao considerar as entidades Departamento e Funcionário sob um relacionamento do tipo um (departamento) para muitos (funcionários), estaria definido que um departamento pode possuir vários funcionários, porém cada funcionário pertence a no máximo um departamento.

O relacionamento muitos para muitos contempla os casos em que as instâncias de ambas as entidades podem ter associação com mais de uma instância da outra. Por exemplo, considerando as entidades Funcionário e Função em um relacionamento muitos para muitos, cada funcionário pode executar mais de uma função, e cada função pode ser executada por mais de um funcionário.

A cardinalidade de cada entidade em um relacionamento é definida por dois números e é representada graficamente pela notação (x, x). O segundo número indica a quantidade máxima de instâncias da entidade que podem participar da relação. É este valor que é utilizado para definir o tipo de relacionamento (um para um, um para muitos ou muitos para muitos).

Paralelamente, o primeiro número indica a quantidade mínima de instâncias que devem participar da relação. Ele pode assumir os valores 0 (onde não há quantidade mínima) e 1 (onde no mínimo uma instância da entidade deve participar da relação).

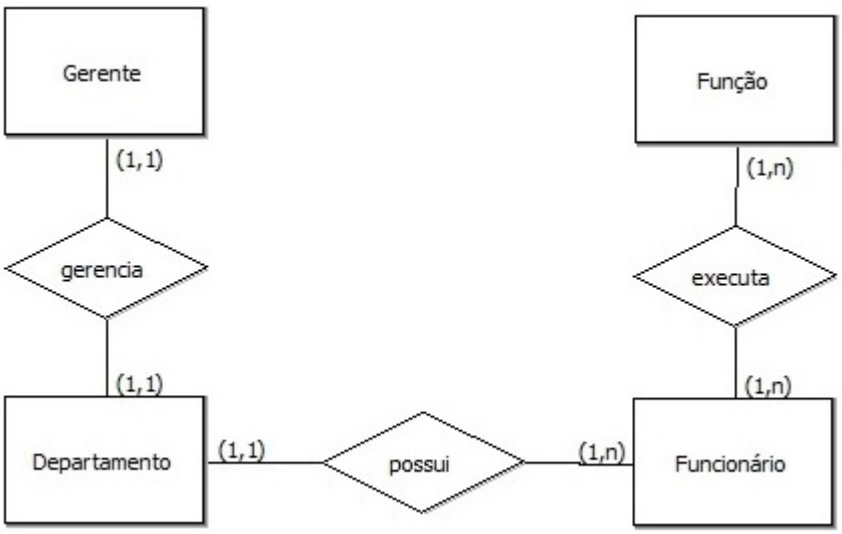
A tabela 2.1 apresenta as combinações possíveis destes valores e seus significados.

Tabela 2.1 – Tipos de Relacionamentos em um MER

`(0,1)`	No máximo uma instância da entidade pode participar do relacionamento, mas pode não haver nenhuma
`(1,1)`	Exatamente uma instância da entidade deve participar do relacionamento
`(0,n)`	Pode haver qualquer quantidade de instâncias da entidade participando do relacionamento, inclusive nenhuma
`(1,n)`	Podem haver uma ou mais instâncias da entidade participando do relacionamento

A figura 2.10 retrata os relacionamentos entre as entidades Gerente, Departamento, Funcionário e Função, exemplificados nesta seção.

Figura 2.10 – Exemplo de Relacionamentos em um MER



Fonte: Elaborada pelo autor.

2.8 Tecnologias de Desenvolvimento

2.8.1 PHP

PHP (*PHP: Hypertext Preprocessor*) é uma linguagem de *script* de código aberto majoritariamente utilizada para a construção de aplicações web. É uma das linguagens de programação mais utilizadas no mundo.

Seu nome é um acrônomo recursivo, uma brincadeira comum no meio da computação. O primeiro P em PHP refere-se a PHP, cujo primeiro P refere-se a PHP, e assim por diante.

2.8.2 MySQL

O MySQL é um sistema gerenciador de banco de dados (SGBD) relacional de código aberto que faz uso da linguagem SQL (*Structured Query Language*). Ele opera no modelo cliente-servidor e é um dos SGBDs mais populares do mundo, destacando-se por sua flexibilidade e facilidade de uso, além de ser considerado muito seguro.

2.8.3 React Native

React Native é um framework criado pelo Facebook e voltado para o desenvolvimento de aplicações móveis para Android e iOS.

As aplicações desenvolvidas com este framework são híbridas, ou seja, podem ser publicadas tanto para Android, quanto para iOS, por meio do mesmo código produzido.

O *React Native* faz uso da linguagem de programação *Javascript* e gera aplicações nativas que independem de outras ferramentas para serem executadas no *smartphone*.

2.9 Teste de Software

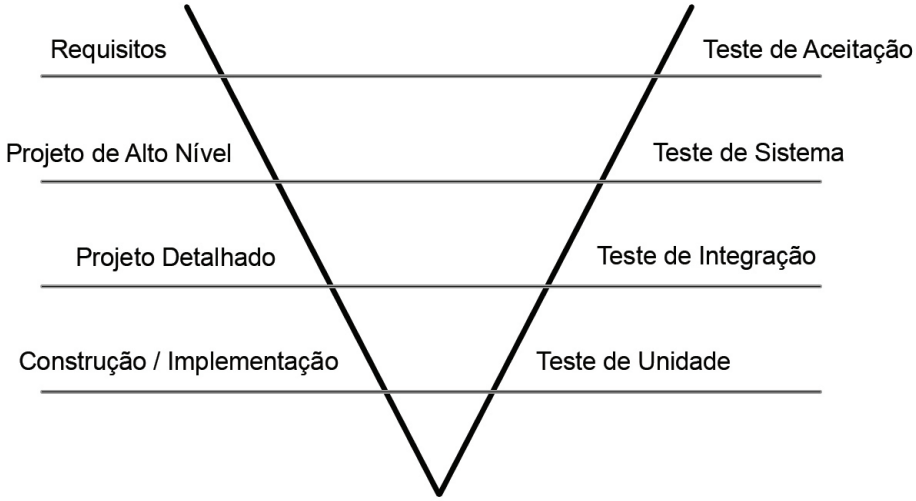
O teste de software é o processo de se buscar incoerências nas funcionalidades do software. Este processo procura melhorar a qualidade final do produto, reduzindo a ocorrência de imprevistos na sua execução. É importante ser ressaltado que o teste de software não busca provar que o funcionamento do software é perfeito, tampouco torná-lo assim. Nas palavras de (AMMANN; OFFUTT, 2016), "o teste de software pode mostrar somente a presença de falhas, não sua ausência". Em outras palavras, como explicam (MYERS; SANDLER; BADGETT, 2011), até mesmo em sistemas pequenos e de pouca complexidade, há inúmeras combinações possíveis de entrada e saída de dados, o que torna impraticável o teste de todas elas. Por este motivo, o processo de teste de software deve ser feito de maneira consciente, o que é possível por meio do emprego de técnicas predefinidas.

