

UNIVERSIDADE DO CONTESTADO – UnC
CURSO DE ENGENHARIA DE SOFTWARE

DEBORA CRISTINA FONTANELLA

**PROTÓTIPO DE SOFTWARE PARA BUSCA E DIVULGAÇÃO DE PRODUTORES
LOCAIS DE ALIMENTOS**

CONCÓRDIA
2020

DEBORA CRISTINA FONTANELLA

**PROTÓTIPO DE SOFTWARE PARA BUSCA E DIVULGAÇÃO DE PRODUTORES
LOCAIS DE ALIMENTOS**

Trabalho de Conclusão de curso, apresentado como exigência para obtenção do título de Bacharel em Engenharia de Software, pela Universidade do Contestado - UNC, Campus Concórdia, sob Orientação Específica do Professor Moacir Solano Kichel e Orientação Metodológica da Professora Gisleine Merib Kichel.

CONCÓRDIA
2020

**PROTÓTIPO DE SOFTWARE PARA BUSCA E DIVULGAÇÃO DE PRODUTORES
LOCAIS DE ALIMENTOS**

DEBORA CRISTINA FONTANELLA

Este trabalho de Conclusão de Curso foi submetido ao processo de avaliação pela Banca Examinadora para a obtenção do Título de:

Bacharel em Engenharia de Software

E aprovada(o) na sua versão final em **XX** de **XXXXXXX** de **XXXX**, atendendo às normas da legislação vigente da Universidade do Contestado e Coordenação do Curso de **XXXXXXXXXX**.

Prof. **XXXXXXX**

Coordenador do Curso de **XXXXXXXXXX**

BANCA EXAMINADORA:

Profº. **XXXXXXXXXX**

(Orientador)

Prof. **XXXXXXXXXX**

(Avaliador)

Prof. **XXXXXXXXXX**

(Avaliador)

*A todos aqueles que estiveram próximos
durante toda minha jornada, contribuindo
com conhecimento, motivação e carinho.
Em especial, à minha mãe, que tornou
tudo isso possível.*

AGRADECIMENTOS

Agradeço primeiramente à minha mãe, Geni Datsch, por sempre me incentivar a estudar e construir em mim a consciência de que o conhecimento é o único bem que não podem nos tirar, bem como o maior meio de transformação de nossa realidade. Além disso, por fornecer todo o suporte e muitas vezes abrir mão de si para possibilitar meus estudos, sendo um exemplo para mim de força e altruísmo. À minha irmã, Ana por junto de minha mãe me dar todo o apoio e amor durante essa jornada.

À minha namorada, Lara Tibolla, por se fazer presente em todos os momentos, especialmente durante o processo de desenvolvimento desse trabalho, acreditando no meu potencial, me incentivando, compreendendo as limitações do momento e me ajudando a manter firmeza e constância, sendo meu porto seguro em todos os momentos.

Aos meus amigos, por compreenderem minhas ausências e por todas as palavras de motivação e carinho.

Ao meu psicólogo, Daniel, por me acompanhar nessa jornada e ajudar a me preparar para esse momento, possibilitando passar por essa fase com mais calma, sabedoria e autoconfiança.

A todos os professores que ao longo dos anos me guiaram através da construção do conhecimento, mostrando comprometimento e dedicação por sua profissão. Em especial, ao meu orientador, Moacir Solano Kichel, por confiar no meu trabalho e se mostrar sempre disposto a auxiliar para aperfeiçoá-lo, contribuindo com seu extenso conhecimento e experiência. Ao professor Geordano Dalmédico, por me orientar durante o início do trabalho, estando sempre à disposição para auxiliar. E, por fim, ao professor e amigo Frederico Barbosa Muniz, por colocar os primeiros tijolos na construção do meu conhecimento específico em TI e me inspirar a entrar na área de desenvolvimento de software, bem como por todos os conselhos dados quando precisei.

A todos, o meu carinhoso muito obrigado! Cada um tem sua contribuição nesse trabalho.

*"Só existe uma maneira de se viver pra sempre, irmão
Que é compartilhando a sabedoria adquirida
E exercitando a gratidão, sempre
É o homem entender que ele é parte do todo
É sobre isso que o manifesto fala
Nem ser menos e nem ser mais, ser parte da natureza, certo?
Ao caminhar na contramão disso
A gente caminha pra nossa própria destruição"*

(Emicida, Lucas Silveira)

RESUMO

O estilo de vida cada vez mais acelerado implica na perda de qualidade de vida, inclusive no que diz respeito à alimentação. Nesse sentido, a busca por uma alimentação mais saudável pode ser tomada por diversas frentes, uma delas é a preferência por consumo de alimentos produzidos próximos ao local de consumo. A esse movimento, dá-se o nome de Locavorismo, que implica no consumo de alimentos com menos agrotóxicos e conservantes, resultando em menor impacto ambiental, fortalecimento da economia local, bem como benefícios ao produtor e consumidor. Entre os problemas enfrentados para a popularização do movimento, encontra-se a dificuldade para conectar consumidores a produtores. Dessa forma, beneficiando-se da crescente popularização de soluções de software para os mais diversos problemas, o protótipo proposto nesse trabalho fornece uma plataforma para centralização de produtores locais de alimentos, possibilitando que consumidores os encontrem de maneira facilitada.

Palavras-Chave: Locavorismo. Alimento Local. Aplicação de Software.

ABSTRACT

The increasingly accelerated lifestyle implies a loss of quality of life, including with regard to food. In this sense, the search for a healthier diet can be taken on several fronts, one of which is the preference for consuming food produced near the place of consumption. This movement is called Locavorism, which implies the consumption of food with less pesticides and preservatives, resulting in less environmental impact, local economy strengthening, as well as benefits for producers and consumers. One of the problems faced to popularize the movement, there is the difficulty in connecting consumers to producers. Thus, taking advantage of the growing popularity of software solutions for the most diverse problems, the prototype proposed in this work provides a platform for centralizing local food producers, allowing consumers to find them in an easier way.

Keywords: Locavorism. Local Food. Software Applications.

LISTA DE FIGURAS

Figura 1: Camadas da engenharia de software.....	27
Figura 2: Representação da metodologia do processo de software.....	29
Figura 3: Ciclo de vida clássico.....	32
Figura 4: Paradigma de prototipação.....	34
Figura 5: Modelo de processo orientado a cronograma.....	36
Figura 6: Representação gráfica UML de Ator, Caso de Uso e Relacionamento de comunicação.....	41
Figura 7: Exemplo de relacionamento de inclusão.....	42
Figura 8: Exemplo de relacionamento de extensão.....	43
Figura 9: Exemplo de relacionamento de generalização entre atores.....	44
Figura 10: Exemplo de relacionamento de generalização entre casos de uso.....	44
Figura 11: Exemplo de caso de uso com representação de fronteira de sistema.....	45
Figura 12: Representação de uma Classe em UML.....	46
Figura 13: Duas formas de representação gráfica de interfaces.....	47
Figura 14: Exemplos de associações entre classes.....	48
Figura 15: Exemplos de associação com uso de símbolos de multiplicidade.....	49
Figura 16: Exemplo de utilização dos recursos UML nome da associação, direção de leitura e papéis.....	50
Figura 17: Exemplo de representação de uma classe associativa.....	51
Figura 18: Exemplo de notação de relações de agregação e composição.....	51
Figura 19: Exemplo de notação de relação de herança.....	52
Figura 20: Exemplo de notação de realização de uma interface.....	52
Figura 21: Relação de dependência.....	53
Figura 22: Exemplos de notação UML para objetos em um diagrama de sequência	54
Figura 23: Notação UML para linha de vida.....	55
Figura 24: Notação UML para mensagens reflexivas.....	57
Figura 25: Exemplo de notação de ocorrência de execução.....	57
Figura 26: Exemplo de diagrama de atividade composto por todos os componentes	60
Figura 27: Notação gráfica de um componente.....	62
Figura 28: Exemplo de arquitetura cliente-servidor para uma biblioteca de filmes....	64

Figura 29: Padrão MVC.....	66
Figura 30: Diagrama de classes de uma aplicação implementada no padrão Transaction Script.....	67
Figura 31: Exemplo de modelo de entidade-relacionamento.....	71
Figura 32: Representação de um sistema de banco de dados.....	72
Figura 33: Implementação cliente-servidor de uma aplicação Web.....	78
Figura 34: Camadas de uma aplicação web.....	78
Figura 35: Estrutura de um elemento HTML.....	79
Figura 36: estrutura de um conjunto de regras CSS.....	80
Figura 37: Exemplo de um bloco de código PHP entre código HTML.....	82
Figura 38: Editor SQL Embutido do MySQL Workbench.....	87
Figura 39: Interface para criação de modelos ER do MySQL Workbench.....	88
Figura 40: Interface do StarUML.....	89
Figura 41: Interface do editor VS Code.....	90
Figura 42: Diagrama de Caso de Uso.....	100
Figura 43: Diagrama de classes.....	124
Figura 44: Diagrama de atividade do fluxo completo da aplicação.....	126
Figura 45: Visão dos relacionamentos entre entidades do banco de dados.....	128
Figura 46: Visão expandida das entidades apresentadas do modelo ER.....	129
Figura 47: Menu compactado.....	131
Figura 48: Visão expandida do menu para usuário desconectado.....	131
Figura 49: Visão expandida do menu para usuário produtor.....	132
Figura 50: Interface de Cadastro de usuário.....	133
Figura 51: Interface de Login.....	134
Figura 52: À esquerda, perfil do usuário padrão, à direita, perfil do usuário produtor.	135
Figura 53: Formulário para alteração dos dados do perfil, visão do produtor.....	136
Figura 54: <i>Frame</i> para alteração de senha.....	137
Figura 55: <i>Frame</i> para exclusão da conta.....	137
Figura 56: Interface para anexo de imagens ao perfil.....	138
Figura 57: Lista de produtos cadastrados.....	139
Figura 58: Formulário de edição de produto.....	140
Figura 59: Interface para anexo de imagens ao produto.....	141

Figura 60: Lista de endereços cadastrados para o usuário.....	142
Figura 61: Formulário para edição do endereço.....	143
Figura 62: Frame para seleção da localização parâmetro da busca.....	144
Figura 63: Resultado da busca por produtos.....	145
Figura 64: Resultados da busca por produtores.....	146
Figura 65: Tela inicial - Lista de produtores favoritos.....	148
Figura 66: Tela inicial – Lista de resultados compatíveis com interesses.....	148
Figura 67: Resultados da busca dispostos no mapa.....	150
Figura 68: Tela de detalhes do produtor.....	150
Figura 69: Interface de detalhes do produto.....	152
Figura 70: Lista de contatos associados a um endereço.....	154
Figura 71: Formulário de edição de contato.....	154
Figura 72: Lista e cadastro de palavras-chave de interesses.....	155
Figura 73: Formulário para envio de feedback para a equipe de desenvolvimento.	155
Figura 74: Diagrama de componentes representando a arquitetura geral de componentes da aplicação.....	156
Figura 75: Diagrama de componentes do módulo de cadastro de produtos na aplicação cliente.....	158
Figura 76: Diagrama de componentes do módulo de cadastro de produtos na API	160

LISTA DE QUADROS

Quadro 1: Atividades de apoio típicas.....	29
Quadro 2: Simbologia para representar multiplicidades.....	49
Quadro 3: Conectividades em relação às multiplicidades.....	50
Quadro 4: Notação gráfica para tipos de mensagem em um diagrama de sequência	57
Quadro 5: Requisitos funcionais.....	98
Quadro 6: Requisitos não funcionais.....	99
Quadro 7: Descrição do Caso de uso UC01.....	102
Quadro 8: Descrição do Caso de uso UC02.....	102
Quadro 9: Descrição do caso de uso UC03.....	103
Quadro 10: Descrição do Caso de uso UC04.....	104
Quadro 11: Descrição do Caso de uso UC05.....	105
Quadro 12: Descrição do Caso de uso UC06.....	106
Quadro 13: Descrição do Caso de uso UC07.....	107
Quadro 14: Descrição do Caso de uso UC08.....	108
Quadro 15: Descrição do Caso de uso UC09.....	109
Quadro 16: Descrição do Caso de uso UC10.....	109
Quadro 17: Descrição do Caso de uso UC11.....	110
Quadro 18: Descrição do Caso de uso UC12.....	111
Quadro 19: Descrição do Caso de uso UC13.....	111
Quadro 20: Descrição do Caso de uso UC14.....	112
Quadro 21: Descrição do Caso de uso UC15.....	113
Quadro 22: Descrição do Caso de uso UC16.....	115
Quadro 23: Descrição do Caso de uso UC17.....	116
Quadro 24: Descrição do Caso de uso UC18.....	116
Quadro 25: Descrição do Caso de uso UC19.....	117
Quadro 26: Descrição do Caso de uso UC20.....	118
Quadro 27: Descrição do Caso de uso UC21.....	119
Quadro 28: Descrição do Caso de uso UC22.....	120
Quadro 29: Descrição do Caso de uso UC23.....	122
Quadro 30: Descrição do Caso de uso UC24.....	123
Quadro 31: Descrição do Caso de uso UC25.....	124

LISTA DE ABREVIATURAS E/OU SIGLAS

- App – Aplicativo
Ajax – *Asynchronous JavaScript and XML*
CSS – *Cascade Style Sheet*
DCU – Diagrama de Caso de Uso
DDL – *Data-Definition Language*
DML – *Data Manipulation Language*
HTML – *Hyper Text Markup Language*
HTTP – *Hyper Text Transfer Protocol*
HTTPS – *Hyper Text Transfer Protocol Secure*
JSON – *JavaScript Object Notation*
MCU – Modelo de Caso de Uso
MVC – *Model View controller*
PHP – *Hypertext Preprocessor*
PWA – *Progressive Web App*
REST – *Representational State Transfer*
SGBD – Sistema Gerenciador de Banco de Dados
SIBC – Sistema de informação baseado em computador
SOAP – *Simple Object Access Protocol*
SQL – *Structured Query Language*
UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO.....	19
1.1 OBJETIVOS.....	20
1.1.1 Objetivo Geral.....	20
1.1.2 Objetivos Específicos.....	20
2 REFERENCIAL TEÓRICO.....	21
2.1 LOCAVORISMO.....	21
2.1.1 Alimento Local.....	22
2.2 SOFTWARE E SISTEMAS DE INFORMAÇÃO.....	23
2.3 ENGENHARIA DE SOFTWARE.....	26
2.4 PROCESSO DE SOFTWARE.....	28
2.5 MODELOS DE PROCESSO DE SOFTWARE.....	31
2.5.1 Modelo Cascata.....	32
2.5.2 Modelos de Processo Evolucionários.....	33
2.5.2.1 Prototipação.....	33
2.5.2.2 Modelo orientado a cronograma.....	35
2.6 REQUISITOS DE SOFTWARE.....	36
2.6.1 Requisitos Funcionais e Não Funcionais.....	36
2.6.2 Engenharia de Requisitos.....	37
2.7 UML.....	38
2.7.1 Diagrama de Caso de Uso.....	39
2.7.1.1 Caso de uso.....	39
2.7.1.2 Atores.....	40
2.7.1.3 Relacionamentos.....	40
2.7.1.3.1 Relacionamento de comunicação.....	40
2.7.1.3.2 relacionamento de inclusão.....	41
2.7.1.3.3 Relacionamento de extensão.....	42
2.7.1.3.4 Relacionamento de generalização.....	43
2.7.1.4 Fronteira do sistema.....	44
2.7.2 Diagrama de Classes.....	45
2.7.2.1 Classes.....	46
2.7.2.1.1 Interfaces.....	47

2.7.2.2 Associações.....	47
2.7.2.3 Multiplicidades.....	48
2.7.2.4 Participações.....	49
2.7.2.5 Nome de associação, direção de leitura e papéis.....	49
2.7.2.6 Classes associativas.....	50
2.7.2.7 Agregações e composições.....	51
2.7.2.8 Generalizações e especializações.....	52
2.7.2.9 Realização.....	52
2.7.2.10 Dependência.....	53
2.7.3 Diagrama de Sequência.....	54
2.7.3.1 Atores.....	54
2.7.3.2 Objetos.....	54
2.7.3.3 Linha de vida.....	55
2.7.3.4 Mensagens.....	55
2.7.3.5 Ocorrências de execução.....	57
2.7.3.6 Criação e destruição de objetos.....	58
2.7.4. Diagrama de Atividade.....	58
2.7.4.1 Nó de ação.....	58
2.7.4.2 Nó inicial.....	59
2.7.4.3 Nó final.....	59
2.7.4.4 Fork.....	59
2.7.4.5 Join.....	59
2.7.4.6 Nó de decisão.....	59
2.7.4.7 Raias.....	60
2.7.5 Diagrama de Componentes.....	61
2.7.5.1 Componentes.....	61
2.8 ARQUITETURA DE SOFTWARE.....	62
2.8.1 Arquitetura Cliente-Servidor.....	63
2.8.2 Arquitetura MVC.....	64
2.8.3 Arquitetura de Transação (<i>Transaction Script</i>).....	66
2.9 TESTES.....	67
2.10 BANCO DE DADOS.....	69
2.10.1 Modelos de Dados.....	70

2.10.1.1 Modelo de entidade-relacionamento.....	70
2.10.2 Sistema Gerenciador de Banco de Dados.....	71
2.10.2.1 MySQL.....	72
2.11 PROGRESSIVE WEB APPS.....	74
2.11.1 Requisitos.....	76
2.12 APLICAÇÕES WEB.....	77
2.12.1 Construção.....	78
2.12.1.1 HTML.....	79
2.12.1.2 CSS.....	79
2.12.1.3 JavaScript.....	80
2.12.1.4 PHP.....	81
2.12.1.5 <i>Frameworks</i>	82
2.12.1.5.1 Lumen.....	82
2.12.1.5.1.1 Eloquent ORM.....	83
2.12.1.5.1.2 <i>Migrations</i>	83
2.12.1.5.2 Express.....	84
2.12.1.5.3 VueJS.....	84
2.13 RESTFUL WEBSERVICES.....	85
2.14 AMBIENTE.....	86
2.14.1 MySQL Workbench.....	86
2.14.2 StarUML.....	88
2.14.3 Visual Studio Code.....	89
3 MATERIAL E MÉTODOS.....	91
3.1 PESQUISA BIBLIOGRÁFICA.....	91
3.2 IDENTIFICAÇÃO DE STAKEHOLDERS.....	91
3.3 ANÁLISE DE REQUISITOS.....	91
3.4 PLANEJAMENTO.....	92
3.5 MODELAGEM.....	92
3.5.1 Diagrama de Casos de Uso.....	93
3.5.2 Diagrama de Classes.....	93
3.5.3 Diagrama de Sequência.....	93
3.6 CONSTRUÇÃO.....	93
3.6.1 Documentação dos Casos de Teste.....	94

3.6.2 Codificação.....	94
3.6.3 Execução dos Casos de Teste.....	95
3.7 ENTREGA.....	95
4 RESULTADOS E DISCUSSÕES.....	96
4.1 PESQUISA.....	96
4.2 IDENTIFICAÇÃO DOS STAKEHOLDERS.....	96
4.3 ANÁLISE DE REQUISITOS.....	97
4.4 PLANEJAMENTO.....	99
4.5 ARQUITETURA, LINGUAGENS E FRAMEWORKS.....	99
4.6 MODELAGEM.....	99
4.6.1 Diagrama de Caso de Uso.....	99
4.6.2 Diagrama de Classes.....	123
4.6.3 Diagrama de Atividade.....	125
4.7 CONSTRUÇÃO.....	127
4.7.1 Banco de Dados.....	127
4.7.2 Codificação.....	129
4.7.2.1 Menu.....	130
4.7.2.2 Cadastro.....	132
4.7.2.3 Login.....	132
4.7.2.4 Meu Perfil.....	133
4.7.2.5 Imagens da página (produtor).....	137
4.7.2.6 Meus produtos.....	138
4.7.2.7 Anexar imagens ao produto.....	139
4.7.2.8 Meus endereços.....	140
4.7.2.9 Busca.....	142
4.7.2.10 Início.....	146
4.7.2.11 Ver no mapa.....	148
4.7.2.12 Ver detalhes de produtor.....	149
4.7.2.13 Ver detalhes de produto.....	150
4.7.2.14 Manter canais de contato.....	151
4.7.2.15 Meus interesses.....	152
4.7.2.16 Dar sugestão ou <i>feedback</i>	153
4.8 DIAGRAMA DE COMPONENTES.....	154

5 CONCLUSÃO.....	159
5.1 QUANTO A PESQUISA BIBLIOGRÁFICA.....	159
5.2 QUANTO AO ESTADO ATUAL.....	160
5.3 QUANTO AOS RESULTADOS OBTIDOS.....	160
5.4 TRABALHOS FUTUROS.....	162
REFERÊNCIAS.....	164

1 INTRODUÇÃO

O cotidiano cada vez mais acelerado que vem sendo levado, principalmente em grandes centros, cria a necessidade de otimização do tempo, a fim de criar espaço para a realização de mais atividades ao longo do dia. Implicando, inclusive na alimentação, onde o preparo e apreciação de refeições vem sendo substituído por refeições rápidas e, em sua maioria, baseada em alimentos industrializados e altamente processados. Contribuindo, junto de outros hábitos, para a redução da qualidade de vida. Tomando ciência dos impactos negativos desse estilo de vida, as pessoas vêm buscando minimizá-lo, através da adoção de hábitos alimentares mais saudáveis, por exemplo.

Isso inclui a busca por refeições compostas de alimentos menos processados, cultivados com menos agrotóxicos e mais frescos, os quais são frequentemente encontrados em pequenos produtores locais. Há uma tendência alimentar emergente, que incorpora esses hábitos, baseando-se na busca por uma alimentação mais saudável e sustentável, consequentemente sugerindo um modelo baseado no consumo de alimentos locais, denominada locavorismo.

O locavorismo opõe-se às cadeias produtivas globalizadas de alimentos e busca o estabelecimento de um的习惯 alimentar mais natural e menos prejudicial ao meio ambiente, sugerindo que o consumidor conheça as origens por trás do alimento que está adquirindo e, consequentemente, busque adquirir diretamente do pequeno produtor local. Os impactos positivos dessa proposta são o fortalecimento e valorização da agricultura familiar e da economia local. Além disso, reduz-se os impactos ambientais causados por fazendas de monocultura e criação de animais, além de dispensar o transporte por longas distâncias.

Uma dos principais desafios para viabilização e popularização do locavorismo é a dificuldade do consumidor conhecer e ter acesso a produtores locais. Atualmente, as formas mais comuns de comercialização de alimentos locais são através de feiras ou do fornecimento para lojas e supermercados. Para o produtor, o problema da comercialização em feiras é que o comércio sofre limitação de espaço físico e também temporal, uma vez que as mesmas não acontecem diariamente, por exemplo. Tratando-se do fornecimento para lojas e supermercados, a margem de lucro dos produtores acaba sendo baixa e até mesmo injusta, se comparado ao valor do produto pago pelo consumidor final e a margem de lucro do mercado ou loja dos quais são fornecedores.

Tendo em vista a crescente tendência da computação móvel e de desenvolvimento de aplicativos para as mais distintas finalidades, bem como a popularização dos dispositivos móveis, a existência de um aplicativo em que pequenos produtores locais de alimentos pudessem manter um perfil, fornecendo informações sobre si e seus produtos, para serem apresentados em lista aos consumidores interessados nas categorias de produtos fornecidos, permitiria o maior alcance de vendas em potencial, contribuindo para a viabilização do locavorismo e garantindo assim os diversos benefícios socioambientais citados acima. Dessa forma, que características um protótipo de aplicativo deveria ter para permitir a busca e divulgação de produtores de orgânicos, artesanais e coloniais?

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Desenvolver um protótipo de aplicativo que possibilite a busca e divulgação de produtores locais de alimentos.

1.1.2 Objetivos Específicos

- Revisar bibliografia acerca do tema abordado e especificações técnicas relacionadas à construção do protótipo;
- Realizar levantamento de requisitos do projeto a ser desenvolvido;
- Com base nos requisitos, modelar o protótipo;
- Implementar o modelo, desenvolvendo um protótipo de *Progressive Web App*;
- Disponibilizar protótipo desenvolvido para testes.

2 REFERENCIAL TEÓRICO

2.1 LOCAVORISMO

Segundo Rudy (2012), locavorismo é um movimento de ativismo alimentar que defende o consumo baseado em alimentos locais e produzidos de forma sustentável, como melhor modelo para a saúde humana e do meio ambiente. O movimento une pessoas que, infelizes com alimentos industrializados, originários de fazendas de monocultura, cultivados com agrotóxicos, que são oferecidos atualmente em supermercados, buscam uma forma diferente de se alimentar. A palavra “locavorismo” deriva do termo “*locavore*”, cunhado por Jessica Pretience em 2006, que designa a pessoa interessada no consumo de alimentos locais. Em 2007, *locavore* foi escolhida pelo *New Oxford American Dictionary* como palavra do ano e desde então, o movimento e suas ideias centrais passaram a ganhar popularidade nos Estados Unidos.

Algumas definições de locavorismo sugerem uma alimentação baseada apenas em alimentos produzidos no raio de até 100 milhas, equivalente a aproximadamente 160,93 km, outras sugerem o consumo de alimentos produzidos somente dentro do estado ou região. Apesar dessas definições não estarem incorretas, a questão central vai além de simples limites geográficos, trata principalmente sobre o desenvolvimento de uma relação alimentar diferente, onde as origens do alimento são conhecidas, isto é, a fazenda e o agricultor que o produziu. Atualmente, nos EUA, ser um locavorista é também conhecer os métodos e produtos envolvidos no processo produtivo e o impacto ambiental causado durante a produção do alimento. Implicitamente, sugere-se que sejam consumidos mais vegetais de época e também que se reduza o consumo de carne, tendo em vista os impactos ambientais da indústria agroalimentar. (RUDY, 2012).

Azevedo (2015) apresenta que, na percepção do locavorista, o alimento local é mais saboroso e fresco, sua produção favorece a agricultura familiar e promove a agricultura urbana, sistemas agroalimentares sustentáveis e o bem-estar animal, além de estimular a economia local através da venda direta ao consumidor. Um estudo realizado por Painter (2008, apud AZEVEDO, 2015) realizado com *chefs* e donos de restaurante, indica novamente a visão do alimento local como mais fresco e superior em qualidade, além de criar um diferencial para o negócio e apoiar agricultores locais.

2.1.1 Alimento Local

Não há consenso sobre a definição de alimento local. Diferentes estudos sugerem limites de distância entre o local de produção e consumo a fim de conceder ao alimento a característica de “local”, entretanto, não apresentam consenso entre si. Outros, consideram limites geográficos como estados, regiões, ou até mesmo um conjunto de estados. Uma definição aparentemente mais adequada é apresentada pelo Departamento de Agricultura dos Estados Unidos (USDA), ao assumir mercados alimentares locais e regionais como aqueles onde o produto é adquirido diretamente do produtor, relacionando-o, frequentemente a pequenas propriedades rurais localizadas próximo a regiões metropolitanas (JOHNSON; AUSSENBERG; COWAN, 2013; MARTINEZ et al.; 2010, apud AZEVEDO, 2015).

Thompson, Harper e Kraus (2008), por sua vez, conceituam alimento local tendo em vista a forma como a produção, processamento, transporte e consumo afetam a saúde e o meio ambiente. Para os autores, a proximidade o local de produção do alimento indica uma maior probabilidade de superioridade qualitativa, além de um processo de transporte com menor uso de energia e emissão de poluentes, características certamente buscadas por locavoreiros ao consumir alimentos locais. Essa é uma das partes da definição de alimento local, a outra considera aspectos de sustentabilidade aplicadas no processo produtivo do alimento como capazes caracterizá-los “locais” em relação a alimentos de origens convencionais. Métodos sustentáveis incluem a eliminação ou redução do uso de fertilizantes sintéticos ou derivados do petróleo, bem como de práticas que prejudiquem o solo, provoquem incêndios ou poluam o ar e a água. Para alguns consumidores, a sustentabilidade está também ligada a práticas trabalhistas justas e ao bem-estar animal. Por fim, estendem o conceito de produtor local também àqueles responsáveis pelo processamento do alimento, mas que incorporam essas noções ao processo produtivo, preocupando-se com a história por trás da comida.

Para Thompson, Harper e Kraus (2008), além dos benefícios ambientais associados à adoção de práticas sustentáveis na produção de alimentos locais, evitar o uso de pesticidas e demais químicos podem beneficiar a saúde dos consumidores. Halweil (2003) apresenta outros benefícios, como a redução de queima de combustíveis fósseis (pesquisas apontam que uma refeição preparada com ingredientes importados pode facilmente consumir energia e emitir gases de efeito-estufa quatro vezes mais do que uma refeição preparada com ingredientes locais); preservação propriedades agrícolas e

agricultores locais (agricultores incluídos no comércio local estão menos propensos à extinção); sabor superior (uma das razões pelas quais o locavorismo atraiu a atenção de chefs, críticos de comidas e demais apreciadores); redução de riscos de segurança alimentar (viajar por longas distâncias e passar por inúmeras mãos expõe o alimento a maiores chances de contaminação).

Os empecilhos identificados para a expansão do locavorismo incluem:

[...] restrições de produção para pequenas propriedades; falta de sistemas de distribuição para levar os alimentos locais para os mercados convencionais; pesquisa limitada; falta de treinamento e informação do consumidor para promover a comercialização; e as incertezas relacionadas aos requisitos padronizados de segurança sanitária das agências reguladoras que podem afetar a produção local de alimentos (JOHNSON; AUSSENBERG; COWAN, 2013; MARTINEZ et al., 2010, apud AZEVEDO, 2015, p. 84).

Nesse sentido, medidas como a criação de canais de comercialização alternativos, apoio institucional, como a criação de programas governamentais ou políticas de subsídios econômicos para agricultores, vêm contribuindo para que produtores locais se mantenham no mercado (JOHNSON, AUSSENBERG; COWAN , 2013 apud AZEVEDO, 2015).

2.2 SOFTWARE E SISTEMAS DE INFORMAÇÃO

Basicamente, podemos definir sistema como um conjunto de elementos inter-relacionados com um objetivo comum. Em uma definição mais elaborada, sistema seria um conjunto de elementos interconectados harmonicamente, de modo a formar um todo organizado. Um sistema artificial, isto é, um sistema produzido, que não existe naturalmente, é intencionalmente criado como um conjunto de partes, elementos ou componentes inter-relacionados (os subsistemas) que visa a realização de determinado objetivo, no ambiente em que está inserido. Para alcançar os objetivos do sistema, cada subsistema tem seu objetivo próprio e se acha conectado a outro subsistema, do qual recebe os insumos ou entradas (SBROCCO e MACEDO, 2012).

Um sistema de informação, por sua vez, é um sistema artificial que coleta, processa, armazena, analisa e dissemina informação para um propósito específico. O propósito dos sistemas de informação tem sido definido como obter a informação correta para as pessoas certas, no tempo certo, na quantidade certa e no formato certo. Um sistema de informação não necessariamente é um sistema computadorizado, embora a maioria o seja. Por isso, frequentemente o termo “sistema de informação” é usado como

sinônimo de “sistema de informação baseado em computador” (RAINER JR. e CEGIELSKI, 2016).

Ainda, de acordo com Rainer Jr. e Cegielski (2016), um sistema de informação baseado em computador (SIBC), é um sistema de informação que usa a tecnologia de computador para realizar algumas ou todas as tarefas pretendidas pelo sistema. Sendo composto por:

- Hardware: dispositivos físicos, responsáveis pela recepção de dados de entrada (teclado, mouse), processamento (processador) e exibição (monitor, impressora);
- Software: um programa ou conjunto de programas que definem como o processamento de dados deverá ocorrer;
- Banco de dados: conjunto de arquivos ou tabelas responsável pelo armazenamento de dados;
- Rede: sistema de conexão (com ou sem fio) que permitem aos computadores compartilhar recursos;
- Procedimentos: instruções sobre como combinar todos os componentes para processar informações e gerar a saída desejada;
- Pessoas (ou peopleware): indivíduos que utilizam o hardware e software.

Os recursos mais importantes de SIBCs, segundo Rainer Jr. e Cegielski (2016) são: realizar cálculos numéricos de alta velocidade e alto volume; fornecer comunicação e colaboração rápidas e precisas dentro da organização e entre organizações; armazenar enormes quantidades de informação em um espaço fácil de acessar, ainda que pequeno; permitir acesso rápido e barato a enormes quantidades de informação em todo o mundo; interpretar grandes quantidades de dados de modo rápido e eficiente; automatizar tanto processos de negócio semiautomáticos quanto tarefas manuais. A parte do sistema de informação projetada para permitir a realização de tarefas como as citadas é o software.

Para Carvalho e Lorena (2017), um software é “nada mais que uma sequência de passos ou instruções descritos por um algoritmo, que, quando executados, fazem com que o computador realize uma tarefa”. O algoritmo é uma sequência de instruções, escritas com uma linguagem de programação, que coordenam o hardware para a execução de um determinado conjunto de operações para obtenção do resultado pretendido. O funcionamento de um software pode ser resumido, de forma básica, no seguinte processo: o software recebe dados ou valores como entrada, em seguida realiza

o conjunto de operações descritos pelo algoritmo, e então retornam um resultado que pode ser novos dados ou uma sequência de ações.

Pressman (2016) classifica softwares em sete grandes categorias, sendo elas:

- Software de sistema: Conjunto de programas feito para atender a outros programas, por exemplo: sistemas operacionais, drivers, compiladores, gerenciadores de arquivos.
- Software de aplicação: Programas independentes criados para solucionar uma necessidade específica de negócio. A exemplo: processamento de dados para facilitar tomadas de decisão ou operações comerciais.
- Software de engenharia/científico: são programas utilizados no suporte a atividades científicas ou de engenharia, através do fornecimento de um modelo de simulação, ou realização de grande volume de cálculos complexos que levariam muito mais tempo caso fossem executados por um humano, por exemplo.
- Software embarcado: é um programa residente em um produto ou sistema, que implementa funções limitadas a fim de permitir o controle ou utilização de recursos de seu hospedeiro. Exemplos: controle do painel de um forno microondas, painéis digitais de automóveis.
- Software para linha de produtos: Trata-se de um conjunto de softwares que compartilham características em comum a fim de solucionar problemas específicos, para que possam ser reutilizados na construção de softwares que busquem solucionar tais problemas.
- Aplicações Web/aplicativos móveis: trata-se de uma categoria de software voltada a redes que compreende uma grande variedade de aplicações. Sua principal característica é a mobilidade e a possibilidade de utilização através de navegadores (aplicações web) ou instalação em dispositivos móveis (aplicativos móveis).

Para Pressman (2016), software é, atualmente, uma tecnologia indispensável para negócios, ciência e engenharia e se incorpora a sistemas das mais diversas áreas. É responsável pela viabilização de novas tecnologias, como engenharia genética e nanotecnologia; extensão de tecnologias existentes, como telecomunicações; e mudança radical em tecnologias antigas, como a mídia. Foi também a força motriz por trás da revolução do computador pessoal e, atualmente pode ser acessado no mundo todo através de um navegador web, utilizado em aplicações para dispositivos móveis, além de

permear as mais diversas atividades humanas, como relacionamentos, compras, pesquisas bibliográficas, entre outras.

2.3 ENGENHARIA DE SOFTWARE

Schach (2010) define engenharia de software como uma disciplina cujo objetivo é produzir software isento de falhas, entregue dentro de prazo e orçamento previstos, atendendo também às necessidades do cliente. Além disso, o software deve ser fácil de ser modificado quando as necessidades do usuário mudarem.

Em uma definição formal, de acordo com a IEEE (*[?]* apud. PRESSMAN 2016, p. 15), engenharia de software é: “(1) A aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia ao software. (2) O estudo de abordagens como definido em (1)”. Porém, segundo Pressman (2016), essa abordagem deve também apresentar adaptabilidade e agilidade, para que seja aplicável aos diversos cenários e equipes de desenvolvimento.

De acordo com Sommerville (2011), a engenharia de software preocupa-se com todos os aspectos da produção do software, desde os estágios iniciais de especificação, até a manutenção, incluindo não apenas aspectos técnicos de desenvolvimento, mas também atividades como gerenciamento de projeto e desenvolvimento de ferramentas, métodos e teorias para apoiar a produção de software. Além disso, enquanto uma disciplina de engenharia, tem como objetivo aplicar teorias, métodos e ferramentas adequados ao cenário e suas restrições.

Ainda, segundo o autor, a importância da disciplina tem dois motivos. O primeiro é que cada vez mais, indivíduos e sociedade dependem de softwares avançados e portanto deve-se viabilizar sua construção econômica e rapidamente.. Em segundo, para a maior parte dos sistemas, o maior custo está relacionado a mudanças no software depois que já está sendo utilizado e por isso a aplicação de métodos e técnicas de engenharia de software torna o desenvolvimento mais barato a longo prazo.

Pressman (2016) apresenta engenharia de software como uma tecnologia em camadas, conforme ilustrado pela Figura 1.

Figura 1: Camadas da engenharia de software



Fonte: Pressman (2016, p. 16)

Na base está o foco na qualidade, que, conforme defendido pelo autor, deve ser a base de qualquer disciplina de engenharia. O comprometimento organizacional com a qualidade, aliado a uma cultura de aperfeiçoamento contínuo de processos leva ao desenvolvimento de abordagens cada vez mais eficazes de engenharia de software.

A base da engenharia de software é a camada de processos. O processo de software constitui a base para o controle do gerenciamento de projetos de software e estabelece os contextos para aplicação de métodos, produção de artefatos e estabelecimento de marcos, garantindo a qualidade e gestão apropriada de mudanças. Esse conceito será aprofundado no capítulo seguinte.

Os métodos de software envolvem uma ampla variedade de tarefas, como comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte, por exemplo. São eles que fornecem as informações técnicas para o desenvolvimento de software, isto é, como as tarefas devem ser executadas.

Por fim, as ferramentas fornecem suporte automatizado ou semi automatizado para processos e métodos. Quando integradas de forma que as informações geradas por uma ferramenta alimentam outra, um sistema para o suporte ao desenvolvimento de software é estabelecido, denominado engenharia de software com o auxílio do computador (PRESSMAN, 2016).

2.4 PROCESSO DE SOFTWARE

Processo de software é um roteiro composto por uma sequência de passos previsíveis que deve ser seguido para a criação ordenada dos artefatos necessários para o desenvolvimento de um software. Esses passos podem ser atividades metodológicas, ações de engenharia de software e tarefas, as quais são organizadas dentro de uma

metodologia ou modelo que determina sua relação com o processo e também entre si. A metodologia engloba também um conjunto de atividades de apoio aplicáveis a todo o processo de software (PRESSMAN, 2016).

Segundo Pressman (2016), uma atividade metodológica possui um objetivo amplo. Para atingi-lo, um conjunto de ações de engenharia de software deve ser executado. Cada ação é definida por um conjunto de tarefas, o qual identifica as tarefas de trabalho a ser completadas, os artefatos de software (programas, documentos ou dados) que serão produzidos, os fatores de garantia da qualidade que serão exigidos e os marcos utilizados para indicar o progresso.

Ainda de acordo com o autor, as atividades metodológicas são complementadas por diversas atividades de apoio, que auxiliam a gerenciar e controlar o andamento, a qualidade, as alterações e os riscos. O Quadro 1 apresenta atividades de apoio típicas.