2.9.1 Modelo V - Desenvolvimento de Software

Como enfatizam (MATHUR; MALIK, 2010), o teste de software é importante demais para ser deixado apenas para as etapas finais de desenvolvimento. Tendo isto como base, o modelo V incorpora o processo de teste a todo o ciclo de vida do desenvolvimento de software, caracterizando um modelo de desenvolvimento. Para (MATHUR; MALIK, 2010), isso ocorre por causa do modelo V considerar a existência de diferentes níveis de teste e como cada nível se relaciona com uma etapa do desenvolvimento de software.

A cada etapa do desenvolvimento de software, trabalha-se cada vez com um escopo menor e mais específico. De maneira contrária, a cada etapa de teste, trabalha-se com um escopo mais abrangente. É com base nesse fundamento que o modelo V recebe seu nome, pois graficamente esse processo assume a forma de um V. Uma representação gráfica do modelo V é apresentada na figura 2.11.

Figura 2.11 – Modelo V



Fonte: Elaborada pelo autor.

Além de apresentar o ciclo de vida do desenvolvimento, o modelo V mostra como cada etapa de teste se relaciona com cada etapa de implementação do software. O teste de unidade avalia o que foi produzido na etapa de construção/implementação, o teste de integração avalia a conformidade com o projeto detalhado, o teste de sistema avalia a conformidade com o projeto de alto nível e o teste de aceitação avalia o que foi produzido na etapa de levantamento de requisitos.

No teste de unidade, são realizados testes em componentes individuais do software, de forma a examinar se operam da maneira prevista. Conforme (MYERS; SANDLER; BADGETT, 2011), essa etapa tem seu foco no teste de subprogramas, subrotinas, classes e procedimentos do software. De acordo com os autores, a realização de testes de unidade oferece três benefícios claros: a) oferece uma forma de gerenciar os elementos de teste, por focar a atenção em pequenas unidades do software, b) facilita a correção dos erros encontrados, por indicar exatamente onde o erro ocorreu e c) possibilita paralelismo no processo de testes, pois diversos componentes podem ser testados concomitantemente. O teste de unidade faz uso de técnicas de teste de caixa branca (MATHUR; MALIK, 2010).

No teste de integração, o escopo de realização do teste é ampliado, sendo analisada a integração de diferentes módulos e componentes do sistema, e sua execução conjunta. Esta etapa faz uso de técnicas de teste de caixa preta (MATHUR; MALIK, 2010).

No teste de sistema, o software é examinado como um todo, de maneira completa e integrada, com o objetivo de verificar o seu funcionamento de acordo com os requisitos identificados na etapa de projeto.

Conforme explicam (MATHUR; MALIK, 2010), essa etapa de testes não requer conhecimento sobre o funcionamento interno do software.

No teste de aceitação, analisa-se a consonância do software com os requisitos e expectativas do cliente.

2.9.2 Teste de Caixa Preta

O teste de caixa preta é uma técnica de teste que não leva em consideração o funcionamento interno do sistema ou componente testado. Nele, examinam-se os resultados produzidos a partir de um conjunto de entradas, de maneira a certificar a adequação do sistema ou componente ao que foi projetado. Conforme (AMMANN; OFFUTT, 2016), esta técnica de teste se baseia nas descrições externas do software, como especificações, requisitos e projeto.

2.9.3 Teste de Caixa Branca

Ao contrário do teste de caixa preta, o teste de caixa branca atua com base no funcionamento interno do componente (AMMANN; OFFUTT, 2016). Os testes são projetados de maneira a verificar o fluxo de diferentes entradas ou combinações de entradas dentro do componente, e como ele lida com cada uma.

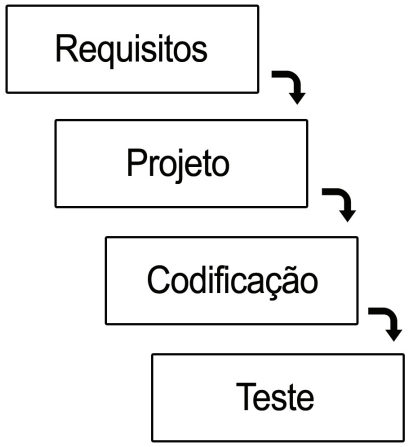
2.9.4 Teste de Caixa Cinza

O teste de caixa cinza busca combinar a qualidade de ambos, teste de caixa preta e teste de caixa branca. Esta técnica de teste utiliza a mesma abordagem do teste de caixa preta, analisando os resultados produzidos para cada variação de entrada, porém usufruindo do conhecimento da estrutura interna do componente, característica do teste de caixa branca.

3 PROCESSO DE DESENVOLVIMENTO

Para consecução do objetivo deste TCC, desenvolver um aplicativo móvel e uma API, foi utilizado o modelo cascata de desenvolvimento, e estabelecidas quatro etapas: a) requisitos; b) projeto; c) codificação; e d) teste, conforme representado na Figura 3.1. A opção pelo modelo cascata justifica-se por se tratar de uma aplicação pequena, com seus requisitos bem definidos e desenvolvida por apenas uma pessoa.