Quadro 1: Atividades de apoio típicas

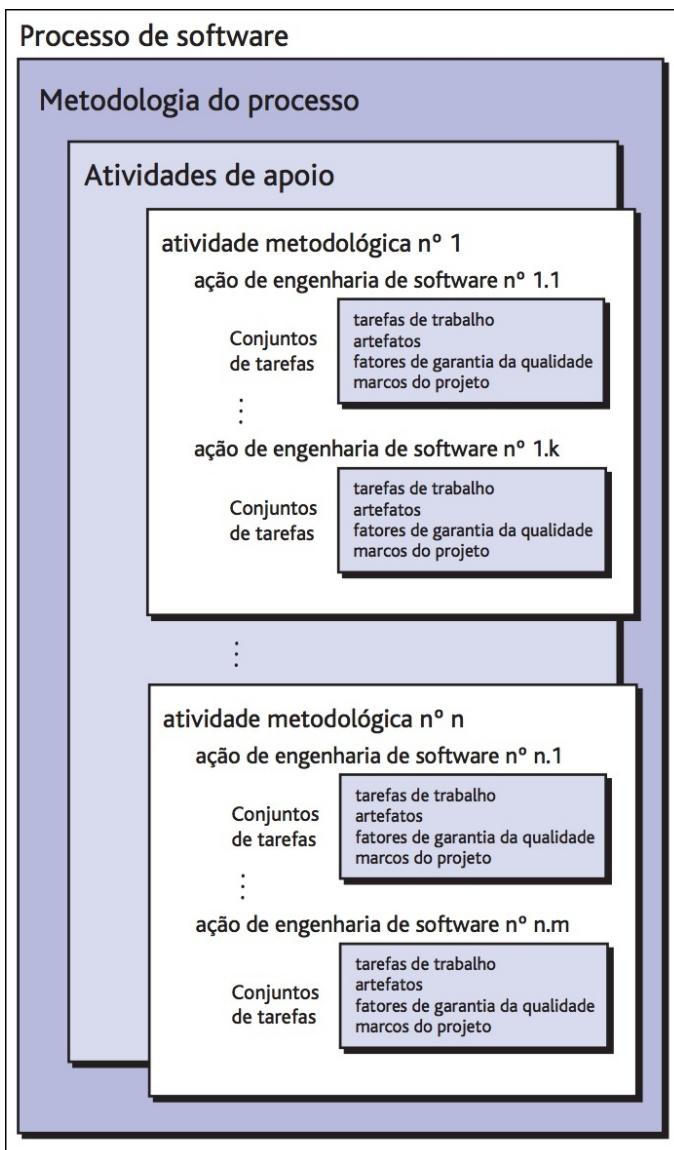
Atividade de apoio	Descrição
Controle e acompanhamento do projeto	Possibilita que a equipe avalie o progresso em relação ao plano do projeto e tome as medidas necessárias para cumprir o cronograma
Administração de riscos	Avalia riscos que possam afetar o resultado ou a qualidade do produto/projeto
Garantia da qualidade de software	Define e conduz as atividades que garantem a qualidade do software
Revisões técnicas	Avaliam artefatos da engenharia de software, tentando identificar e eliminar erros antes que se propaguem para a atividade seguinte
Medição	Define e coleta medidas (do processo, do projeto e do produto). Auxilia na entrega do software de acordo com os requisitos; pode ser usada com as demais atividades (metodológicas e de apoio)
Gerenciamento da configuração de software	Gerencia os efeitos das mudanças ao longo do processo
Gerenciamento da capacidade	Define critérios para a reutilização de artefatos (inclusive componentes de software) e estabelece mecanismos para a

de reutilização	obtenção de componentes reutilizáveis
Preparo e produção de artefatos de software	Engloba as atividades necessárias para criar artefatos como, por exemplo, modelos, documentos, logs, formulários e listas

Fonte: Adaptado de Pressman (2016, p. 18)

A Figura 2 representa como atividades metodológicas e atividades de apoio são organizadas dentro de uma metodologia de processo de software.

Figura 2: Representação da metodologia do processo de software



Fonte: Pressman (2016, p. 32)

De acordo com Pressman (2016), uma metodologia genérica de processo de software é definida por cinco atividades metodológicas:

- Comunicação. O objetivo dessa atividade é entender os objetivos do cliente e dos envolvidos no projeto e identificar requisitos.
- Planejamento. A atividade de planejamento tem como o objetivo o desenvolvimento de um “mapa” para o projeto de software - denominado plano de projeto de software. O plano de projeto de software define o trabalho de engenharia de software, descrevendo tarefas técnicas, riscos de projeto, recursos necessários, produtos resultantes e o cronograma de trabalho.
- Modelagem. Nesta etapa, são desenvolvidos modelos que auxiliam a compreender melhor as necessidades do software e do projeto que irá atendê-las. Os modelos gerados fornecem uma visão do todo, apresentando características como arquitetura e funcionalidades desejadas, por exemplo.
- Construção. A atividade de construção combina a geração de código para a construção do modelo, aliada a testes para a identificação de erros na codificação.
- Entrega. Por fim, o software ou um incremento parcial é entregue ao cliente, que avalia o produto e fornece um feedback.

Independente do porte do software a ser desenvolvido, essas atividades permanecerão as mesmas. Nesse sentido, é importante destacar que em engenharia de software, uma metodologia de processo não é uma prescrição rígida de como desenvolver um software, muito pelo contrário: trata-se de uma abordagem flexível que permite a escolha das ações, tarefas e atividades de apoio adequadas às especificações de cada projeto e equipe. Isso significa dizer que, apesar das atividades metodológicas propostas serem as mesmas para todos os projetos, os detalhes do processo podem e devem ser adaptados às particularidades de cada cenário. (PRESSMAN, 2016).

Outro aspecto importante do processo de software é o chamado fluxo de processo. Um fluxo de processo descreve como são organizadas as atividades metodológicas, bem como a sequência e os momentos em que as ações e tarefas ocorrem dentro de cada atividade. Um fluxo de processo pode ser linear, onde cada uma das atividades metodológicas é executada em sequência, do início ao fim; iterativo, onde uma ou mais atividades se repetem antes de seguir para a próxima; evolucionário, onde o processo linear é executado de forma cílica, incrementando o software a cada entrega; ou

paralelo, em que uma ou mais atividades são executadas paralelamente (PRESSMAN, 2016).

2.5 MODELOS DE PROCESSO DE SOFTWARE

Segundo Pressman (2016), um modelo de processo de software¹ fornece uma sequência de passos para a realização de um trabalho de engenharia de software disciplinado. Para isso, cada modelo acomoda atividades metodológicas abordadas com diferentes ênfases, invocadas em um fluxo de processo definido. Além disso, define também as ações e tarefas, o grau de iteração, os artefatos e a organização do trabalho a ser feito.

Segundo Sommerville (2011), os processos de software podem ainda ser classificados como dirigidos² ou ágeis. Em um processo de software dirigido, todas as atividades são planejadas com antecedência, e o progresso é avaliado por comparação com o planejamento inicial. Um processo ágil, o planejamento é gradativo, tornando o processo mais flexível e adaptável a mudanças de requisitos.

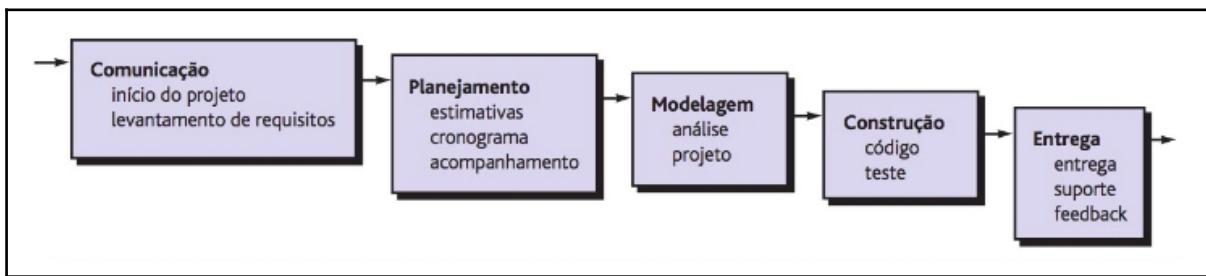
Ainda de acordo com Sommerville (2011), um modelo de processo de software não é uma prescrição definitiva de uma metodologia, mas sim uma abstração, que pode ser adaptada e combinada com outros modelos de processo, a fim de determinar um processo específico adequado às particularidades de cada projeto.

2.5.1 Modelo Cascata

De acordo com Pressman (2016), o modelo cascata, também chamado de “ciclo de vida clássico” é o paradigma mais antigo da engenharia de software. O modelo acomoda as atividades de comunicação, planejamento, modelagem, construção e entrega, em um fluxo linear, conforme representado na Figura 3.

1 Também chamados de “ciclo de vida”, conforme Wazlawick (2013).
 2 Também denominado prescritivo, ou tradicional (PRESSMAN, 2016)

Figura 3: Ciclo de vida clássico



Fonte: Pressman (2016, p. 42)

Pode ser uma abordagem adequada quando têm-se requisitos estáveis e bem definidos, porém, a aplicação do modelo frequentemente enfrenta alguns problemas. Os principais problemas, destacados por Pressman (2016) e Ellis (2010, apud WAZLAWICK, 2013, p. 26) são:

- Dificuldade de definir requisitos completos antes da codificação;
- Dificuldade para identificar todos os requisitos do cliente, que provavelmente só serão notados depois da entrega do produto;
- Projetos reais raramente seguem exatamente o fluxo estabelecido pelo processo, podendo provocar confusões no progresso do projeto, devido a mudanças;
- Falta de flexibilidade de requisitos, sendo difícil voltar atrás para corrigir requisitos mal definidos;
- Não produz resultados tangíveis até a etapa de codificação, exceto para pessoas familiarizadas com artefatos de documentação;
- Uma versão operacional do programa só ficará disponível ao final do projeto, aumentando o risco de impacto por erros não identificados durante o processo de desenvolvimento.

2.5.2 Modelos de Processo Evolucionários

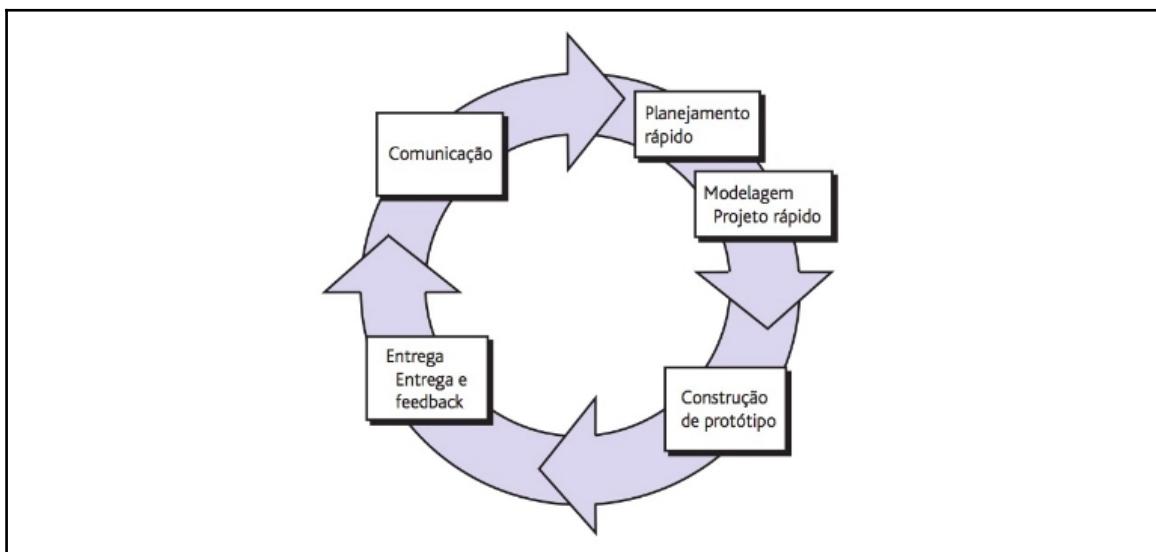
Um modelo de processo evolucionário adota o fluxo iterativo e apresenta características que permitem o desenvolvimento de um produto cada vez mais completo a cada iteração. Trata-se de um modelo de processo projetado especificamente para o desenvolvimento de produtos que crescem e mudam. A aplicação pode ser adequada para cenários onde as regras de negócio e requisitos podem sofrer alterações, quando não se tem tempo hábil para o desenvolvimento e entrega de um produto completo ou quando requisitos não estão bem definidos, por exemplo (PRESSMAN, 2016).

2.5.2.1 Prototipação

O paradigma da prototipação consiste, basicamente, na construção de uma “demonstração” do software ou de parte dele (geralmente construída rapidamente, incorporando funcionalidades básicas), para que seja avaliado pelos envolvidos no projeto a fim de auxiliar na compreensão do que exatamente deve ser desenvolvido. É uma abordagem recomendada para casos em que se tem uma ideia geral do que deve ser construído, porém os requisitos não são suficientes ou não estão bem definidos (PRESSMAN, 2016).

O modelo adota um fluxo iterativo, incorporando as etapas conforme representado na Figura 4. O ponto de partida é a atividade de comunicação, na qual são definidos os objetivos gerais do software, identificados os requisitos e quais áreas necessitam de uma definição mais ampla. Com isso, uma iteração de prototipação é planejada, na qual desenvolve-se um modelo do protótipo (normalmente priorizando aspectos visuais do software), que é construído e na sequência deve ser avaliado e receber feedback dos envolvidos. Com isso, é possível refinar os requisitos e ter maior clareza das necessidades que o software deve atender. A próxima iteração dedica-se na melhoria do protótipo e assim sucessivamente, obtendo a cada iteração um produto mais próximo do atendimento dos requisitos, até que o objetivo do processo seja cumprido (PRESSMAN, 2016).

Figura 4: Paradigma de prototipação



Fonte: Pressman (2016, p. 45)

De acordo com Wazlawick (2013), o modelo de processo de prototipação distingue-se em duas técnicas: *throw-away* e *cornerstone*.

A abordagem *throw-away*, consiste na construção de protótipos usados unicamente com o objetivo de estudar aspectos do sistema, entender melhor seus requisitos e reduzir riscos, sendo descartado após o cumprimento de sua finalidade (CRINNION, 1991, apud WAZLAWICK, 2013, p. 37).

Na abordagem *cornerstone*, o protótipo é utilizado com as mesmas finalidades da abordagem *throw-away*, porém, ao invés de ser descartado, o protótipo será parte do sistema final, sendo evoluído até que se torne um sistema “entregável”. Por essa característica, requer um planejamento mais cuidadoso, visando evitar que um protótipo com erros componha o produto final. O modelo de prototipação evolucionária se baseia nessa técnica (BUDDLE et al., 1992, apud. WAZLAWICK, 2013, p. 37).

2.5.2.2 Modelo orientado a cronograma

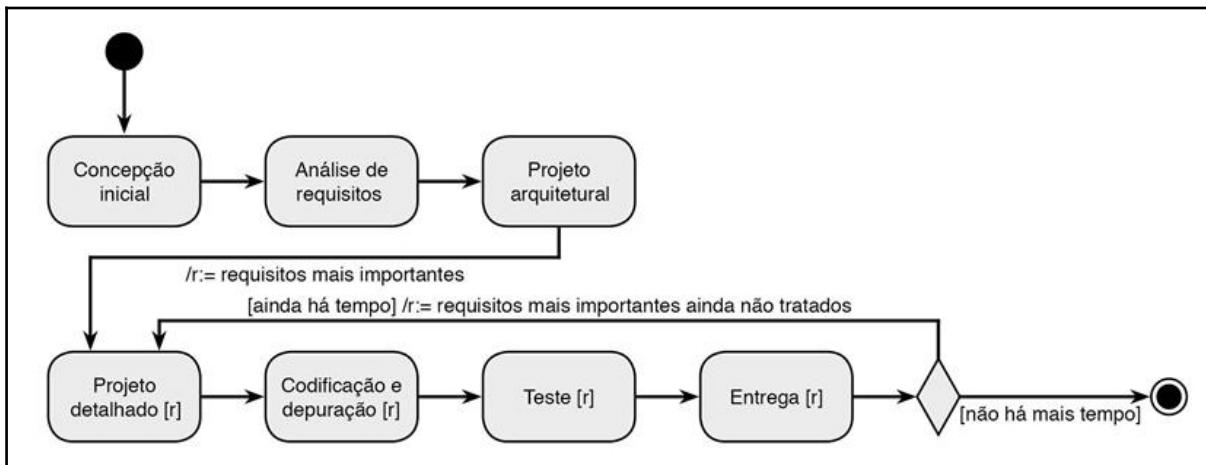
De acordo com Wazlawick (2013), o modelo de processo orientado a cronograma é uma variação melhor estruturada do paradigma de prototipação. De modo geral, o modelo adota um fluxo iterativo, para o atendimento gradual dos requisitos, ordenados por prioridade, até que todos os requisitos sejam atendidos ou o tempo se esgote. Com isso, espera-se que ao final do projeto, mesmo que nem todos os requisitos tenham sido atendidos, ao menos um conjunto de requisitos com maior prioridade foi atendido. Trata-se de uma abordagem adequada, principalmente, para projetos com uma data limite de entrega intransferível.

A Figura 5 apresenta o diagrama de atividades UML para o modelo. Antes de iniciar as iterações de construção, é necessário realizar a análise de requisitos, com base na qual será desenvolvido o projeto arquitetural, que organizará os requisitos em unidades funcionais, apresentando também a forma como se interconectam e colaboram. Com o projeto arquitetural completo, é possível iniciar as iterações, que esforçam-se em detalhar e atender os requisitos (WAZLAWICK, 2013).

O processo se encerra quando todos os requisitos forem atendidos, ou o tempo disponível para desenvolvimento se esgote, por isso a importância da priorização. Para que a técnica seja bem sucedida, é importante ter clareza dos requisitos e sua

importância para o cliente e das dependências entre os componentes funcionais, para a correta priorização (WAZLAWICK, 2013).

Figura 5: Modelo de processo orientado a cronograma



Fonte: Wazlawick (2013, p. 38)

Segundo Ellis (2010, apud WAZLAWICK, 2013, p. 37),

Uma das principais vantagens desse modelo é o fato de colocar funcionalidades úteis nas mãos do cliente antes de completar o projeto. Se os estágios forem planejados cuidadosamente, funcionalidades importantes estarão disponíveis muito mais cedo do que com outros ciclos de vida. Além disso, esse modelo provê entregas mais cedo e de forma contínua, o que pode aliviar um pouco a pressão de cronograma colocada na equipe.

2.6 REQUISITOS DE SOFTWARE

Para Sommerville (2011), os requisitos de um sistema refletem as necessidades dos clientes, compreendendo descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento. O processo de descobrir, analisar, documentar e verificar requisitos é chamado engenharia de requisitos.

Sommerville (2011) classifica requisitos em dois grupos: requisitos de usuário e requisitos de sistema. Requisitos de usuário são requisitos de alto nível que expressam os serviços que o sistema deve fornecer e sob quais restrições irá operar, normalmente através de linguagem natural e diagramas. Um requisito de usuário pode gerar um ou mais requisitos de sistema. Requisitos de sistema são versões expandidas dos requisitos de usuário, servem como ponto de partida para o projeto do sistema.

2.6.1 Requisitos Funcionais e Não Funcionais

De acordo com Sommerville (2011), os requisitos de sistema são frequentemente classificados em requisitos funcionais e não funcionais. Requisitos funcionais são declarações de funcionalidades que o sistema deve implementar e regras de negócio, algumas vezes também podem indicar o que o sistema não deve implementar. Requisitos não funcionais, por sua vez são requisitos que não estão diretamente relacionados com os serviços específicos oferecidos pelo sistema, são restrições aos serviços ou funções oferecidos pelo sistema.

Segundo o autor, requisitos funcionais devem apresentar completude (todas as necessidades do usuário devem ser identificadas) e consistência (não devem apresentar contradição). Na prática, é praticamente impossível alcançar essas características na análise de requisitos funcionais, pois as inconsistências tendem a surgir após uma análise mais profunda ou após a entrega total ou parcial do produto.

Ainda de acordo com Sommerville (2011), ao contrário de requisitos funcionais, requisitos não funcionais podem impactar em uma ou mais funcionalidade, por vezes até mesmo inutilizando o sistema. Requisitos não funcionais devem ser mensuráveis, para que seja possível verificar o atendimento ou não do requisito. Podem ser classificados em:

1. Requisitos de produto: determinam características de comportamento do software, como desempenho e confiabilidade, por exemplo;
2. Requisitos organizacionais: resultantes de políticas e procedimentos organizacionais, pode-se citar como exemplo restrições de linguagem de programação e forma de acesso ao sistema;
3. Requisitos externos: são requisitos que derivam de fatores externos, como órgãos reguladores e legislação, por exemplo.

2.6.2 Engenharia de Requisitos

De acordo com Pressman (2016) a engenharia de requisitos é uma ação de engenharia de software que ocorre entre as atividades de comunicação e modelagem. Ela deve ser adaptada às necessidades do projeto, porém abrange, de modo geral sete tarefas distintas, respectivamente descritas a seguir:

1. Concepção. A concepção busca o entendimento básico do problema, a identificação dos interessados na solução e qual sua natureza.

2. Levantamento: busca-se compreender os objetivos do sistema para os envolvidos, estabelecer as metas de negócio, bem como estabelecer métodos de atribuição de prioridades.
3. Elaboração: esta etapa concentra-se no refinamento das informações obtidas nas duas etapas anteriores, buscando desenvolver um modelo de requisitos refinado.
4. Negociação: junto dos interessados no projeto, resolvem-se conflitos entre requisitos e limitações tecnológicas, também nessa etapa os requisitos devem ser ordenados e priorizados.
5. Especificação: na especificação de requisitos, desenvolve-se uma descrição detalhada dos requisitos, que pode ser apresentada em linguagem natural, modelos gráficos, modelos matemáticos ou casos de uso, individualmente ou combinados.
6. Validação: a validação de requisitos examina a especificação para garantir que todos os requisitos de software tenham de forma correta, não ambígua e consistente.
7. Gestão: a gestão de requisitos é um conjunto de atividades que visa a identificação, controle e acompanhamento das mudanças de requisitos

O documento de requisitos de software, também chamado de Especificação de Requisitos de Software, é uma declaração oficial de o que os desenvolvedores do sistema devem implementar. Deve incluir tanto requisitos de usuário quanto requisitos de sistema, descritos separadamente ou não, dependendo da quantidade de requisitos e do padrão adotado. Ainda de acordo com o padrão adotado, por conter mais informações, como arquitetura e modelos de alto nível e aspectos da evolução, por exemplo (SOMMERVILLE, 2011).

2.7 UML

A Linguagem de Modelagem Unificada (em inglês, Unified Modeling Language, UML) é uma linguagem padrão para a descrição e documentação e projetos de software. Pode ser usada para visualização, especificação, documentação e construção de artefatos de um software. (BOOCH e JACOBSEN, 2004, apud PRESSMAN, 2016, p. 869).

De acordo com Bezerra (2015), a UML define elementos visuais que podem ser utilizados na modelagem de sistemas, em especial aqueles que adotam o paradigma de orientação a objetos. Através dos elementos gráficos da linguagem, é possível construir diagramas que representam diversas perspectivas de um sistema.

De acordo com UML.org (2020, tradução minha), a versão 2.0 da linguagem define treze tipos de diagramas, dos quais seis representam a estrutura estática do aplicativo, três representam comportamentos gerais, e quatro representam aspectos de interações.

2.7.1 Diagrama de Caso de Uso

A modelagem de caso de uso é frequentemente utilizada no apoio à elicitação de requisitos. Um caso de uso pode ser considerado um cenário simples que descreve o que o usuário espera de um sistema (SOMMERVILLE, 2011).

De acordo com Bezerra (2015), um caso de uso é definido como a “especificação de uma sequência completa de interações entre um sistema e um ou mais agentes externos a esse sistema”, que corresponde ao relato de uso de uma funcionalidade. O Modelo de Caso de Uso (MCU) de uso abrange a perspectiva externa do sistema, o que significa dizer que não deve revelar a estrutura e comportamento internos. Com a compreensão do MCU, é possível saber quais são as funcionalidades oferecidas pelo sistema e quais os resultados externos produzidos por cada uma delas. O diagrama UML que representa o MCU é o Diagrama de Caso de Uso (DCU).

O DCU é composto por três elementos principais: casos de uso, atores e relacionamentos. Também é possível representar as fronteiras do sistema através de um retângulo. Cada um dos elementos de um DCU é descrito a seguir, de acordo com Bezerra (2015):

2.7.1.1 Caso de uso

Um caso de uso corresponde a uma funcionalidade do sistema. É graficamente representado por uma elipse, com o nome do caso de uso posicionado abaixo ou dentro da figura, conforme representado na Figura 6. A descrição textual de um caso de uso relata as interações entre os elementos externos de um sistema e o sistema em si. Diferentemente da forma de representação gráfica, a UML não define um padrão para a estrutura da descrição de um caso de uso

2.7.1.2 Atores

São agentes externos que interagem com o sistema, normalmente responsáveis pelo início da execução de um caso de uso. Um ator corresponde a um papel em relação ao sistema, por isso é uma boa prática nomear os atores com o papel que desempenham, ao invés de nomear os indivíduos. Esses papéis podem ser descritos por cargos, organizações ou divisões de uma organização, outros sistemas, ou equipamentos com os quais o sistema se comunica. A notação UML para representar atores em um DCU é a figura de um boneco “de palitos”, com seu papel posicionado abaixo da figura, conforme apresentado na Figura 6.

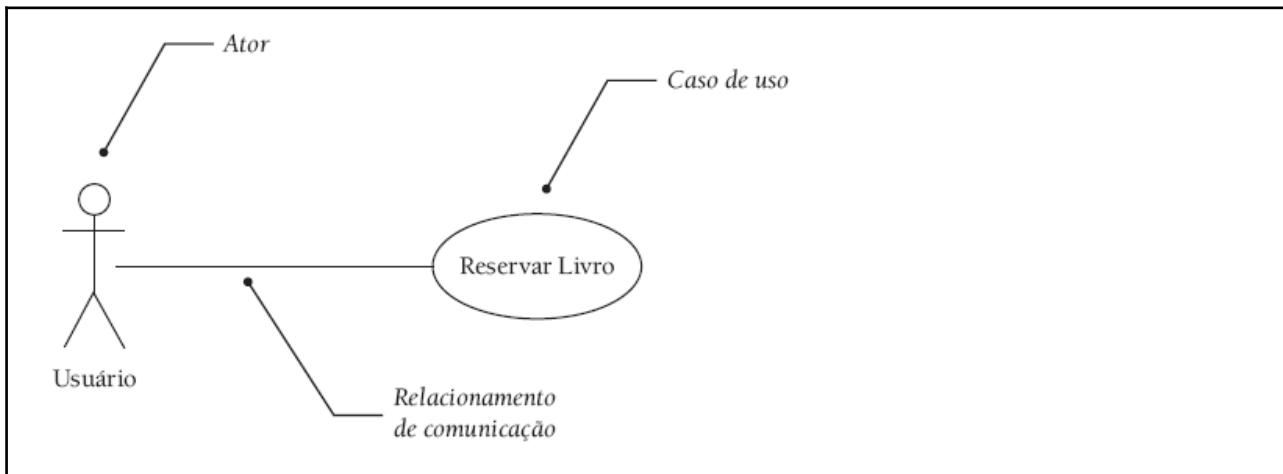
2.7.1.3 Relacionamentos

O terceiro componente de um DCU são os relacionamentos, cuja função é relacionar atores e casos de uso. Um ator deve estar relacionado a um ou mais casos de uso, pode haver também relacionamentos entre casos de uso ou entre atores. Os relacionamentos são classificados em: comunicação, inclusão, extensão e generalização

2.7.1.3.1 Relacionamento de comunicação

Um relacionamento de comunicação ocorre entre um ator e um caso de uso, indicando que o ator interage (troca informações) com o sistema, através do caso de uso que se relaciona. A representação gráfica do relacionamento de comunicação é um segmento de reta ligando o ator e o caso de uso, conforme representado na Figura 6. De acordo com a UML, pode-se também imprimir o sentido da reta, para denotar onde a interação foi iniciada, apesar de ter pouco uso prático e não ser muito usual.

Figura 6: Representação gráfica UML de Ator, Caso de Uso e Relacionamento de comunicação

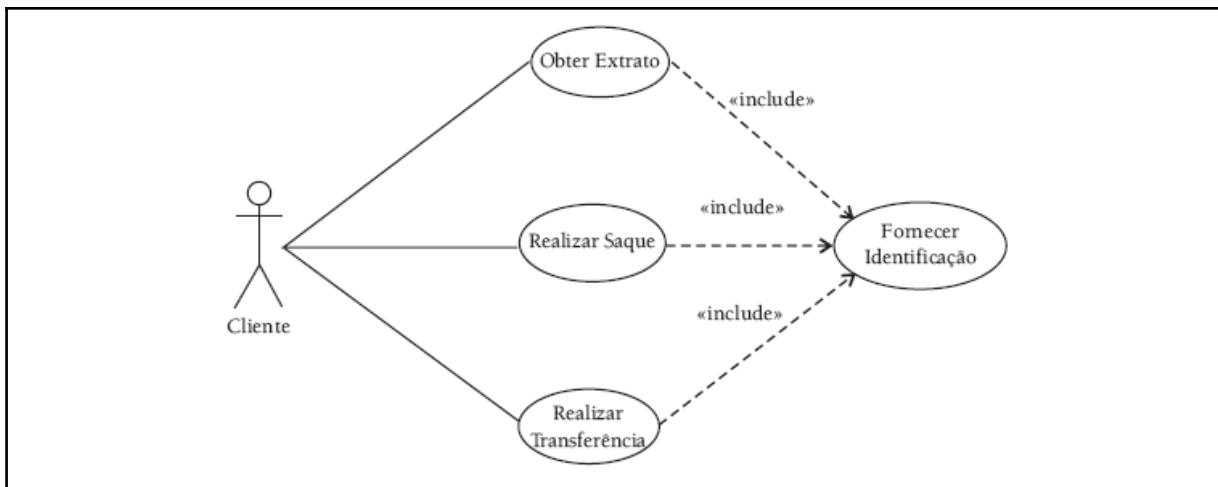


Fonte: Bezerra (2015, p. 70)

2.7.1.3.2 relacionamento de inclusão

O relacionamento de inclusão ocorre somente entre casos de uso. Quando dois ou mais casos de uso incluem uma sequência comum de interações, a sequência comum pode ser descrita em outro caso de uso, que será incluído por cada um dos casos de uso que a compartilham. Isso evita a repetição da descrição da sequência comum, além de tornar a descrição dos casos de uso mais simples e facilitar a manutenção. O relacionamento de inclusão entre um caso de uso inclusor A (aquele que inclui o comportamento) e caso de uso incluso B (aquele cujo comportamento é incluído por outros) é representado por uma seta direcionada de A para B, tendo seu eixo tracejado e rotulado com o estereótipo *include*, demonstrado na Figura 7.

Figura 7: Exemplo de relacionamento de inclusão

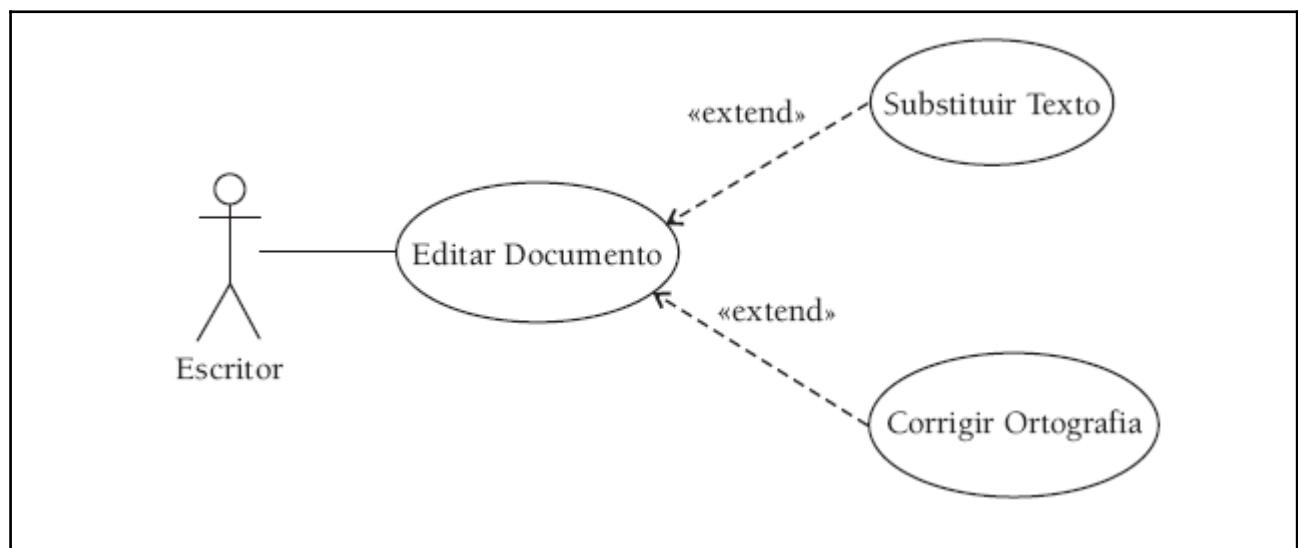


Fonte: Bezerra (2015, p. 71)

2.7.1.3.3 Relacionamento de extensão

Assim como o relacionamento de inclusão, o relacionamento de extensão ocorre somente entre casos de uso. Esse relacionamento é usado para modelar situações em que diferentes casos de uso (extensores) podem ser inseridos em cenários específicos de um caso de uso (estendido). Entretanto, o caso de uso estendido por outros casos de uso deve ser descrito com completude, sem indicar que deve necessariamente haver uma extensão em seu comportamento. A representação gráfica de um relacionamento de extensão, em que um caso de uso A estende um caso de uso B, é representado por uma seta direcionada de A para B, com eixo tracejado e rotulado com o estereótipo *extend*, conforme exemplo da Figura 8.

Figura 8: Exemplo de relacionamento de extensão



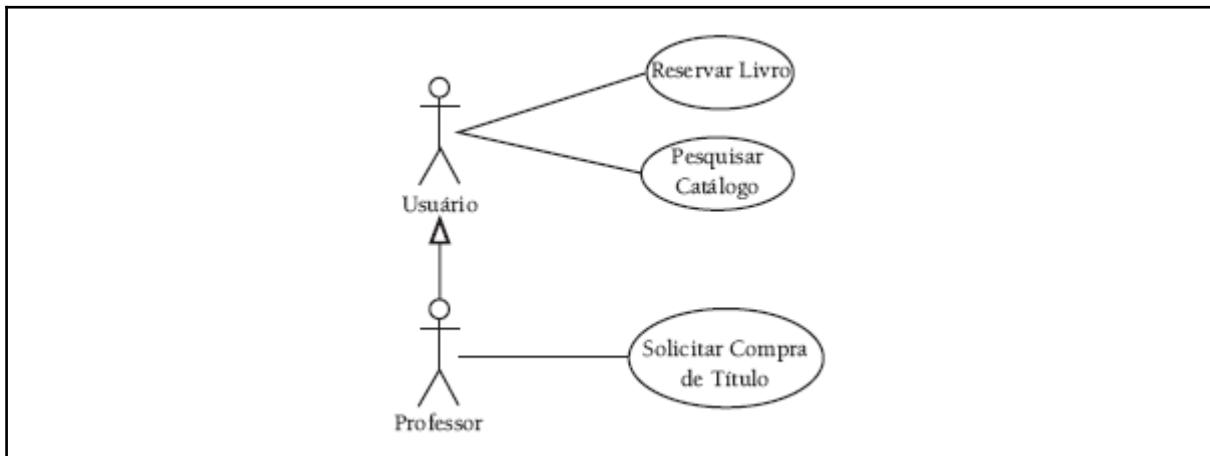
Fonte: Bezerra (2015, p. 72)

2.7.1.3.4 Relacionamento de generalização

O relacionamento de generalização pode existir entre dois casos de uso ou entre dois atores. Quando entre casos de uso, permite que um caso de uso B herde o comportamento e os relacionamentos de A, podendo incluir novos comportamentos e relacionamentos. Entre atores, a generalização permite que o ator B, herdeiro de A, interaja com todos os casos de uso de A, podendo interagir com novos casos de uso, restritos a B. Uma seta com a ponta em formato de triângulo, na direção do caso de uso

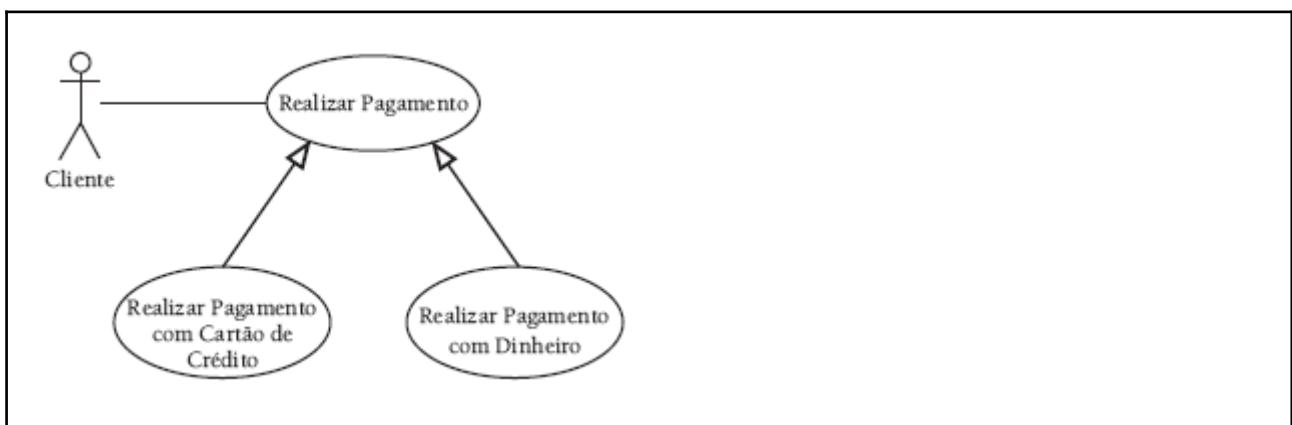
ou ator do qual as características são herdadas, com uma linha sólida compondo o eixo, é o elemento gráfico responsável pela representação desse relacionamento, exemplificado na Figura 9 (generalização entre atores) e Figura 10 (generalização entre casos de uso).

Figura 9: Exemplo de relacionamento de generalização entre atores



Fonte: Bezerra (2015, p. 72)

Figura 10: Exemplo de relacionamento de generalização entre casos de uso

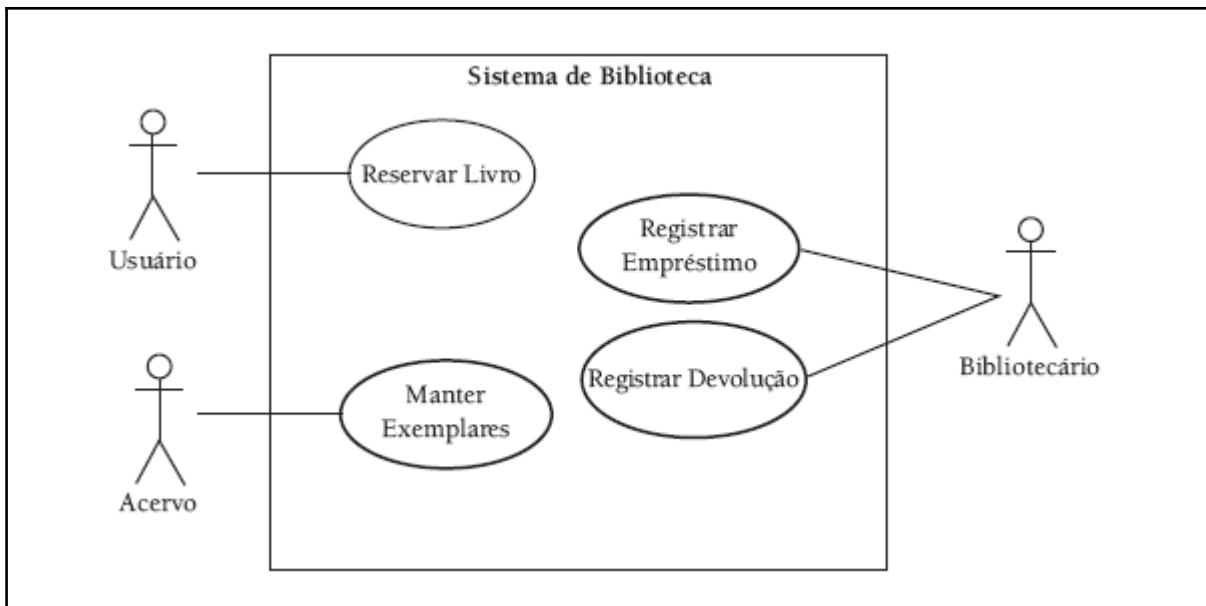


Fonte: Bezerra (2015, p. 72)

2.7.1.4 Fronteira do sistema

A fronteira do sistema é utilizada para enfatizar a divisão entre o interior e exterior do sistema. Para isso, utiliza-se um retângulo no interior do qual são inseridos os casos de uso, os atores devem ser posicionados fora do retângulo, conforme exemplificado na Figura 11.

Figura 11: Exemplo de caso de uso com representação de fronteira de sistema



Fonte: Bezerra (2015, p. 70)

2.7.2 Diagrama de Classes

O modelo de classes é utilizado durante a maior parte do desenvolvimento iterativo e incremental de um Sistema de Software Orientado a Objeto (SSOO) e evoluiu durante as iterações. À medida que as iterações ocorrem o modelo de classes evolui, sendo incrementado com novos detalhes descobertos durante o desenvolvimento. (BEZERRA, 2015).

Conforme Bezerra (2015), o modelo de classes é utilizado durante a maior parte do desenvolvimento iterativo, através do qual evolui à medida que novos detalhes são descobertos e incrementados ao modelo. Durante o processo, o modelo de classes passa por três estágios sucessivos de abstração: análise, especificação e implementação.