Figura 3.1 – Processo de Desenvolvimento



Fonte: Elaborada pelo autor.

A etapa de requisitos consiste em compreender o objetivo da aplicação que será desenvolvida e levantar quais funções são necessárias para ela alcançar esse objetivo. Nessa etapa, é imprescindível o conhecimento do domínio de negócio em que a aplicação irá atuar, de maneira a planejar como ela poderá ser útil. A saída desta etapa é um documento de requisitos, a prototipação de interface e um diagrama de casos de uso por ator.

A etapa de projeto consiste em estabelecer a arquitetura do sistema, o MER e os diagramas de classes, de pacotes, de sequência e de implantação. Nesaa etapa, ocorre o planejamento da estrutura e funcionamento do sistema que será codificado. A codificação será orientada pelos diagramas produzidos nesta etapa.

A etapa de codificação consiste em produzir o código do software, de acordo com o que foi projetado na etapa anterior. Nessa etapa, os diagramas construídos previamente são traduzidos para uma linguagem de programação.

A etapa de teste, consiste em examinar o produto desenvolvido com o objetivo de encontrar falhas ou inconformidades. Nessa etapa, são aplicadas diferentes técnicas de teste em diferentes níveis do software. Serão avaliados desde componentes e módulos específicos, a integração e comunicação entre essas diferentes

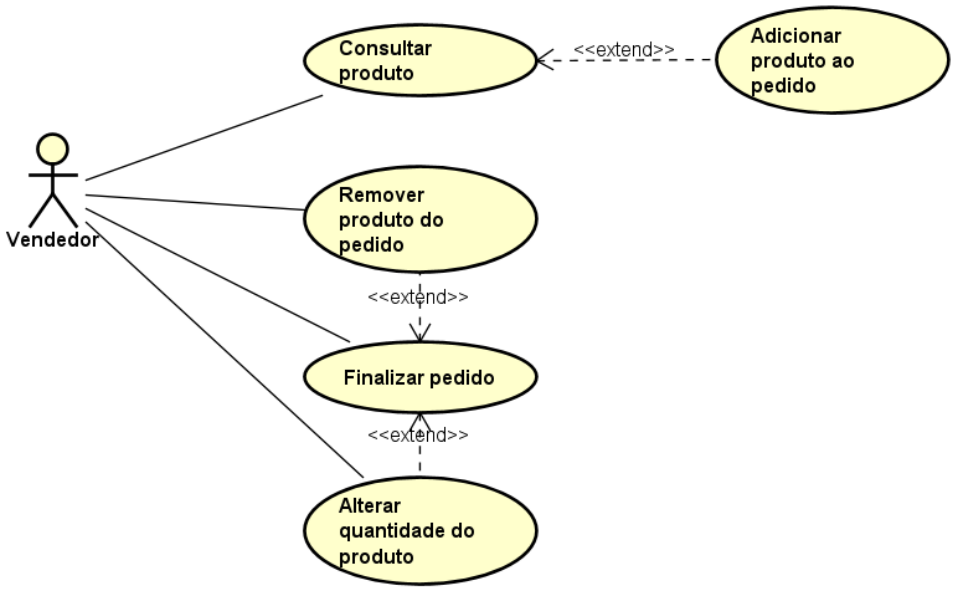
partes do software até o produto de software como um todo, seu atendimento aos requisitos identificados e a aceitação do produto final pelo cliente.

4 DESENVOLVIMENTO

4.1 Requisitos

Após o levantamento dos requisitos do sistema, foram identificados os casos de uso a) consultar produto, b) adicionar produto ao pedido, c) remover produto do pedido, d) alterar quantidade do produto e e) finalizar pedido. Na Figura 4.1 é apresentado o diagrama de casos de uso para o ator Vendedor, o único ator que utiliza a aplicação.

Figura 4.1 – Diagrama de Casos de Uso



Fonte: Elaborada pelo autor.

Conforme representado na Figura 4.1, o caso de uso Adicionar Produto ao Pedido estende o caso de uso Consultar Produto, como indicado pela notação *«extend»*. Isso significa que, após consultar o produto, o vendedor pode optar por adicioná-lo ao pedido. Da mesma forma, os casos de uso Remover Produto do Pedido e Alterar Quantidade do Produto estendem o caso de uso Finalizar Pedido. Estes, por outro lado, podem ser realizados de modo direto.

Após a construção do diagrama, foi realizada a descrição de cada caso de uso. As figuras 4.2 e 4.3 apresentam a descrição dos casos de uso Consultar Produto e Confirmar Pedido, respectivamente.

Para o caso de uso Consultar Produto, conforme a Figura 4.2, o fluxo principal consiste de envio de dados de identificação do produto pelo vendedor, consulta ao banco de dados e exibição de informações do

Figura 4.2 – Descrição do Caso de Uso Consultar Produto

Nome do Caso de Uso:	Consultar Produto	
Resumo:	O vendedor realiza a consulta de informações de um produto específico	
Pré-condição:	O vendedor precisa estar autenticado no sistema	
Pós-condição:		
Fluxo Principal:	Vendedor: 1. Informa o código ou parte do nome do produto desejado;	Sistema: 2. Consulta o produto no banco de dados; 3. Exibe as informações do produto para o vendedor.
Fluxo Alternativo 1:	Cliente:	Sistema: 2. Se o produto não foi encontrado, exibe a mensagem "Produto não encontrado".

Fonte: Elaborada pelo autor.

Figura 4.3 – Descrição do Caso de Uso Confirmar Pedido

Nome do Caso de Uso:	Finalizar Pedido	
Resumo:	O vendedor confirma um pedido.	
Pré-condição:	O vendedor precisa estar autenticado no sistema	
Pós-condição:		
Fluxo Principal:	Vendedor: 1. Clica em "Finalizar Pedido".	Sistema: 2. Salva os dados do pedido no banco de dados; 3. Envia email de novo pedido registrado para a gerência. 4. Exibe mensagem "Pedido Cadastrado" para o vendedor.