O Diagrama de Classes é utilizado na construção do modelo de classes desde o nível de análise até o nível de especificação. No nível de análise, o objetivo do modelo de dados é descrever o problema a ser resolvido pelo software, sem considerar características de solução. Na etapa de especificação, detalhes da solução a ser utilizada são descritos em alto nível de abstração (BEZERRA, 2015). A seguir, são apresentados os elementos de um diagrama de classes UML.

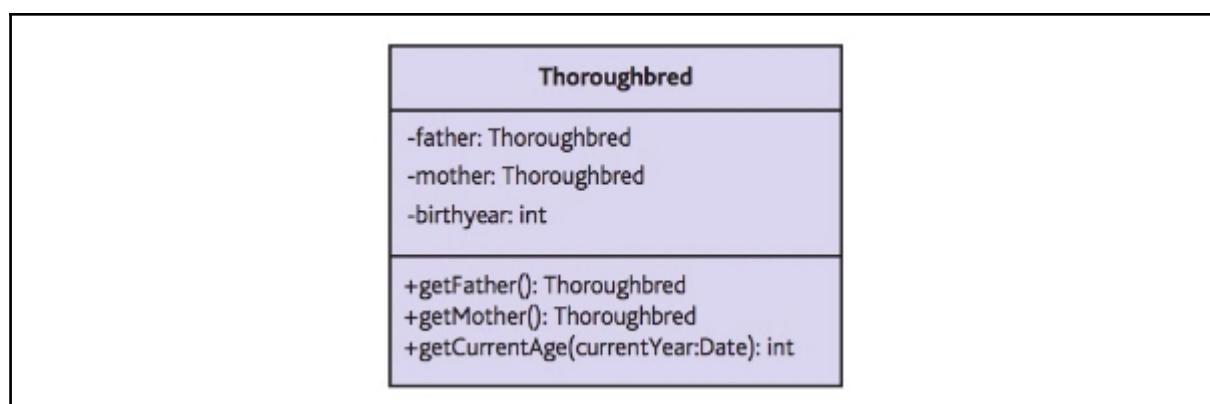
2.7.2.1 Classes

Uma classe é representada por uma “caixa” com, no máximo três compartimentos. O primeiro compartimento (de cima para baixo) é obrigatório e contém o nome da classe. Por convenção o nome deve ser apresentado no singular, e com as palavras que o compõem iniciando com letra maiúscula. Os compartimentos seguintes contém, respectivamente, os atributos e os métodos da classe (BEZERRA, 2015).

De acordo com Pressman (2016), atributos podem ser valores calculados pela classe a partir de suas variáveis de instância, ou valores obtidos dos objetos que a compõem. Cada atributo pode ter um nome, um tipo e um nível de visibilidade, sendo obrigatório somente o nome. O atributo pode ainda ser caracterizado como estático ou de classe, sublinhando um atributo estático. Os métodos de uma classe correspondem às operações que ela pode executar. Também apresentam um nível de visibilidade e nome, junto de seus parâmetros e retornos, com o tipo indicado.

A visibilidade de atributos e métodos é representada pelo sinal –, #, ~ ou +, precedente ao nome do atributo para indicar, respectivamente, as visibilidades *private*, *protected*, *package* ou *public*. O tipo de um atributo, parâmetro ou retorno pode corresponder a um tipo primitivo, ou a um objeto. (PRESSMAN, 2016). A Figura 12 ilustra a representação de uma classe, com atributos e métodos.

Figura 12: Representação de uma Classe em UML



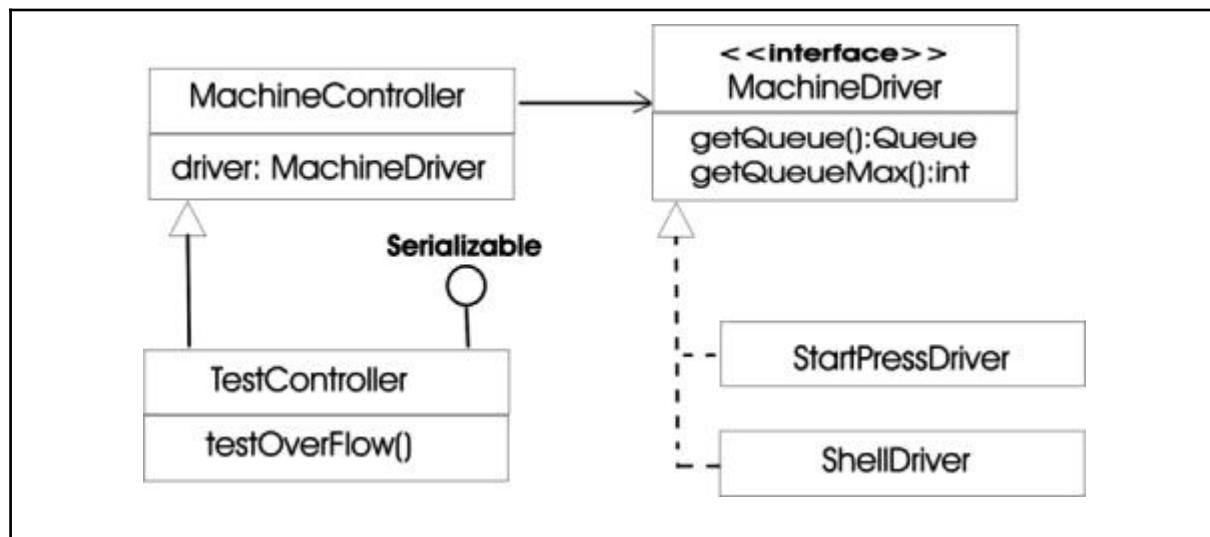
Fonte: Pressman (2016, p. 870)

2.7.2.1.1 Interfaces

De acordo com Booch (2000), “uma interface é uma coleção de operações utilizadas para especificar um serviço de uma classe ou componente”. Uma interface é

representada por um círculo nomeado, ou em sua forma expandida, pode ser representada por uma classe estereotipada com «*interface*», com o objetivo de expor suas propriedades. As duas representações são expressas, respectivamente na Figura 13, pelas interfaces *Serializable* e *MachineDriver*

Figura 13: Duas formas de representação gráfica de interfaces

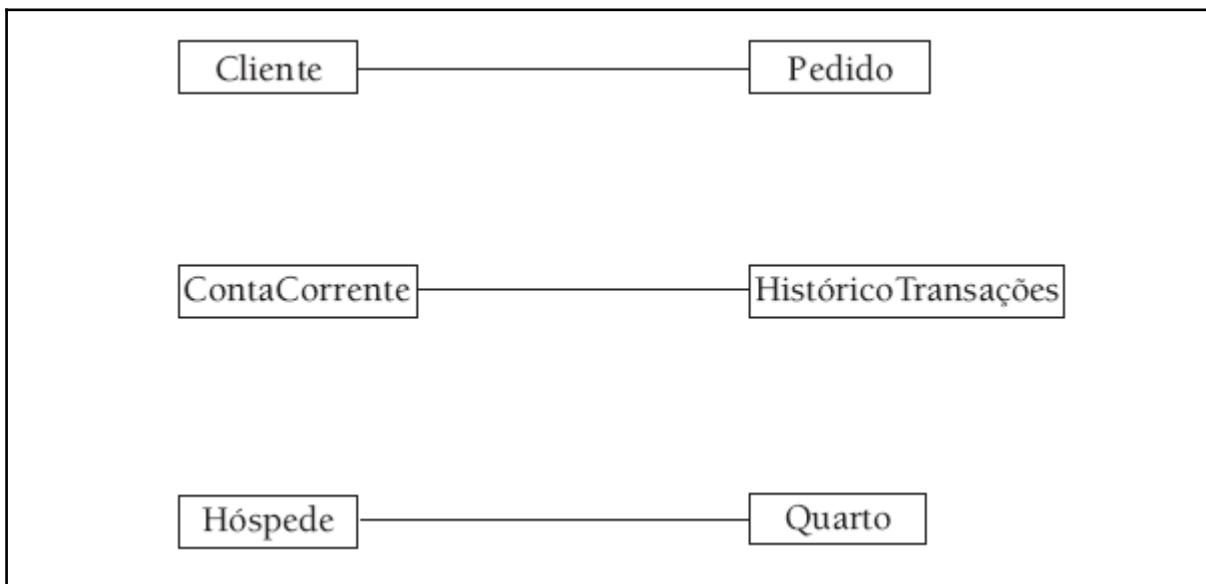


Fonte: Devmedia (2020)

2.7.2.2 Associações

De acordo com Bezerra (2015), Cada ocorrência de uma classe é chamada de objeto ou instância. Durante a execução de um sistema, objetos se relacionam uns com os outros, o que permite que troquem mensagens e colaborem entre si para produzir as funcionalidades do sistema. A associação é o elemento utilizado para indicar um relacionamento entre classes, representado por uma linha ligando as classes às quais pertencem os objetos relacionados, conforme exemplo da Figura 14. Associações possuem diversas características, como multiplicidade, nome, direção de leitura, papéis, tipos de participação e conectividade que serão descritas a seguir.

Figura 14: Exemplos de associações entre classes



Fonte: Bezerra (2015, p. 114)

2.7.2.3 Multiplicidades

Conforme Bezerra (2015), cada associação em um diagrama de classes possui duas multiplicidades, uma em cada extremo da linha que a representa. A multiplicidade de uma associação permite a representação dos limites inferior e superior da quantidade de objetos aos quais o outro objeto pode estar associado. Os símbolos possíveis para representar uma multiplicidade estão descritos no Quadro 2.

Quadro 2: Simbologia para representar multiplicidades

Nome	Simbologia
Apenas um	1
Zero ou muitos	0..*
Um ou Muitos	1..*
Zero ou um	0..1
Intervalo específico	I _i ..I _s

Fonte: Bezerra (2015, p. 115)

Conectividade é o nome dado ao tipo de associação entre duas classes. Ela depende dos símbolos de multiplicidade utilizados na associação (BEZERRA, 2015). O

Quadro 3 relaciona os símbolos de multiplicidade utilizados na associação e a conectividade correspondente.

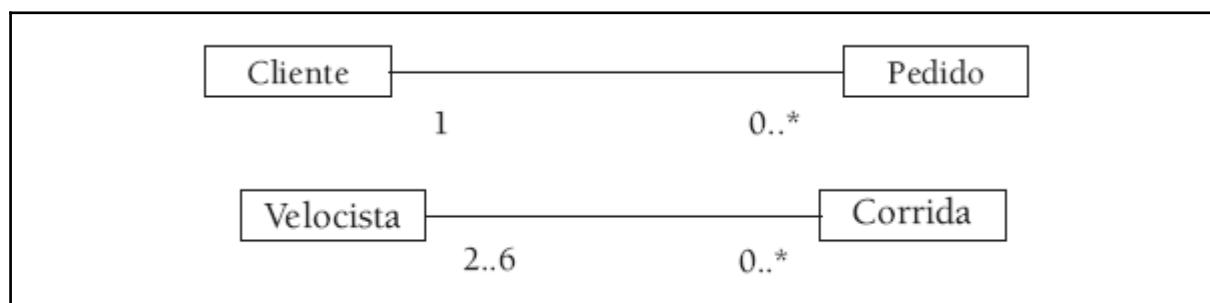
Quadro 3: Conectividades em relação às multiplicidades

Conectividade	Multiplicidade de um extremo	Multiplicidade do outro extremo
Um para um	0..1 ou 1	0..1 ou 1
Um para muitos	0..1 ou 1	* ou 1..* ou 0..*
Muitos para Muitos	* ou 1..* ou 0..*	* ou 1..* ou 0..*

Fonte: Bezerra (2015, p. 116)

A Figura 15 apresenta um exemplo de associação com notação de multiplicidade.

Figura 15: Exemplos de associação com uso de símbolos de multiplicidade



Fonte: Bezerra (2015, p. 115)

2.7.2.4 Participações

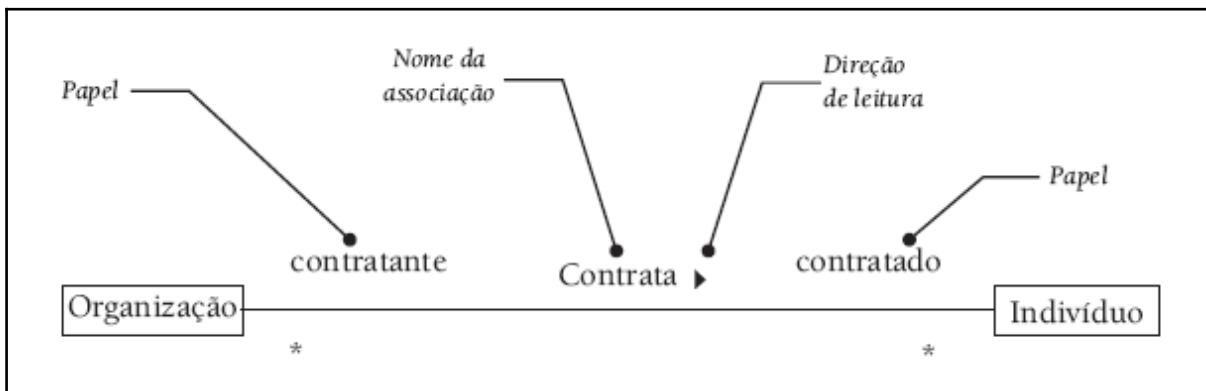
Participação é o nome dado à característica de necessidade ou não da existência de uma associação entre objetos. Uma participação pode ser obrigatória ou opcional, dependendo da multiplicidade da associação. Se o valor mínimo de uma associação é igual a 1, a participação é obrigatória, caso contrário, é opcional (BEZERRA, 2015).

2.7.2.5 Nome de associação, direção de leitura e papéis

De acordo com Bezerra (2015), o nome, direção de leitura e papéis são recursos UML de notação que permitem esclarecer o significado de uma associação. O nome da associação deve ser posicionado no meio da linha da associação e deve fornecer um significado semântico a ela. A direção de leitura é representada por um pequeno triângulo

posicionado próximo ao lado do nome da associação referente à direção que a associação deve ser lida. Por fim, os papéis, posicionados na linha de associação, ao lado do objeto, tem o objetivo de esclarecer a responsabilidade do objeto na associação. Esses recursos são opcionais e devem ser sabiamente empregados para esclarecer a finalidade de uma associação, sua utilização é exemplificada na Figura 16.

Figura 16: Exemplo de utilização dos recursos UML nome da associação, direção de leitura e papéis

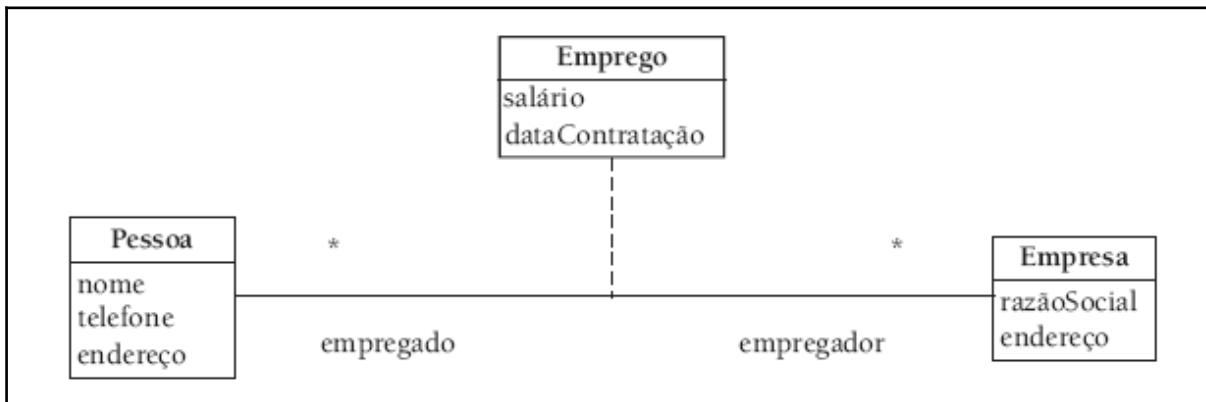


Fonte: Bezerra (2015, p. 118)

2.7.2.6 Classes associativas

Classes associativas são classes ligadas a associações e normalmente aparecem quando é necessário manter informações da associação. É comum que sejam ligadas a associações muitos para muitos, entretanto, podem estar ligadas a associações de qualquer conectividade. Uma classe associativa pode, inclusive, participar de outros relacionamentos. Também pode ser substituída por um classe normal, associada a cada uma das classes associadas na relação que daria origem a uma classe associativa. Na UML, uma classe associativa é representada pela mesma notação utilizada para uma classe comum, porém, liga-se a uma associação através de uma linha tracejada. A Figura 17 apresenta a representação da classe associativa Emprego (BEZERRA, 2015).

Figura 17: Exemplo de representação de uma classe associativa



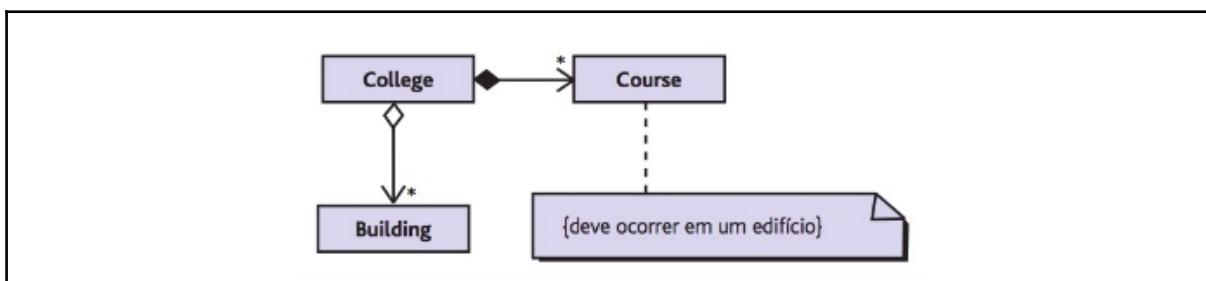
Fonte: Bezerra (2015, p. 119)

2.7.2.7 Agregações e composições

Em um diagrama de classes UML, relações de todo-parte significam que o objeto parte está contido no objeto todo, e o objeto todo contém o objeto parte. Esse tipo de relação é dividido em dois tipos: a agregação e a composição (BEZERRA, 2015).

De acordo com Pressman (2016), tanto a composição quanto a agregação representam relações de todo-parte, entretanto, na composição, os objetos parte não têm papel a desempenhar no sistema de software sem a existência de um objeto todo. A relação de agregação é representada por um losango vazio ao lado da classe todo, na relação de composição, esse losango mantém o posicionamento, porém o losango é preenchido. A Figura 18 apresenta uma relação de agregação entre as classes *College* e *Building*, e uma relação de composição entre as classes *College* e *Course*.

Figura 18: Exemplo de notação de relações de agregação e composição

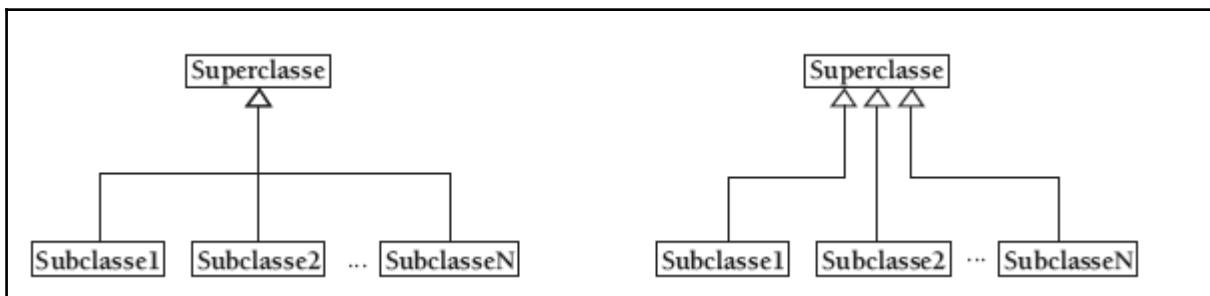


Fonte: Pressman (2016, p. 873)

2.7.2.8 Generalizações e especializações

Conforme Bezerra (2015), relacionamentos generalização/especialização, também chamado de relacionamento de herança, indica uma relação entre classes, onde uma classe A, chamada de subclasse, herda atributos e métodos de uma classe mais genérica B, denominada superclasse. Declara-se então que a classe A é uma especialização de B, e B é uma generalização de A. A notação UML para a relação de herança é representada por uma flecha partindo da subclasse em direção à superclasse, conforme representado na Figura 19.