Fonte: Elaborada pelo autor.

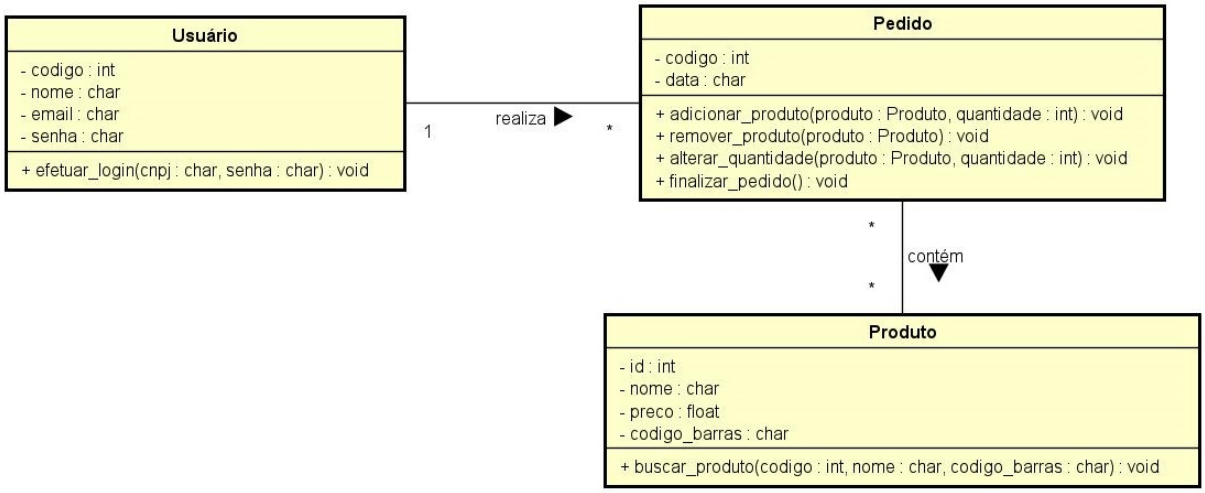
produto pelo sistema. Há também descrito um fluxo alternativo para o caso do produto não ser encontrado no banco de dados.

O caso de uso Finalizar Pedido, conforme Figura 4.3 consiste no vendedor clicar no botão "Finalizar Pedido", na gravação do pedido no banco de dados pelo sistema, além de envio de e-mail para a gerência informando novo pedido cadastrado e a apresentação de uma mensagem de sucesso. Não há fluxo alternativo estabelecido para este caso de uso.

4.2 Projeto

Após a etapa de requisitos, foi projetada a estrutura do sistema. O primeiro passo foi identificar as classes necessárias e construir o diagrama de classes. A figura 4.4 apresenta o diagrama produzido. Também estão representadas no diagrama os relacionamentos entre as classes, onde o Usuário realiza o Pedido, e o Pedido contém Produtos.

Figura 4.4 – Diagrama de Classes



Fonte: Elaborada pelo autor.

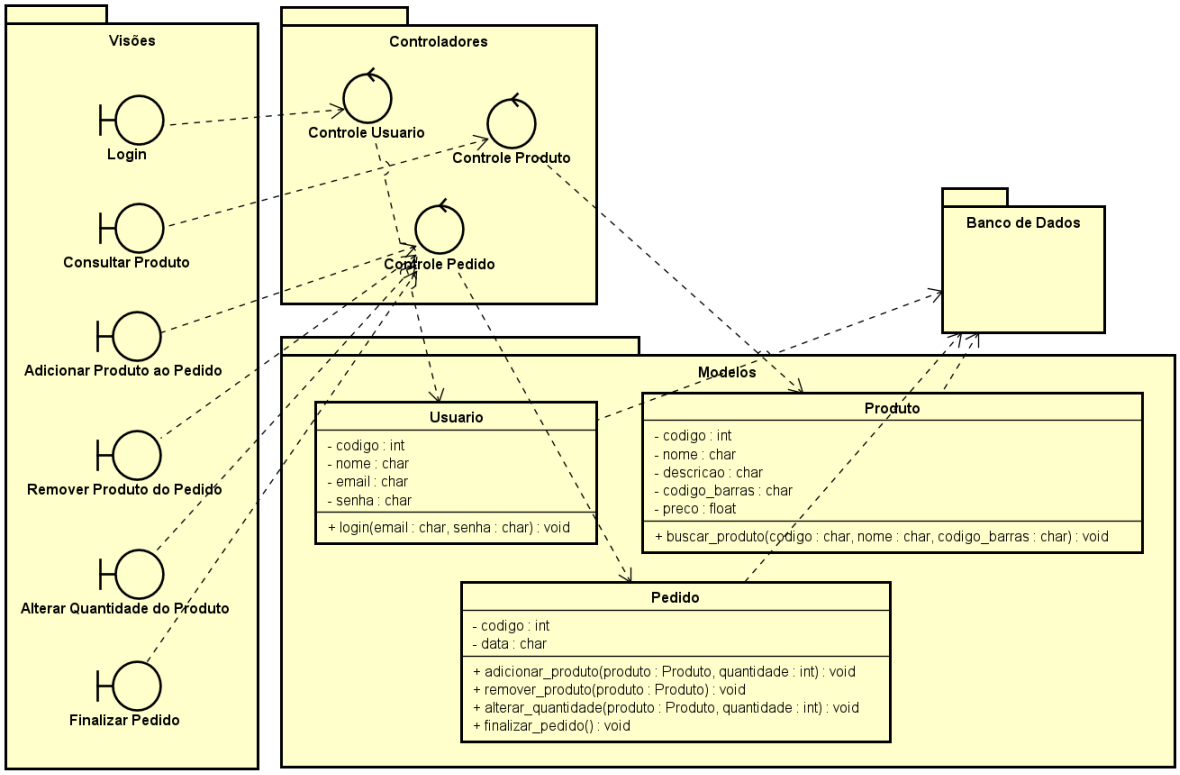
A aplicação foi desenvolvida sob o paradigma de orientação a objetos e construído nos moldes da arquitetura MVC (Modelo, Visão, Controle), onde os modelos realizam o acesso ao banco de dados e manipulam as informações dos elementos da aplicação, as visões trabalham a apresentação de informações e interação com o usuário e os controladores definem o fluxo de execução. Na Figura 4.5, é apresentado o diagrama de pacotes construído.

Conforme observa-se na Figura 4.5, identificou-se a necessidade de criação de três modelos: a) Usuário, b) Produto e c) Pedido, três controladores: a) Controle Usuario, b) Controle Produto e c) Controle Pedido e seis visões: a) Login, b) Consultar Produto, c) Adicionar Produto ao Pedido, d) Remover Produto do Pedido, e) Alterar Quantidade do Produto e f) Finalizar Pedido.

Também durante o projeto da aplicação, foram desenvolvidos os diagramas de sequência. Na Figura 4.6 observa-se o diagrama do caso de uso Consultar Produto.

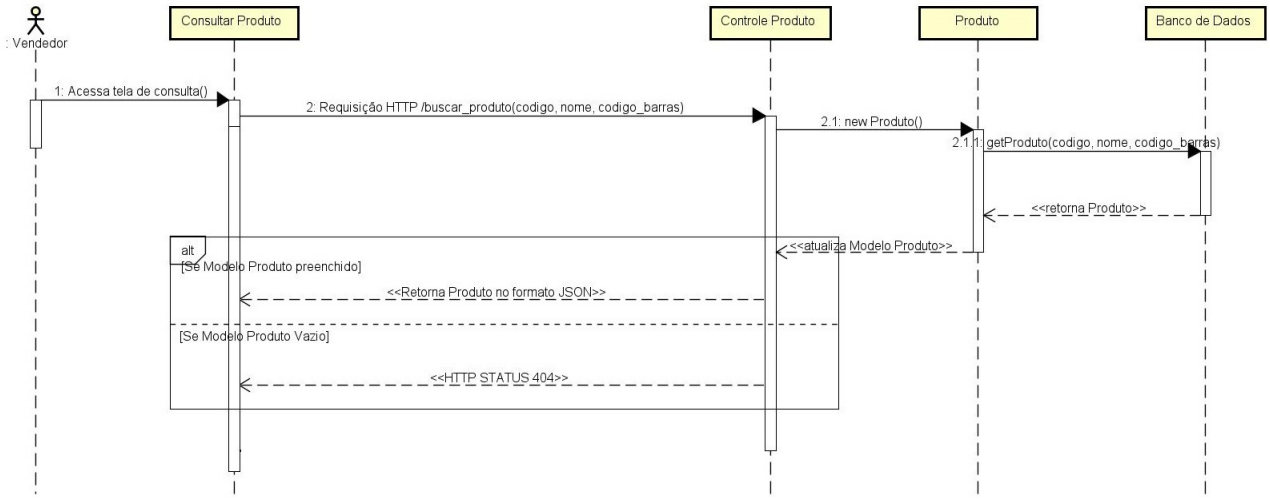
Conforme pode ser visto na Figura 4.6, a visão Consultar Produto se comunica com o Controle Produto através de uma requisição HTTP, enviando os dados informados pelo vendedor na forma de uma mensagem síncrona. O controle aciona o modelo Produto também por meio de uma mensagem síncrona, e este faz o acesso ao banco de dados com as informações recebidas e se atualiza com base no retorno da consulta. Após, conforme é definido no diagrama, há uma condição que verifica o estado do modelo Produto. Caso o modelo esteja preenchido com as informações do produto buscado, o controle responde enviando os dados deste produto para a visão no formato JSON (*JavaScript Object Notation*). Caso o modelo não

Figura 4.5 – Diagrama de Pacotes



Fonte: Elaborada pelo autor.

Figura 4.6 – Diagrama de Sequência

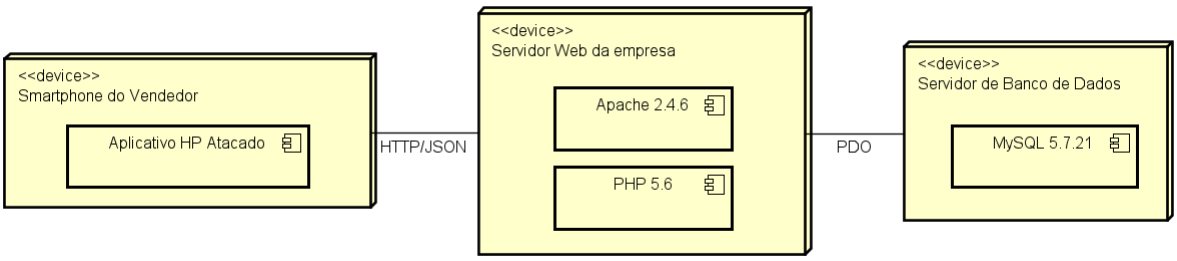


Fonte: Elaborada pelo autor.

esteja preenchido, o controle responde o status HTTP 404 para a visão, o que indica que o produto não foi encontrado.

Ainda na etapa de projeto, foi construído o diagrama de implantação, apresentado na figura 4.7. Para a aplicação descrita neste trabalho, três dispositivos são necessários: a) o smartphone do vendedor que irá executar a aplicação móvel, b) o servidor web da empresa e c) o servidor de banco de dados. A comunicação entre a aplicação móvel no smartphone e o servidor web ocorre sob o protocolo HTTP, com dados organizados no formato JSON e a comunicação do servidor web com o banco de dados ocorre por meio de PDO (*PHP Data Objects*).