Figura 19: Exemplo de notação de relação de herança.

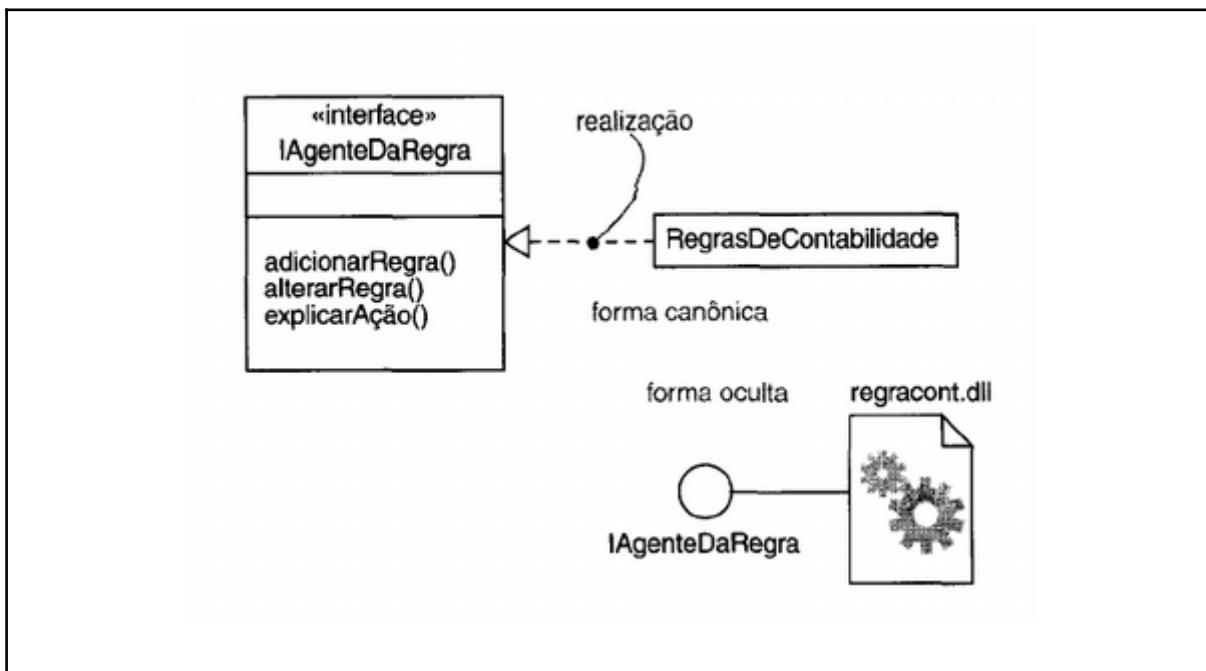


Fonte: Bezerra (2015, p. 129)

2.7.2.9 Realização

A realização é um tipo relacionamento utilizado para indicar a relação onde um classificador (normalmente uma classe ou componente, ou em diagramas de caso de uso, um caso de uso) garante a execução do contrato determinado no classificador realizado (normalmente uma interface, ou em diagramas de caso de uso, um caso de uso). Em outras palavras, indica que uma classe ou componente implementam os serviços determinados em uma interface (BOOCH, 2000). É representada por uma linha tracejada com uma seta vazia apontando para a interface (ou caso de uso) realizada(o), ou em sua forma suprimida, uma linha com um círculo vazio nomeado com a interface realizada, como apresentado na figura 20:

Figura 20: Exemplo de notação de realização de uma interface

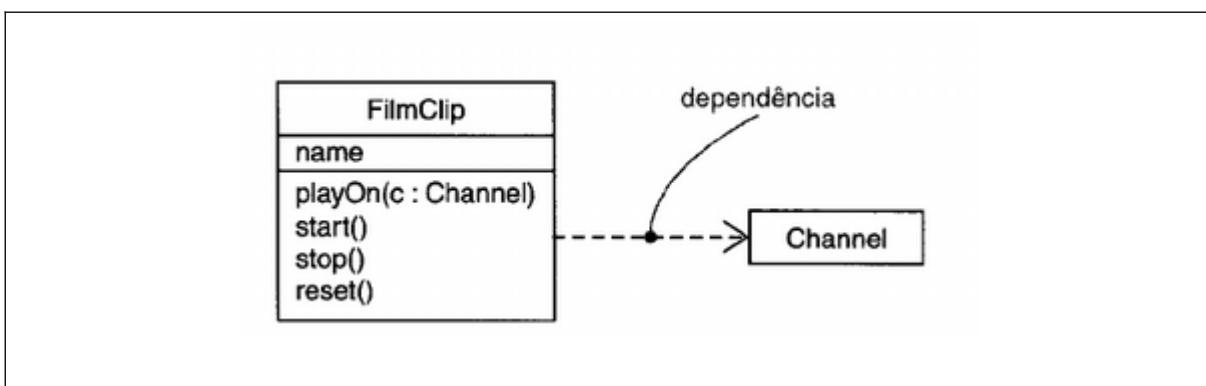


Fonte: Booch (2000, p. 150)

2.7.2.10 Dependência

Uma dependência é um relacionamento da UML indicador de que um item depende de outro e, por consequência, alterações no item podem afetar seus dependentes. Em diagramas de classes, são frequentemente utilizadas quando uma classe é utilizada em uma assinatura de método da classe dependente, por exemplo. Sua representação gráfica ocorre por uma seta tracejada apontando para o item do qual o outro é dependente, conforme expresso na Figura 21 (BOOCH, 2000).

Figura 21: Relação de dependência



Fonte: Booch (2000, p. 63)

2.7.3 Diagrama de Sequência

De acordo com Pressman (2016), um diagrama de sequência é utilizado para indicar as comunicações dinâmicas entre objetos durante a execução de uma tarefa. Ele mostra a troca de mensagens entre objetos em ordem temporal. Pode ser usado para demonstrar as interações em um caso de uso, ou em um cenário de um sistema.

2.7.3.1 Atores

Conforme Bezerra (2015) Em um diagrama de sequência, podem ser representados os atores que participam da realização do caso de uso. A notação gráfica é a mesma utilizada no diagrama de uso, descrita na seção 2.7.1.2.

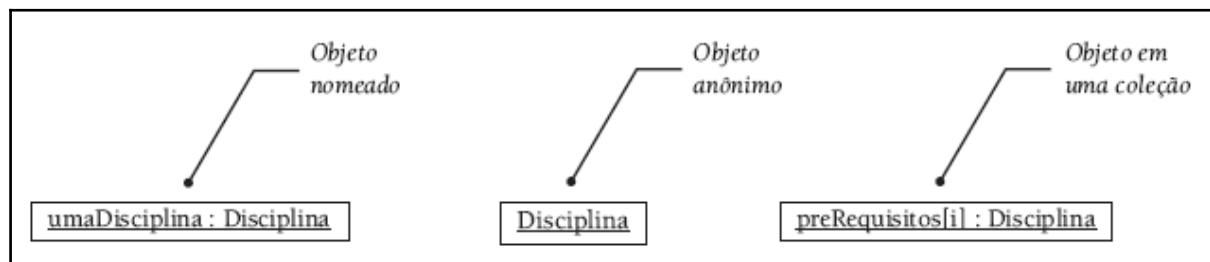
2.7.3.2 Objetos

Um objeto é representado por uma caixa, que deve conter, obrigatoriamente o tipo do objeto (normalmente o nome da classe à qual o objeto corresponde) precedido de dois pontos. Pode-se opcionalmente, declarar antes dos dois pontos o nome dado ao objeto. Esse texto deve ser sublinhado (PRESSMAN, 2016).

Conforme Bezerra (2015), quando deseja-se referenciar uma ocorrência de um objeto em uma coleção, a notação é a referência de seu índice entre colchetes.

A Figura 22 apresenta três diferentes notações para objetos em um diagrama de sequência UML.

Figura 22: Exemplos de notação UML para objetos em um diagrama de sequência



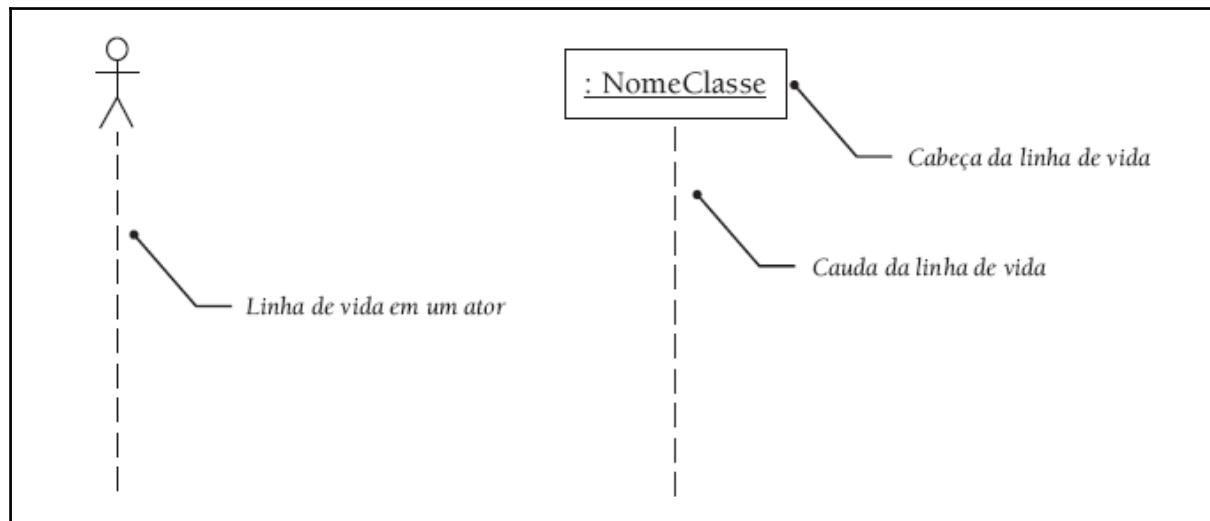
Fonte: Bezerra (2015, p. 200)

De acordo com Bezerra (2015), quando métodos estáticos são chamados, pode-se representar somente a classe que o implementa. Nesse caso, a notação também ocorre por uma caixa, porém somente com o nome da classe e sem sublinhar o texto.

2.7.3.3 Linha de vida

Em um diagrama de sequência, a linha de vida de um objeto é composta por duas partes: cabeça e cauda. A cabeça, corresponde à notação do objeto, e a cauda é representada por uma linha vertical tracejada. Atores também podem apresentar linha de vida. Nesse caso, a cabeça é substituída pela notação gráfica do ator (BEZERRA, 2015). A Figura 23 apresenta as representações gráficas de linhas de vida de atores e objetos.

Figura 23: Notação UML para linha de vida



Fonte: Bezerra (2015, p. 204)

2.7.3.4 Mensagens

Uma mensagem é uma solicitação de execução de uma operação em outro objeto, é por meio do envio de mensagens que os objetos de um sistema colaboram entre si para prover funcionalidades. A notação UML para uma mensagem em um diagrama de sequência é uma flecha ligando a linha de vida do objeto de envio à linha de vida do objeto receptor. O posicionamento vertical de uma mensagem em uma linha de vida, indica sua posição temporal de envio. Quanto mais acima se posiciona, mais cedo foi enviada. Da mesma forma, quanto mais abaixo se posiciona, mais tarde foi enviada. Através disso é possível identificar a sequência de mensagens enviadas e recebidas

durante a realização de um caso de uso. O formato da seta indica o tipo de mensagem que está sendo enviada, conforme Quadro 4. O rótulo da mensagem deve ser posicionado acima da linha da seta (BEZERRA, 2015).

Quadro 4: Notação gráfica para tipos de mensagem em um diagrama de sequência

Tipo de mensagem	Notação gráfica
Mensagem Síncrona	→
Mensagem assíncrona	→
Mensagem de retorno	←-----
Mensagem de sinal	-----→

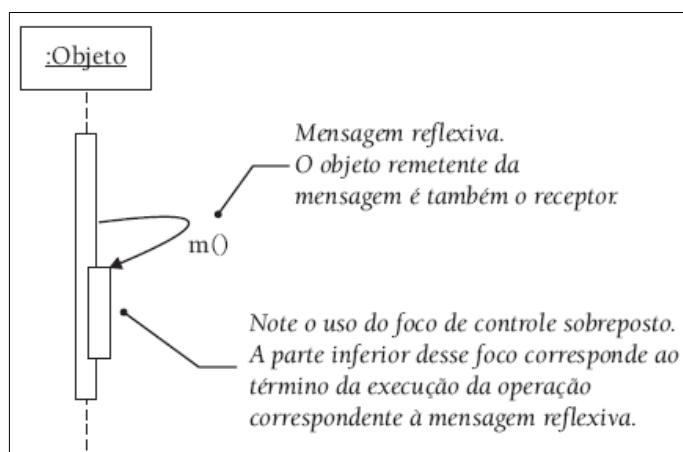
Fonte: Adaptado de Bezerra (2015, p. 205)

De acordo com Pressman (2016), o rótulo de mensagens síncronas e assíncronas deve identificar o método chamado e, opcionalmente incluir os parâmetros, seus tipos, e o tipo de retorno. No caso de mensagens de retorno, o rótulo é opcional.

Conforme Bezerra (2015), mensagens de sinal são utilizadas, principalmente, como uma opção de notação para criação ou destruição de objetos. Nesse caso, os rótulos devem ser, respectivamente <<create>> e <<destroy>>.

Existem ainda as mensagens reflexivas, que indicam o envio de mensagem de um objeto a si mesmo. Nesse caso, a notação deve ser rotulada, porém a seta segue o formato apresentado na Figura 24 (BEZERRA, 2015).

Figura 24: Notação UML para mensagens reflexivas



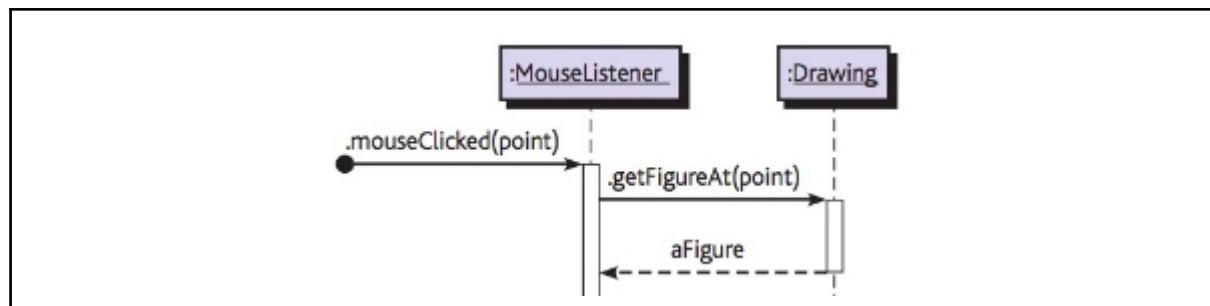
Fonte: Bezerra (2015, p. 205)

2.7.3.5 Ocorrências de execução

Uma ocorrência de execução³ representa o tempo durante o qual um objeto realiza uma operação. A notação gráfica determinada pela UML é a de um retângulo vazio, disposto sob a linha de vida, na posição vertical correspondente ao período de atividade. O início da execução é, portanto, indicado pela parte superior do retângulo, e normalmente está alinhado com a recepção de uma mensagem. A parte de baixo do retângulo indica o fim da execução e normalmente substitui a notação de uma mensagem de retorno, visto que o final da execução já está representado (BEZERRA, 2015).

Na Figura 25, observa-se uma ocorrência de execução na linha de vida do objeto Drawing, iniciada pela chamada do método *getFigureAt*.

Figura 25: Exemplo de notação de ocorrência de execução



Fonte: Adaptado de Pressman (2016, p. 878)

2.7.3.6 Criação e destruição de objetos

Conforme Bezerra (2015), os momentos de criação e destruição de objetos também podem ser representados em um diagrama de sequência. A criação de um objeto corresponde ao momento em que ele passa a existir e realizar suas responsabilidades no sistema. Um objeto é destruído quando não realiza mais operações na execução. As chamadas de criação e destruição de um objeto explícitas são realizadas através da notação apresentada na seção 2.7.3.4.

Se um objeto existe desde o início da execução de uma sequência, então ele deve ser posicionado no topo. Entretanto, caso ele seja criado em determinado momento, deve ser verticalmente posicionado alinhado à mensagem de criação, que pode ser uma chamada ao objeto, ou uma mensagem com o rótulo <<create>>. A destruição de um

³ Também chamada de barra de ativação, conforme Pressman (2016).

objeto pode ser indicada por uma mensagem com o rótulo <<destroy>>, ou por um “X” no final de sua linha de vida.

2.7.4. Diagrama de Atividade

De acordo com Booch (2000), diagramas de atividades compõem o grupo de diagramas UML para modelagem de aspectos dinâmicos de sistemas. São em essência um gráfico de fluxo utilizado para mostrar o fluxo de controle entre atividades, mostrando as sequências entre atividades únicas e frequentemente concorrentes.

Segundo Pressman (2016), um diagrama de atividades é composto por nós de ação (considerado o principal elemento), nó inicial, nó final, *fork*, *join*, nó de decisão e raias. Esses elementos são conectados por setas que representam o fluxo de controle. Os detalhes de cada elemento são descritos a seguir. Na sequência, a Figura 26 apresenta um exemplo de diagrama de atividade composto por todos os elementos.

2.7.4.1 Nó de ação

Representado graficamente por um retângulo arredondado, indica uma tarefa a ser executada por um sistema de software. Ao encerrar uma ação, a atividade segue para o próximo elemento indicado pelo fluxo de controle.

2.7.4.2 Nó inicial

Indica o início da atividade, sendo o primeiro elemento em um diagrama de atividade, é representado graficamente por um círculo preto preenchido

2.7.4.3 Nó final

Representado graficamente por um círculo preto preenchido, envolvido por um círculo com contorno preto. Indica o final de uma atividade, isto é, após a execução de todo o fluxo, marca o fim do fluxo de controle.

2.7.4.4 Fork

Indica uma bifurcação do fluxo de controle em dois ou mais fluxos de atividades que ocorrem em concorrência. Graficamente representado por uma barra preta horizontal conectada por um fluxo único de entrada, podendo ter como saída dois ou mais fluxos, de acordo com a quantidade de fluxos que ocorrem em concorrência.

2.7.4.5 *Join*

A função *join* sincroniza fluxos de controle concorrentes. É representada por uma barra preta horizontal, conectada por dois ou mais fluxos de entrada, correspondentes aos fluxos de controle que sincroniza, seguido de um fluxo de saída, que leva para o próximo controle da atividade. Importante mencionar que a função não produz saída sem que todos os fluxos concorrentes sejam concluídos.

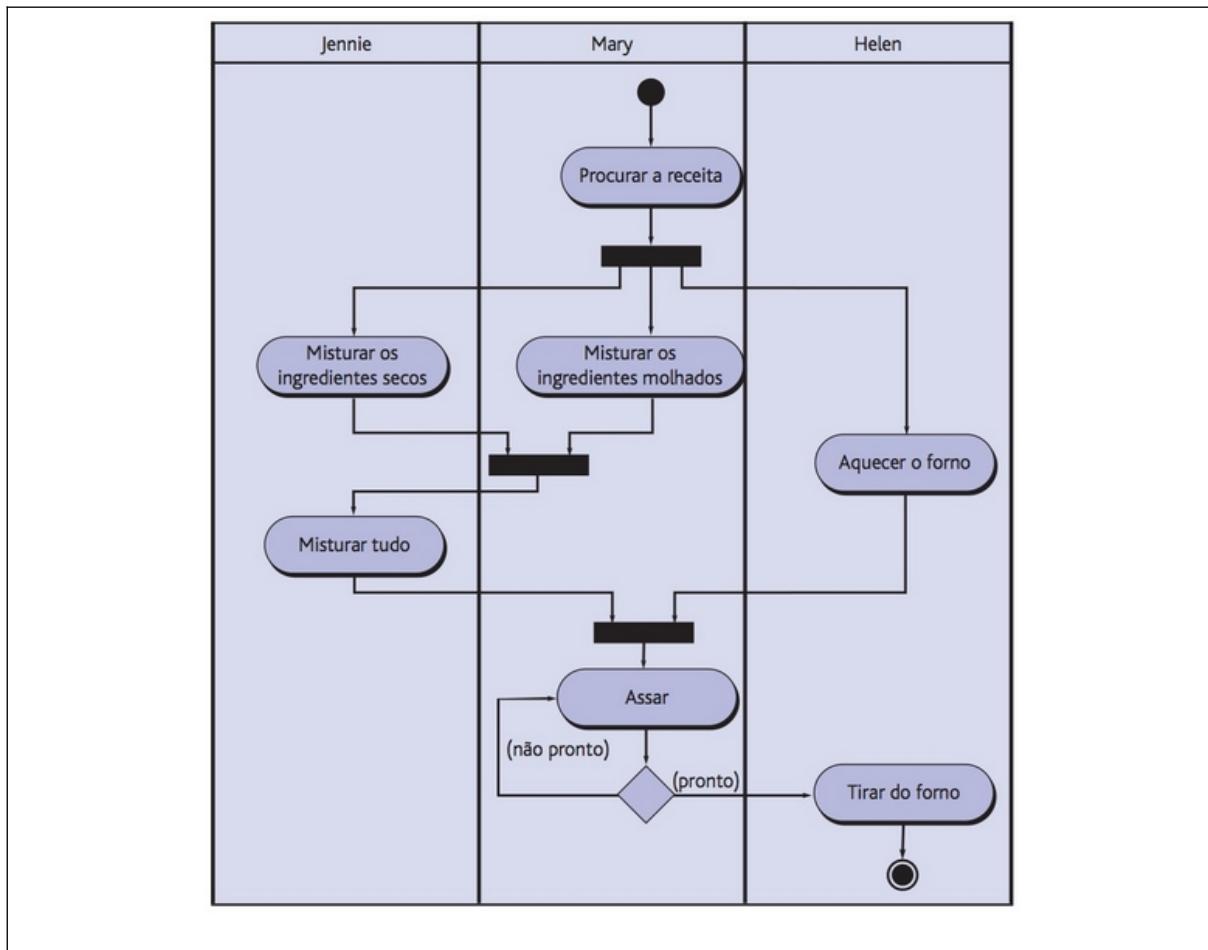
2.7.4.6 Nô de decisão

Um nó de decisão indica uma ramificação no fluxo de controle, baseada em uma condição. É indicado por um losango conectado por um fluxo de controle de entrada, e dois ou mais fluxos de saída, cada qual correspondente e indicado por uma condição a ser atendida. O fluxo de atividades segue pela condição que for atendida. Deve-se certificar que todas as condições sejam abrangidas e que somente uma delas seja atendida para cada nó de decisão.

2.7.4.7 Raias

Trata-se de um recurso opcional, utilizado para indicar o ator de cada ação. São separadores verticais ou horizontais, com um cabeçalho contendo o nome do ator responsável pela execução da atividade. Logo, os elementos posicionados em determinada raia indicam que são de responsabilidade do ator correspondente.

Figura 26: Exemplo de diagrama de atividade composto por todos os componentes



Fonte: Pressman (2016, p. 883)

2.7.5 Diagrama de Componentes

Segundo Booch (2000), Diagramas de componentes, junto de diagramas de implantação são os dois tipos de diagramas UML para modelagem de aspectos físicos de sistemas orientados a objetos. O diagrama de componentes, especificamente, mostra a organização e dependências existentes entre um conjunto de componentes.

São diagramas empregados na modelagem da visão estática de implementação de um sistema, isso inclui itens físicos como executáveis, bibliotecas, tabelas, arquivos e documentos. Em essência, são diagramas de classe com foco em componentes de um sistema, empregados para especificar detalhes para a construção, bem como visualizar o aspecto estático desses componentes e seus relacionamentos.

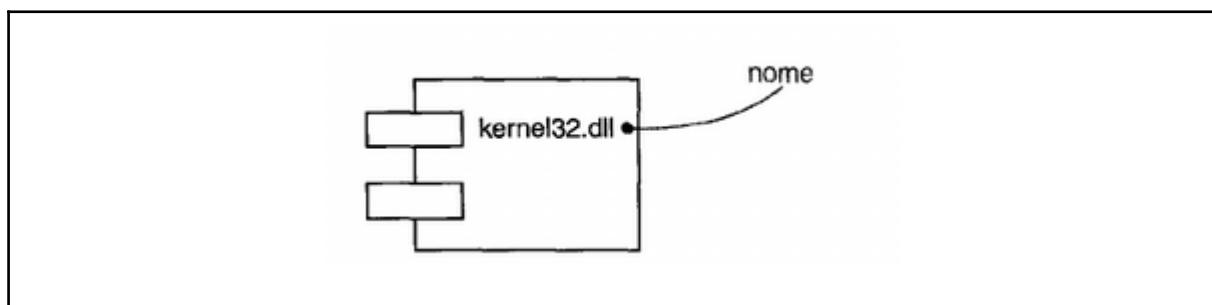
Os elementos comuns de um diagrama de componentes são: componentes; interfaces (capítulo 2.7.2.1.1); e relacionamentos de dependência (2.7.2.10), generalização (capítulo 2.7.2.8), associação (capítulo 2.7.2.2) e realização (capítulo

2.7.2.9), assim como diagramas de classe. Podem também conter pacotes, subsistemas e instâncias, além disso, assim como todos os diagramas UML, podem conter notas e restrições.

2.7.5.1 Componentes

Os demais elementos de um diagrama de componentes foram vistos nos capítulos anteriores, indicados no parágrafo anterior. Desse modo, resta descrever o elemento componente. Segundo Booch (2000), “um componente é uma parte física e substituível de um sistema ao qual se adapta e fornece a realização de um conjunto de interfaces”. Graficamente é representado como um retângulo com abas, sendo composto também por um nome, conforme expresso na Figura 27.

Figura 27: Notação gráfica de um componente



Fonte: Booch (2020, p. 343)

2.8 ARQUITETURA DE SOFTWARE

De acordo com Bass, Clements e Kazman (2003, apud PRESSMAN 2016, p. 254), a arquitetura de um software é “a estrutura, ou estruturas, do sistema, o que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles.”. Conforme Pressman (2016), a arquitetura é uma representação que permite analisar a efetividade do projeto ao atender os requisitos, considerar alternativas de arquitetura em um estágio em que ainda é fácil fazer mudanças de projeto e, por fim, reduzir os riscos associados à construção do software.

Para Sommerville (2011), a arquitetura de software identifica os principais componentes estruturais de um sistema e como se relacionam, unindo a engenharia de requisitos ao projeto. Para Bosch (2000, apud SOMMERVILLE, 2011) características da

arquitetura de software são capazes de afetar o desempenho, robustez, capacidade de distribuição e manutenibilidade de um sistema.

De acordo com Pressman (2016), um padrão arquitetural é a descrição de uma forma padronizada de organizar um sistema. Um estilo arquitetural, por sua vez é um padrão arquitetural de referência, cuja aplicação é difundida e identificada em diversos sistemas. Sommerville (2011), entretanto, trata padrão arquitetural como um termo equivalente a estilo arquitetural, nesse trabalho essa é a nomenclatura adotada.

Sommerville (2011). A arquitetura de um software pode se basear em um determinado padrão ou estilo de arquitetura. Um padrão de arquitetura é uma forma de organizar o sistema, aplicada em diversos sistemas de software. Um padrão busca descrever a essência da arquitetura de um software e seu fluxo de trabalho, com base na percepção de características arquiteturais entre as aplicações observadas.

Cada padrão de arquitetura descreve uma categoria de sistema que engloba:

“(1) um conjunto de componentes (por exemplo, um banco de dados, módulos computacionais) que realiza uma função exigida por um sistema, (2) um conjunto de conectores que habilitam a “comunicação, coordenação e cooperação” entre os componentes, (3) restrições que definem como os componentes podem ser integrados para formar o sistema e (4) modelos semânticos que permitem a um projetista compreender as propriedades gerais de um sistema por meio da análise das propriedades conhecidas de suas partes constituintes (BASS; CLEMENT; KAZMAN, 2003, apud PRESSMAN 2016, p. 258).”

Ao tomar decisões sobre a arquitetura de um software, deve-se conhecer os padrões comuns, bem como suas aplicações e pontos fracos e fortes (SOMMERVILLE, 2011). A seguir, serão descritos três padrões de arquitetura de software: cliente-servidor, MVC e arquitetura de transação (ou *transaction script*).

2.8.1 Arquitetura Cliente-Servidor

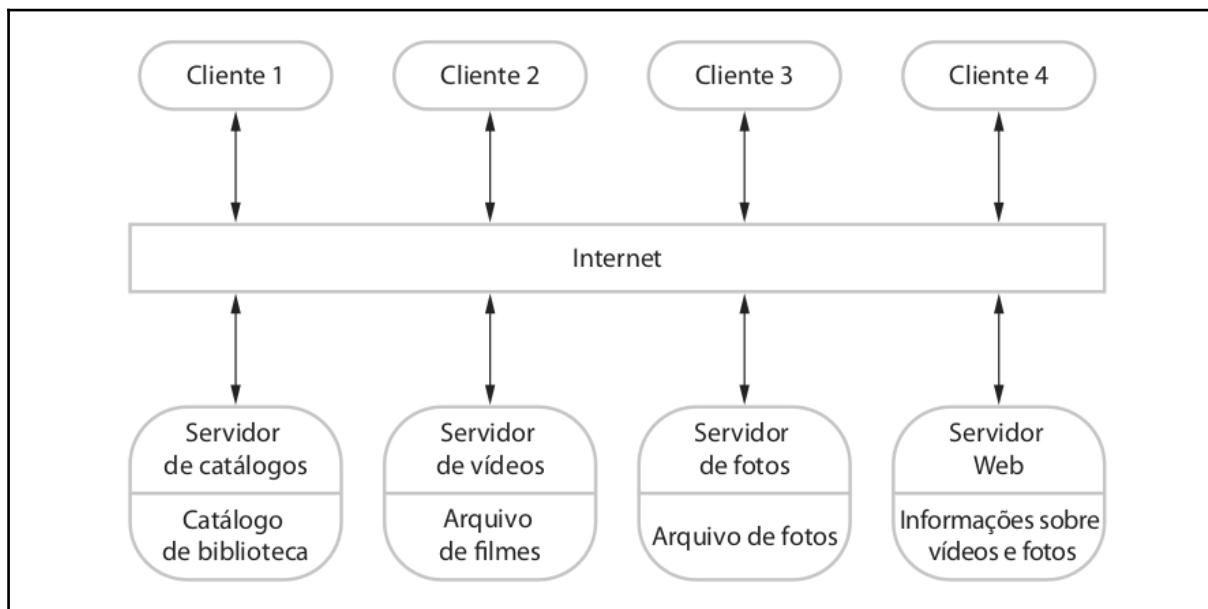
De acordo com Sommerville (2011), o padrão de arquitetura cliente-servidor aborda uma organização muito usada em sistemas distribuídos em tempo de execução. Em uma arquitetura cliente-servidor, a funcionalidade do sistema está organizada em serviços, onde cada serviço é prestado por um servidor. Os clientes são os usuários desses serviços que acessam os servidores para fazer uso deles. Os principais componentes desse padrão são:

1. Um conjunto de servidores que oferecem serviços a outros componentes, como por exemplo: servidores de arquivos e servidores web;

2. Um conjunto de clientes que consomem os serviços oferecidos pelos servidores;
3. Uma rede que permite aos clientes acessar esses serviços.

A maioria dos sistemas cliente-servidor é implementada como sistemas distribuídos, conectados através de protocolos de internet. Entretanto, vale ressaltar que pode-se implementar um modelo lógico de serviços independentes em um único computador (SOMMERVILLE, 2011). Uma arquitetura cliente-servidor para uma biblioteca de filmes é ilustrada na Figura 28, onde os servidores provêm acesso a serviços específicos, que são acessados pelos clientes através da rede, nesse caso, a Internet.

Figura 28: Exemplo de arquitetura cliente-servidor para uma biblioteca de filmes



Fonte: Sommerville (2011, p. 114)

De acordo com Sommerville (2011), entre as vantagens da implementação dessa arquitetura, pode-se citar:

1. O reúso da transformação é de fácil compreensão e suporte;
2. Estilo de *workflow* corresponde à estrutura de muitos processos de negócios.
3. Evolução por adição de transformações é simples.
4. Pode ser implementado tanto como um sistema sequencial quanto concorrente.

Ainda segundo o autor, entre as desvantagens, há a necessidade de um protocolo e formato padrão para transferência de dados, da mesma forma, as saídas devem ser geradas em um formato acordado.

2.8.2 Arquitetura MVC

O padrão Model-View-Controller (MVC) divide uma aplicação em três partes interconectadas: *model*, *view* e *controller* (ou, modelo, visão e controlador). O MVC utiliza uma solução já definida para separar componentes e as responsabilidades de cada um, reduzindo a dependência entre eles (ZENKER et al., 2019).

Segundo Bezerra (2015), o MVC é um padrão arquitetural que propõe uma forma de organizar a interface com o usuário e descreve de que forma o estado dessa interface deve ser atualizado. Busca a separação de entre a lógica da apresentação e a lógica da aplicação, através da separação funcional em três principais componentes.

O componente *model* é a parte da aplicação que contém os dados e suas validações. Esse componente corresponde ao estado, à estrutura e ao comportamento dos dados manipulados pela aplicação. Fornece operações em sua interface que permitem ao restante da aplicação manipular os dados para operações de edição, exclusão, atualização, etc. (BEZERRA, 2015).

O componente *view* do MVC representa cada uma das possíveis formas de apresentação de informações provenientes do *model*, e fornece uma interface para o usuário interagir com a aplicação. O componente de visualização não deve conter inteligência, deve apenas implementar lógica de apresentação, como repetições para exibição de itens em lista e exibição de informações recebidas do *model*, por exemplo. Cada *view* está associada a um componente controlador (o *controller*) que auxilia na implementação da interface gráfica, o Controller (BEZERRA, 2015).

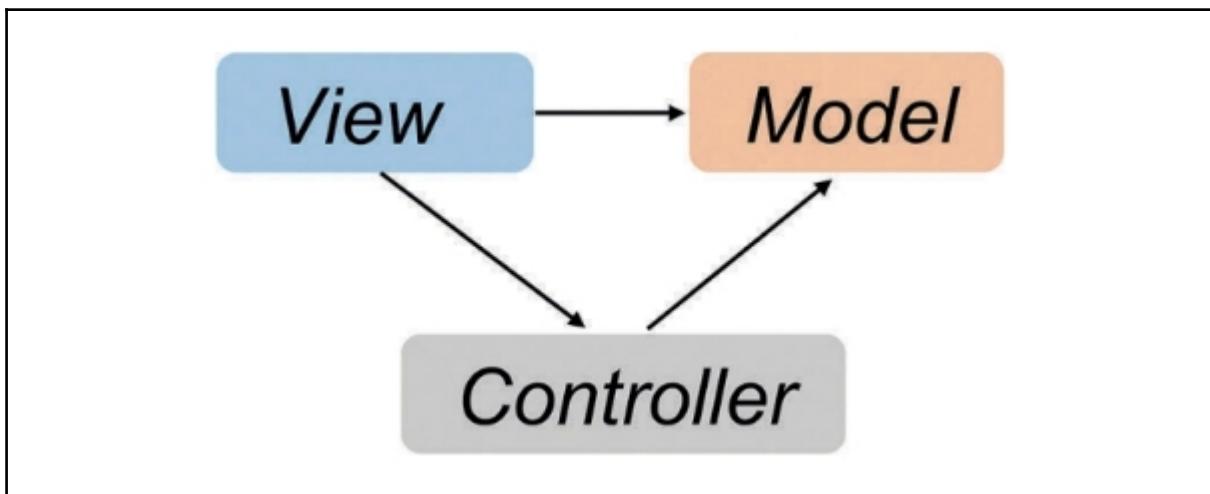
De acordo com Zenker et al. (2019), desde a criação do padrão original, em 1979, diversas variações foram criadas para acompanhar novas demandas de interação com o usuário, aplicando-se a vários tipos de projeto, como *desktop*, *web* e *mobile*. Em sua essência, o fluxo de controle do MVC ocorre da seguinte maneira, conforme Wilson (2015, apud ZENKER et al., 2019, p. 97):

1. O controlador (*controller*) é responsável por controlar e mapear as ações do usuário para que o modelo entenda, implementando a regra de negócio do software. Na prática, o controlador analisa uma solicitação do usuário e a partir disso determina o que deve ser feito.
2. O modelo (*model*) é responsável pela manutenção dos dados, representa o estado das entidades do banco de dados, que tem atributos e operações;

3. Por fim, a visão (view) inclui os elementos de interação com o usuário, nela estão contidos os componentes visuais e o código com a interface a ser mostrada. Ela não faz nenhuma pesquisa diretamente no banco, apenas recebe, manipula e mostra os dados. Em geral, cada modelo está associado a diversas visões, que permitem a realização das diferentes operações associadas ao modelo

A Figura 29 ilustra a interação entre os componentes em uma arquitetura MVC.

Figura 29: Padrão MVC



Fonte: Zenker et al., (2019, p. 97)

2.8.3 Arquitetura de Transação (*Transaction Script*)

Para Fowler (2002, tradução minha), a maioria das aplicações corporativas pode ser concebida como uma série de transações. Uma transação pode ser uma exibição ou alteração de alguma informação, por exemplo. Cada interação com o sistema contém uma certa quantidade de lógica, simples ou complexa. Um *transaction script* (*script* de transação) organiza toda essa lógica em um único procedimento, denominado transação (*transaction*). Uma arquitetura *Transaction Script*, portanto, organiza a aplicação e suas funcionalidades em um conjunto de transações que podem ser realizadas por um sistema.

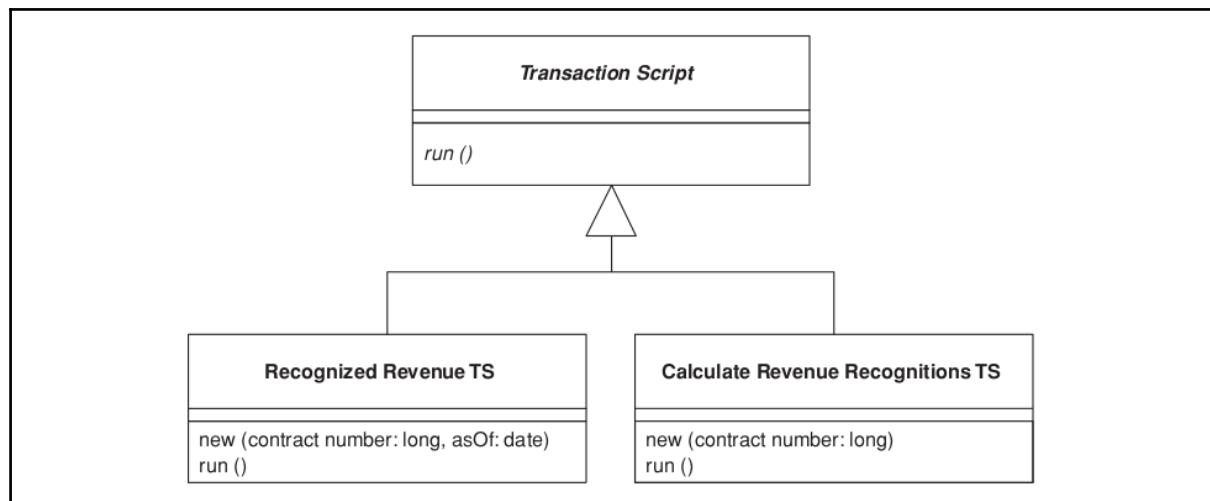
O termo *transaction script* é utilizado porque, para a maioria das operações, é realizada uma transação com o banco de dados, que pode ocorrer através de uma chamada direta, ou por intermédio de uma camada de encapsulamento de dados. Cada transação possui seu próprio *transaction script* que inclui todas as instruções necessárias para a realização de uma operação, pode ainda incluir subtarefas em comum com outros

transaction scripts, que podem ser implementadas como subtarefas (FOWLER, 2002, tradução minha).

Como em qualquer programa, o código deve ser separado em módulos, Fowler (2002, tradução minha) recomenda que os *transaction scripts* sejam agrupadas em classes. A Figura 30 apresenta um diagrama de classes de uma implementação em Transaction Script, onde cada uma das classes herdeiras de *Transaction Script* implementam diferentes funcionalidades. Se a aplicação envolve validações, cálculos e regras de negócio complexas, então o padrão arquitetural *Domain Driven* pode ser uma boa abordagem. Entretanto, caso implemente validações e regras de negócio simples, então *transaction script* é a abordagem adequada.