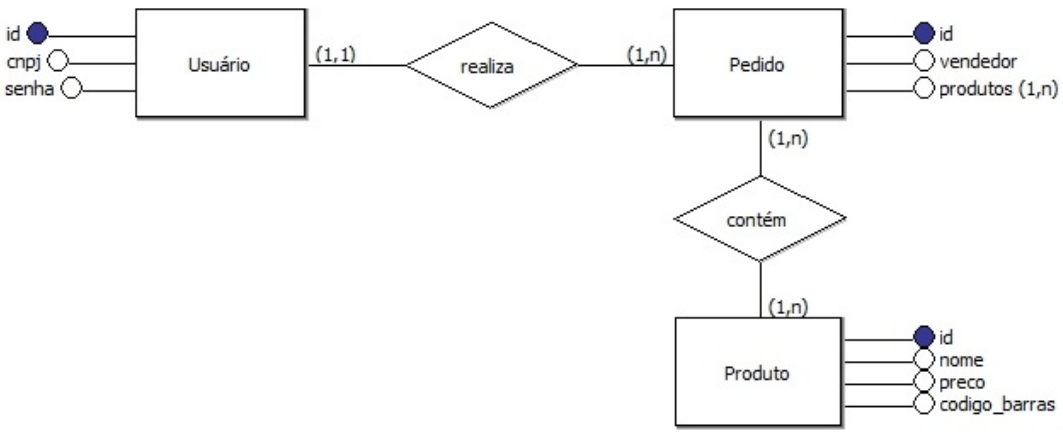
Figura 4.7 – Diagrama de Implantação



Fonte: Elaborada pelo autor.

Também foi desenvolvido, nesta etapa, o Modelo Entidade Relacionamento (MER) da aplicação, representado na Figura 4.8.

Figura 4.8 – MER



Fonte: Elaborada pelo autor.

No MER apresentado na Figura 4.8, observam-se a cardinalidade dos relacionamentos entre as entidades Usuário, Pedido e Produto. Conforme definido, um usuário pode realizar diversos pedidos, porém

cada pedido é realizado por apenas um usuário. Os pedidos, por sua vez, contém diversos produtos, assim como cada produto pode estar contido em diversos pedidos.

Por fim, na fase de projeto, foram desenhados protótipos de interface para as telas da aplicação. Os protótipos foram construídos com o apoio da ferramenta Balsamiq. As Figuras 4.9, 4.10 e 4.11 apresentam os protótipos das telas de Login, Consultar Produto e Finalizar Pedido, respectivamente.

Figura 4.9 – Protótipo da tela de Login

Fonte: Elaborada pelo autor.

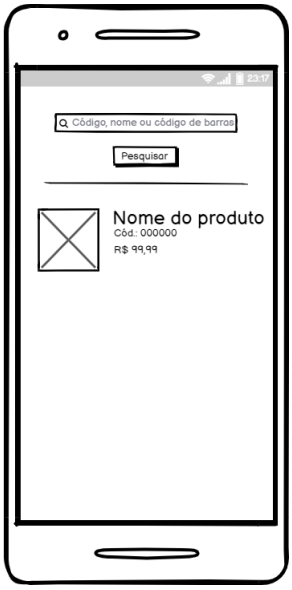
A tela de login consiste de um formulário para inserção dos dados para autenticação e é a primeira tela exibida ao acessar o aplicativo. A tela de consulta possui um campo para inserção de dados do produto a ser buscado, e exibe, abaixo, informações do produto, caso seja localizado. A tela de finalização de pedido, por fim, apresenta o somatório do preço de todos os produtos adicionados, além de um resumo do pedido contendo dados destes produtos.

4.3 Codificação

O projeto foi desenvolvido ao decorrer de 4 meses e foram produzidas 3.569 linhas de código. Destas, 2.355 foram produzidas para a aplicação móvel, e as restantes 1.214 para o back-end e API.

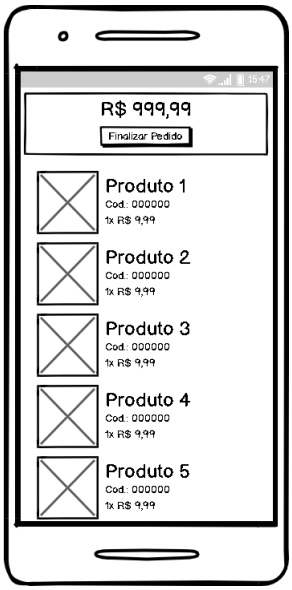
O back-end da aplicação foi desenvolvido com a linguagem PHP, versão 5.6 e está hospedado em um servidor Apache, versão 2.4.6, executando sob um sistema operacional Linux. A escolha destas tecnologias,

Figura 4.10 – Protótipo da tela Consultar Produto



Fonte: Elaborada pelo autor.

Figura 4.11 – Protótipo da tela Finalizar Pedido



Fonte: Elaborada pelo autor.

e suas respectivas versões, sustenta-se no fato de serem as tecnologias atuais utilizadas no *e-commerce* da empresa, hospedado no mesmo servidor da aplicação descrita neste trabalho.

A aplicação móvel foi desenvolvida para as plataformas Android e iOS, por meio do framework *React Native*, na sua versão 0.62, com o suporte do framework Expo, em sua versão 3.27.8. A opção

pelo *React Native* foi devido a (1) familiaridade do autor com a linguagem *Javascript* e (2) se tratar de um framework híbrido, o que possibilita a publicação da aplicação para os dois sistemas operacionais mais comuns para *smartphones*, com a produção de apenas um código.

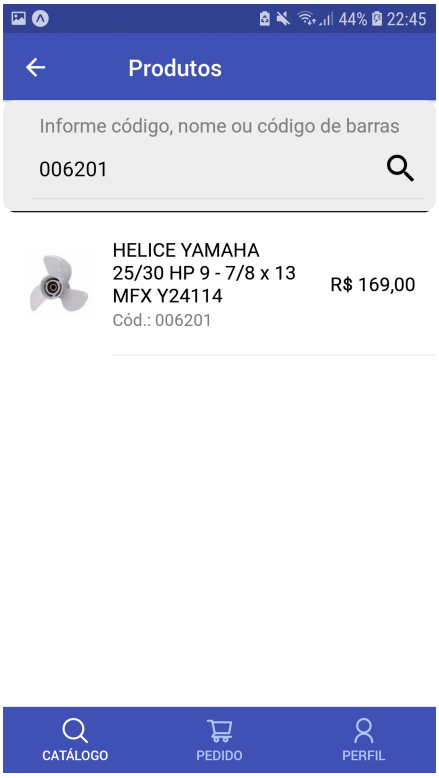
As Figuras 4.12, 4.13 e 4.14 apresentam as telas do produto e do pedido, respectivamente.

Figura 4.12 – Tela de login

Fonte: Elaborada pelo autor.

A tela de login, conforme previamente desenhada, possui um formulário para inserção dos dados para autenticação. A tela de consulta possui um campo para inserção de dados do produto a ser buscado, e exibe informações do produto, caso seja localizado. No exemplo da Figura 4.13, a consulta foi feita por meio da inserção do código do produto. A tela de finalização disponível na Figura 4.14 contém todas as informações definidas durante a fase de projeto, porém com uma pequena alteração: a inserção de um título no topo, que tem a intenção de lembrar o vendedor que o pedido precisa ser confirmado para que seja enviado para a empresa. Foi também alterado o texto do botão "Finalizar Pedido", que se tornou, na versão atual, "Enviar Pedido".

Figura 4.13 – Tela de consulta



Fonte: Elaborada pelo autor.

4.4 Testes

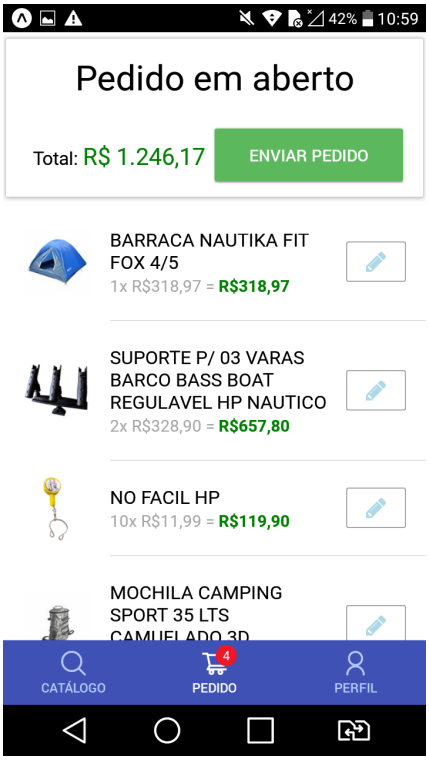
O teste da aplicação se dividiu basicamente em duas etapas: a) teste manual de caixa preta e b) teste de aceitação.

O teste manual de caixa preta ocorreu por meio de duas maneiras: 1) realizando requisições à API utilizando diferentes formatos de dados, e 2) desempenhando os casos de uso definidos durante a etapa de levantamento de requisitos do sistema, tanto para os fluxos pretendidos, quanto para os fluxos alternativos, como por exemplo o envio de informações incorretas.

O teste de aceitação ocorreu através da utilização de uma versão premilinar da aplicação por um número limitado de vendedores. Apesar de não se tratar da aplicação final, a versão utilizada continha todas as funcionalidades estipuladas durante a etapa de levantamento de requisitos do sistema.

Os testes de caixa preta foram fundamentais para identificar inconsistências na execução do sistema, o que permitiu a correção de erros não identificados durante o desenvolvimento.

Figura 4.14 – Tela do pedido



Fonte: Elaborada pelo autor.

O teste de aceitação, por sua vez, possibilitou o empreendimento de melhorias no processo de execução das atividades da aplicação, além de aprimorar a sua interface, de forma a prover maior facilidade e velocidade na sua utilização pelo seu público alvo intencionado.

5 RESULTADOS ALCANÇADOS

Até então, o acesso dos vendedores externos e representantes às informações de preço e disponibilidade dos produtos do catálogo da empresa ocorriam através de duas maneiras: (1) consulta por ligação telefônica e (2) tabela de produtos impressa.

A primeira forma, apesar de garantir o acesso a informação correta e atualizada, requer maior tempo e pouca praticidade, principalmente levando em consideração o fato de que, na maioria das vezes, o vendedor ou representante está na presença do cliente, e possui um tempo limitado para negociar uma venda.

A segunda forma de consulta, por sua vez, oferece maior praticidade pelo fato de o vendedor ter em mãos todas as informações que precisa. Entretanto, muitas vezes essa informação está inconsistente ou defasada, levando a realização de pedidos de compra que posteriormente necessitam de ratificação, seja por indisponibilidade de produtos solicitados, ou por alterações nos preços das mercadorias. Essas ocorrências não são bem vistas aos olhos dos clientes.

Com o emprego da aplicação móvel, constatou-se que foi possível unir os pontos positivos das duas abordagens utilizadas até então. Em posse do aplicativo, o vendedor tem em mãos, de forma rápida e prática, o acesso às informações do catálogo de produtos da empresa e, além disso, essas informações são consistentes e atualizadas.

Paralelamente, uma segunda dificuldade, apesar de menor, se fazia presente nas vendas externas: o envio de pedidos para a sede da empresa.

Antes da utilização do aplicativo móvel, os vendedores externos e representantes tinham o hábito de anotar os pedidos à mão, fotografá-los e enviá-los via aplicativo de mensagens para a empresa. Este processo trazia basicamente dois problemas principais: (1) ilegibilidade de determinados pedidos, ocasionando trabalho extra e atrasos e (2) ambiguidades de produtos, como, por exemplo, constar no pedido uma unidade de “kit chumbada”, sem especificar se isto se refere ao kit com 25, 50 ou 100 unidades.

A utilização da aplicação móvel também colaborou para facilitar esse processo. Por permitir que o cadastro de pedidos seja feito diretamente na aplicação, a empresa tem acesso de forma digital a lista de produtos solicitados, inclusive com o seu código de identificação, o que elimina quaisquer possíveis falhas de interpretação ou ambiguidades.

Considerando os ganhos obtidos, supracitados, através do uso do aplicativo móvel pelos vendedores em detrimento do uso de catálogo impresso e ligações telefônicas, não há dúvida de que o aplicativo trouxe melhorias no serviço de vendas em atacado da empresa. Apesar de estar em uso há pouco tempo, e por um

número limitado de vendedores, já se cogita o desenvolvimento de novas funcionalidades e da expansão de seu público alvo, podendo futuramente abranger inclusive clientes recorrentes da empresa.

6 CONCLUSÃO

O objetivo deste trabalho foi o desenvolvimento de uma aplicação móvel, acompanhada de uma API para acesso ao servidor, que pudesse facilitar a realização de dois processos por vendedores externos e representantes da empresa HP Pesca, Moto e Náutica Yamaha: a consulta de informações de produtos de maneira rápida e prática, estando essas informações consistentes e atualizadas, e o envio de pedidos para a sede de forma simples e de modo a evitar falhas de interpretação e ambiguidades.

Após o início da utilização do aplicativo móvel por alguns vendedores, constatou-se que esses objetivos foram alcançados com êxito. A aplicação móvel desenvolvida permite ao vendedor a possibilidade de acesso rápido às informações do catálogo de produtos da empresa e, ao mesmo tempo, garante que essa informação é atual e consistente.

O emprego desta aplicação no dia a dia do vendedor externo e representante também colaborou para mitigar as falhas de interpretação e ambiguidades frequentes que ocorriam previamente à posse dessa nova ferramenta. O aplicativo móvel permite o cadastro de pedidos de forma digital, o que permite que a empresa tenha acesso à lista de produtos de maneira detalhada, o que não ocorria quando os pedidos eram manuscritos.

Apesar de trazer os benefícios mencionados, a aplicação, na sua atual versão, possui uma limitação principal: ela depende de acesso a internet constante. Devido a esse fator, o vendedor depende de um plano de internet móvel para utilizar a aplicação em locais que não fornecem conexão de internet. Pretende-se, em uma atualização futura, adaptar a aplicação para uso *offline*, por meio do uso de um banco de dados local, atualizado periodicamente.

Em conclusão, o desenvolvimento e emprego desta aplicação móvel, além de produzir as melhorias já mencionadas para a empresa, também explicita uma questão emergente: aplicativos móveis não são mais exclusividade de grandes companhias. Fica claro que até mesmo empresas locais e regionais podem desfrutar da digitalização de seus processos e da mobilidade oferecida por *smartphones*, que nunca estiveram tão presentes na sociedade quanto agora.

O êxito no emprego desta aplicação móvel no processo de vendas por atacado da empresa já abre portas para a discussão sobre o desenvolvimento de novas funcionalidades. Entre elas, a possibilidade de realização de pedidos recorrentes e a expansão do público alvo para permitir a utilização do aplicativo por alguns clientes frequentes. Por hora, o próximo passo é liberar a aplicação para todos os vendedores externos e representantes da empresa.

REFERÊNCIAS

ABOUT the Unified Modeling Language Specification Version 2.5.1. <https://www.omg.org/spec/UML/About-UML/>. Acessado em 11/05/2021.

AMMANN, P.; OFFUTT, J. **Introduction to software testing**. [S.l.]: Cambridge University Press, 2016.

ANDA, B. et al. Experiences from introducing uml-based development in a large safety-critical project. **Empirical Software Engineering**, Springer, v. 11, n. 4, p. 555–581, 2006.

BELL, D. Uml basics: The sequence diagram. **Diakses dari http://www. ibm. com/developerworks/rational/library/3101. html**, 2004.

BEYNON-DAVIES, P. **Database systems**. [S.l.]: Springer, 2004.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: guia do usuário**. [S.l.]: Elsevier Brasil, 2006.

CHEN, P. P.-S. The entity-relationship model—toward a unified view of data. **ACM transactions on database systems (TODS)**, Acm New York, NY, USA, v. 1, n. 1, p. 9–36, 1976.

FOWLER, M.; KOBRYN, C.; ONLINE, S. T. B. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. Addison-Wesley, 2004. (Object Technology Series). ISBN 9780321193681. Disponível em: <https://books.google.com.br/books?id=nHZslSr1gJAC>.

GENERO, M.; PIATTINI, M.; CALERO, C. Empirical validation of class diagram metrics. In: IEEE. **Proceedings International Symposium on Empirical Software Engineering**. [S.l.], 2002. p. 195–203.

GOMAA, H. **Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures**. Cambridge University Press, 2011. ISBN 9781139494731. Disponível em: <https://books.google.com.br/books?id=TqZi-hAX17YC>.

GUEDES, G. **UML 2 - Uma Abordagem Prática**. Novatec Editora, 2018. ISBN 9788575226445. Disponível em: <https://books.google.com.br/books?id=mJxMDwAAQBAJ>.

LARMAN, C. **Utilizando UML e Padrões**. Grupo A - Bookman, 2000. ISBN 9788577800476. Disponível em: <https://books.google.com.br/books?id=hzl2tmT8QkUC>.

MATHUR, S.; MALIK, S. Advancements in the v-model. **International Journal of Computer Applications**, Citeseer, v. 1, n. 12, p. 29–34, 2010.

MYERS, G.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. Wiley, 2011. (ITPro collection). ISBN 9781118133156. Disponível em: <https://books.google.com.br/books?id=GjyEFPkMCwcC>.

RIBEIRO, K. F. Estudo de caso sobre o uso da linguagem de modelagem uml no processo de levantamento de requisitos no desenvolvimento de aplicações do lcc-ufmg. Universidade Federal de Minas Gerais, 2013.

SOMÉ, S. S. A meta-model for textual use case description. **J. Object Technol.**, v. 8, n. 7, p. 87–106, 2009.