Um dos benefícios dessa abordagem é o fato de precisar se preocupar individualmente com as funcionalidades, visto que cada transação é implementada forma independente. Por outro lado, à medida que a complexidade da aplicação aumenta, pode se tornar difícil manter um bom design arquitetural e prevenir *transaction scripts* duplicados (FOWLER, 2002, tradução minha).

Figura 30: Diagrama de classes de uma aplicação implementada no padrão Transaction Script



Fonte: Fowler (2002, p. 111)

2.9 TESTES

A disciplina de Testes busca verificar os resultados da implementação, através do planejamento, desenho e realização das atividades do processo de testes. Testar um software é indispensável para a identificação de defeitos e avaliação do grau de

qualidade⁴ de um produto e de seus componentes. Fatores como cronograma e orçamento frequentemente impõem restrições à quantidade de testes que é possível executar (PAULA FILHO, 2009).

Segundo Paula Filho (2009), “um teste é uma atividade na qual um produto, sistema ou componente é executado sob condições especificadas, com observação e registro dos resultados e avaliação de um ou mais aspectos”. Cada teste tem, pelo menos, um objetivo de teste, que é um conjunto identificado de características que devem ser testadas, mensuradas e depois comparadas em relação ao comportamento esperado, conforme descrito na documentação do software. Um critério de teste é um critério que o componente testado deve satisfazer para passar em um dado teste. Uma bateria de testes é uma implementação de um teste que atende a um objetivo específico.

Segundo Glass (2003, apud PAULA FILHO, 2009) remover defeitos é a etapa do processo de software que mais consome tempo. Sendo quase impossível executar uma boa remoção de erros sem uso de ferramentas, e raramente são utilizadas ferramentas além de depuradores.

É importante que os testes sejam bem planejados e desenhados para melhor aproveitamento dos recursos alocados para eles. Nem sempre a detecção de um defeito por um teste é algo óbvio, por isso durante e após a realização de um teste, os resultados devem ser inspecionados, comparando os resultados previstos e os resultados obtidos (PAULA FILHO, 2009).

O planejamento e o desenho de testes devem ser feitos por pessoas que conheçam a metodologia de testes usada, enquanto a realização dos testes baseada em um desenho bem definido, pode ser feita por uma pessoa menos experiente ou de forma automatizada. Recomenda-se que os testes sejam feitos, se não por testadores, ao menos por outra pessoa que não aquela que desenvolveu o software, pois frequentemente têm maior dificuldade para identificar problemas nos produtos de seu trabalho (PAULA FILHO, 2009).

Um procedimento de teste é um conjunto detalhado de instruções para execução de um teste, normalmente derivado da descrição de um caso de uso. Um caso de teste, por sua vez, é a especificação das entradas, resultados previstos e condições de execuções para um item a se testar. Um teste pode ser visto como uma coleção desses dois elementos (PAULA FILHO, 2009).

⁴ Quanto maior o número de defeitos detectados em um software, provavelmente maior também o número de defeitos não-detectados. (PAULA FILHO, 2009)

Um procedimento de teste pode ser executado de forma manual ou automatizada. Um script de teste é uma representação formalizada de um procedimento de teste, normalmente é associado a um script executável de forma automatizada, mas testes manuais também podem utilizar representações mais estruturadas (PAULA FILHO, 2009).

O objetivo de um caso de teste não é demonstrar que um programa funciona, e sim encontrar defeitos. Para isso, deve obrigatoriamente incluir uma descrição de saídas esperadas, com as quais será comparado o resultado da execução, e um conjunto de entradas. Devem existir casos de testes tanto para entradas válidas como inválidas. Um caso de teste pode invocar vários procedimentos de teste, pode também depender do resultado de um outro teste, por isso a execução de casos de testes deve ter uma ordem especificada (PAULA FILHO, 2009).

A quantidade potencial de defeitos que podem ser detectados na execução de um testes é denominada cobertura de testes. O objetivo central de toda a disciplina de testes é maximizar a cobertura, que pode ser potencializada com ferramentas de automação (PAULA FILHO, 2009).

Testes de software podem ser classificados de acordo com diferentes características, o que implica em uma nomenclatura diversa. As características pelas quais um teste de software pode ser classificado são: o papel que desempenham dentro do processo de software, visibilidade em relação ao componente de teste; formalismo na execução; e automação da execução (PAULA FILHO, 2009).

2.10 BANCO DE DADOS

Segundo Cardoso e Cardoso (2012), o conceito de banco de dados surgiu diante da necessidade de armazenar dados e disponibilizá-los para consulta e acesso, em um cenário onde, até então, os dados eram mantidos em grupos de registros, armazenados em sistema de arquivos. Essa forma de armazenar dados recorrentemente apresentava problemas de incompatibilidade, redundância, inconsistência, dificuldade de acesso e problemas de segurança. Nesse contexto, surgiu o conceito de banco de dados: uma coleção de dados organizada em uma estrutura para armazenamento de informações e com propriedades determinadas.

Machado (2014) define banco de dados como um conjunto de dados devidamente relacionados, organizado de forma lógica e com um significado inerente, que representam e refletem um aspecto do mundo real. Além disso, deve ser projetado, construído e

preenchido com valores de dados para um propósito específico, visando atender um conjunto predefinido de usuários e de aplicações.

2.10.1 Modelos de Dados

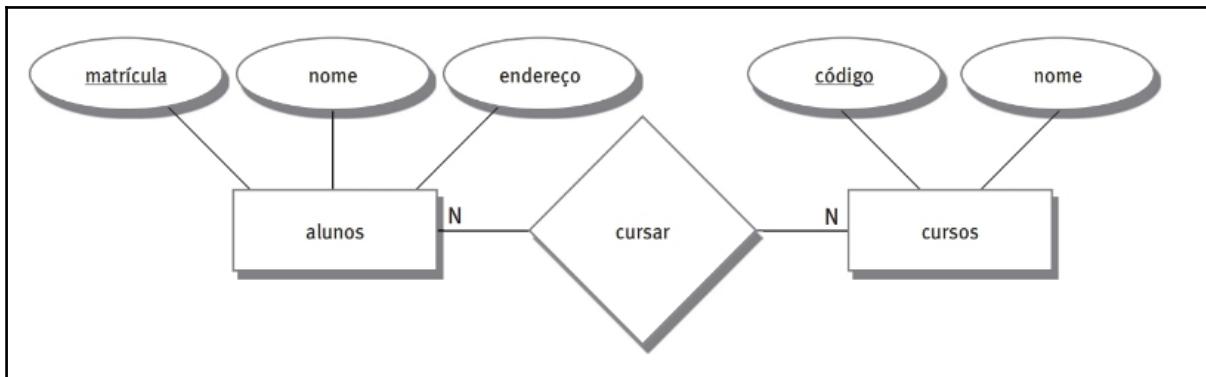
Para serem armazenados em um sistema de banco de dados, os dados devem ser dispostos de forma organizada e coerente. Um modelo de dados é um conjunto de ferramentas utilizadas para fornecer uma representação dos conceitos e estruturas lógica e física, junto de suas propriedades, em um sistema de banco de dados. Os modelos podem ser classificados de acordo com seu nível de abstração, entre modelos de alto nível e modelos de baixo nível (CARDOSO, CARDOSO, 2012).

Um modelo de alto nível é conceitual e fornece uma visão da estrutura de um banco de dados próxima de como ela é vista naturalmente, enquanto um modelo de baixo nível (também chamado de modelo físico) fornece uma visão de como os dados são realmente armazenados em um computador (CARDOSO, CARDOSO, 2012).

2.10.1.1 Modelo de entidade-relacionamento

Conforme Cardoso e Cardoso (2012), nesse modelo, o mundo real é representado por um conjunto de objetos básicos, denominados entidades, e os relacionamentos entre elas. As entidades são descritas por meio de atributos, que representam os dados associados ao objeto representado. Os relacionamentos correspondem às associações entre as entidades. A Figura 31 apresenta um exemplo de modelo de entidade-relacionamento.

Figura 31: Exemplo de modelo de entidade-relacionamento



Fonte: Cardoso e Cardoso (2012, p. 21)

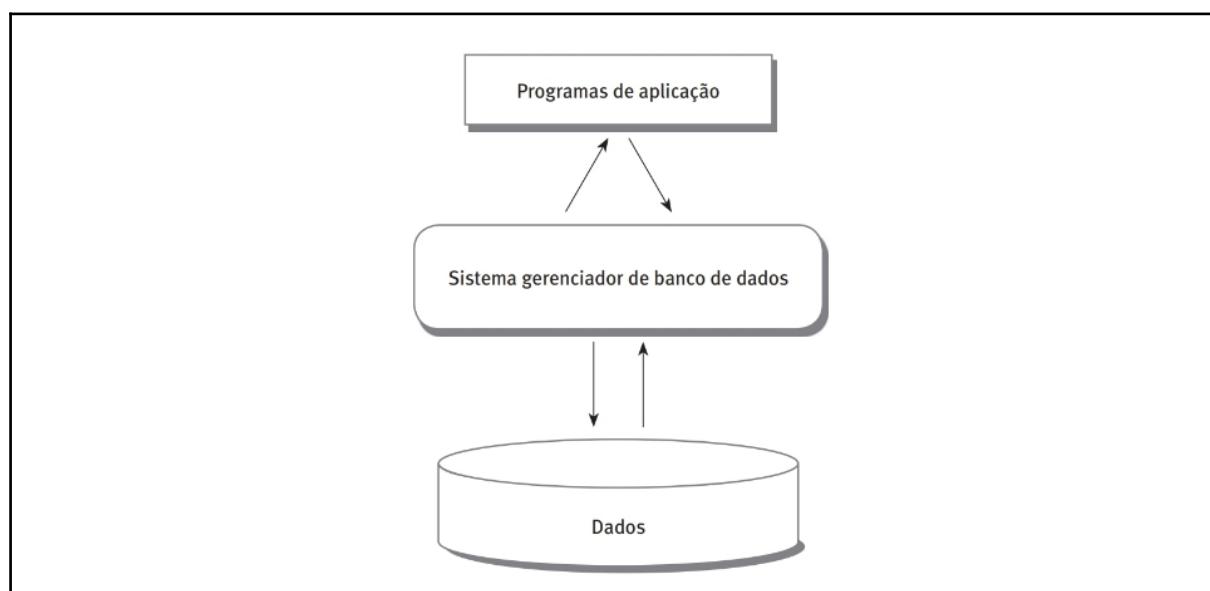
2.10.2 Sistema Gerenciador de Banco de Dados

Conforme Cardoso e Cardoso (2012), um sistema gerenciador de banco de dados é um software projetado para guardar e gerenciar bancos de dados, permitindo que usuários criem e manipulem bancos de dados com diferentes propósitos. Seu objetivo é proporcionar o acesso fácil e eficiente a informações armazenadas. Um SGBD possui uma estrutura que mantém não somente os dados, mas também a forma como são armazenados, contendo uma descrição completa desse banco.

Para possibilitar a definição, manipulação e consulta de dados em um banco de dados, um sistema de banco de dados possui uma linguagem de consulta de alto nível, denominada Structured Query Language (SQL). A linguagem SQL pode ser dividida entre Data-Definition Language (DDL) e Data Manipulation Language (DML), cuja função é, respectivamente, definir esquemas e estruturas de bancos de dados; e manipular (consultar, inserir, remover ou modificar) informações armazenadas (CARDOSO, CARDOSO, 2012).

Segundo as autoras, por meio de um programa de aplicação é possível utilizar consultas SQL para conversar com o SGBD e fazer manipulações e consultas com os dados. O conjunto formado por um banco de dados, um SGBD e as aplicações que o manipulam é chamado de sistema de banco de dados, e é ilustrado na Figura 32, onde fica mais clara a função exercida por um SGBD.

Figura 32: Representação de um sistema de banco de dados



Fonte: Cardoso e Cardoso (2012, p. 18)

2.10.2.1 MySQL

O MySQL é um sistema de gerenciamento de banco de dados relacional, que utiliza a linguagem SQL como interface para consulta e manipulação de dados. É um dos SGBDs mais populares e utilizados no mundo, sendo apreciado por várias empresas, entidades e pessoas por possuir um servidor confiável, rápido e de fácil utilização, que pode ser utilizado com grandes bancos de dados, inclusive em aplicações voltadas para a internet. Trata-se de um software rápido, multitarefa e multiusuário (MANZANO, 2011).

De acordo com Manzano (2011) Grande parte de seu sucesso está associado à fácil integração com a linguagem PHP. O SGBD apresenta as seguintes características: portabilidade para diversos sistemas operacionais; compatibilidade com *wwwdrivers* ODBC, JDBC e .NET; módulos de interfaceamento com as linguagens de programação Java, C, C++, Python, Perl, PHP e Ruby; facilidade de uso; excelente desempenho e estabilidade, exigindo poucos recursos de hardware; além de uma versão baseada na filosofia de software livre.

Além das características típicas de um SGBD, Milani (2006) destaca as principais características do MySQL:

- alta portabilidade e API aberta;
- suporte a multithreads
- vários tipos de tabelas de armazenamento disponíveis, especializadas em características como velocidade ou volume de armazenamento, por exemplo;
- velocidade, proporcionada pelo uso do método de armazenamento MyISAM (baseado em tabelas ISAM), cachês em consultas, indexação BTREE e algoritmos de buscas;
- SQL com velocidade otimizada;
- suporte a tráfego criptografado de senhas;
- uma espécie de firewall de autenticação, que permite limitar acesso de hosts

e usuários;

- suporte a *stored procedures, triggers, views* e cursores

No que diz respeito às capacidades do software apresenta alto poder de execução e armazenamento, permitindo o armazenamento de até 65.536 TB em determinado tipo de tabelas e execução de scripts SQL com até 61 uniões de tabelas. Trata-se de um SGBD extremamente poderoso, apto a realizar mais de um bilhão de consultas por dia, ou até mesmo processar milhares de transações por minuto.

2.11 PROGRESSIVE WEB APPS

Segundo Trindade e Affini (2018), aplicativos móveis (Apps) são softwares projetados para execução em dispositivos móveis, como *smartphones* ou *tablets*. A instalação de Apps pode ser realizada através de lojas online, como a App Store para dispositivos com sistema operacional iOS, da fabricante Apple; Play Store para dispositivos Android, sistema operacional desenvolvido pela Google; entre outros.

Ainda segundo as autoras, um aplicativo desenvolvido para uma plataforma Android não é compatível com a plataforma iOS, por exemplo, e vice-versa. Cada plataforma possui suas próprias tecnologias para construção de aplicativos, o que implica na necessidade de desenvolvimento de um novo aplicativo para cada plataforma, sob as tecnologias definidas. Isso torna o desenvolvimento de aplicativos custoso e demorado, pois há pouca portabilidade entre as plataformas, exigindo que um único aplicativo seja reescrito diversas vezes, para compatibilizar com as plataformas.

De acordo com Lima (2017), em 2007, a visão original de Steve Jobs para aplicativos é de que seriam escritos com as mesmas tecnologias utilizadas para construir aplicações Web. Entretanto, em 2008 foi lançada a Apple Store, uma plataforma que permitia aos usuários do sistema operacional iOS baixar aplicativos nativos, resultando no crescimento do número de aplicativos desenvolvidos e disponibilizados através das diversas lojas de aplicativos disponíveis para as plataformas. Porém, o declínio dos aplicativos nativos vem sendo observado.

Segundo Lima (2017), esse declínio pode ser percebido através da redução de *download* de novos *apps*; grande parte dos aplicativos disponíveis nas lojas online sequer

são baixados; além disso, mais de 80% do tempo de usuários de *smartphones* é gasto no uso de até 5 aplicativos. Esses fatos indicam que usuários tendem a baixar somente aplicativos que de fato irão utilizar.

Nesse contexto, *Progressive Web Apps* (PWAs) surgem como uma boa alternativa, pois, assim como aplicativos, podem ser acessados por um navegador web e instalados, para serem executados a partir das telas iniciais dos dispositivos, além de entregar a interface e (quase todos) os recursos de interação com hardware possíveis em um aplicativo com a praticidade de um site (ROCK CONTENT, 2019).

De acordo com o Google (2020, apud TRINDADE, AFFINI, 2018), um PWA é caracterizado por:

- Progressivo: deve atender qualquer usuário independentemente do navegador ou sistema operacional utilizado;
- Responsivo: deve se adequar a diferentes formatos de exibição (*desktop*, *tablet*, *mobile*), ou seja, diferentes resoluções;
- Independente de conectividade: deve funcionar mesmo com conexões lentas ou até mesmo sem conexão, recurso possibilitado por um *Service Worker*;
- *App-like*: adotar layout, recursos e navegação semelhantes a aplicativos;
- Atualizado: deve se manter atualizado de forma automática e discreta, possibilitado por um *Service Worker*;
- Seguro: deve implementar protocolo HTTPS para evitar invasões e adulterações durante a troca de dados;
- Encontrável: pode ser identificado como um “aplicativo” através de um arquivo de manifesto e ao escopo de registro do *Service Worker*;
- Reengajável: permite o engajamento de usuários por meio de notificações *push*;
- Instalável: fornece uma forma de instalar a aplicação, sem necessidade de acessar uma loja de *apps*;
- Linkável: não requer instalação e pode ser compartilhado por meio de URL.

Entre as vantagens dessa tecnologia, destacam-se a retenção e economia. No que tange a retenção, um PWA requer um número reduzido de etapas necessárias para acesso às funcionalidades do produto em relação a um aplicativo comum. Um PWA pode ser acessado diretamente através de um navegador web e fornece ao usuário a opção de instalação simplificada, enquanto a interação com um aplicativo comum requer que o usuário ao menos faça a busca em uma loja online, faça o *download* e execute, para

então experimentar suas funcionalidades. Conforme comentado anteriormente, um PWA não exige a construção de uma nova aplicação para cada plataforma e com isso, os benefícios de economia estão associados à dispensa da necessidade de uma equipe especializada em desenvolvimento para cada plataforma, além do tempo investido para construção do mesmo aplicativo por mais de uma vez (LIMA, 2017).

2.11.1 Requisitos

Segundo Trindade e Affini (2018), para iniciar um projeto *web* utilizando conceitos de PWA são necessários três requisitos: Conexão HTTPS, *Web App Manifest* e *Service Worker*. Vale ressaltar que o funcionamento do *app manifest* ou do *service worker* depende da compatibilidade com o navegador, entretanto, o navegador *Google Chrome* já possui compatibilidade para dispositivos Android e iOS, enquanto o navegador *Safari* já sinaliza desenvolvimento de suporte para PWA e estudos apontam investimento massivo em PWA pela empresa *Microsoft*. Apesar disso, a aplicação deve funcionar mesmo sem compatibilidade com a tecnologia, visto que é uma aplicação *web*.

HTTPS é a sigla em inglês de *Hyper Text Transfer Protocol Secure* (“Protocolo de transferência de hipertexto seguro”, em português), é uma implementação do protocolo HTTP com uma camada de segurança que utiliza o protocolo SSL/TLS. Com isso, os dados são transmitidos por uma conexão criptografada e é possível verificar a autenticidade do servidor e do cliente por meio de certificados digitais Wikipedia (2018, apud TRINDADE, AFFINI, 2018).

O Manifesto do Aplicativo *Web*, conhecido como *Web App Manifest*, é um arquivo em formato JSON⁵ que fornece informações sobre o aplicativo (como nome, autor, ícone e descrição) e como deve se comportar quando instalado no dispositivo. Um *web app manifest* faz parte de uma coleção de tecnologias de *Progressive Web Apps*, que permite a instalação em um dispositivo sem estar em uma loja de aplicativos, além de recursos como trabalhar *offline* e receber notificações *push* (TRINDADE, AFFINI, 2018).

Por fim, um *Service Worker* é um script que o navegador executa em segundo plano, possibilitando recursos que não precisam de uma página ou de interação com o usuário, como notificações *push* e sincronização em segundo plano. Além disso, são capazes de interceptar e tratar solicitações de rede, incluindo o gerenciamento

⁵ JavaScript Object Notation (JSON), é um modelo para armazenamento de transmissão de informações em formato de texto. Bastante utilizado por aplicações *Web* devido a sua estrutura mais compacta do que o modelo XML (TRINDADE, AFFINI, 2018).

programático de um *cache* de respostas. Comportam-se como uma camada entre a aplicação e o navegador, que possibilita criação de experiências offline, interceptação de requisições de rede e atualização de repositórios residentes no servidor, fornecendo uma boa experiência ao usuário (TRINDADE, AFFINI, 2018).

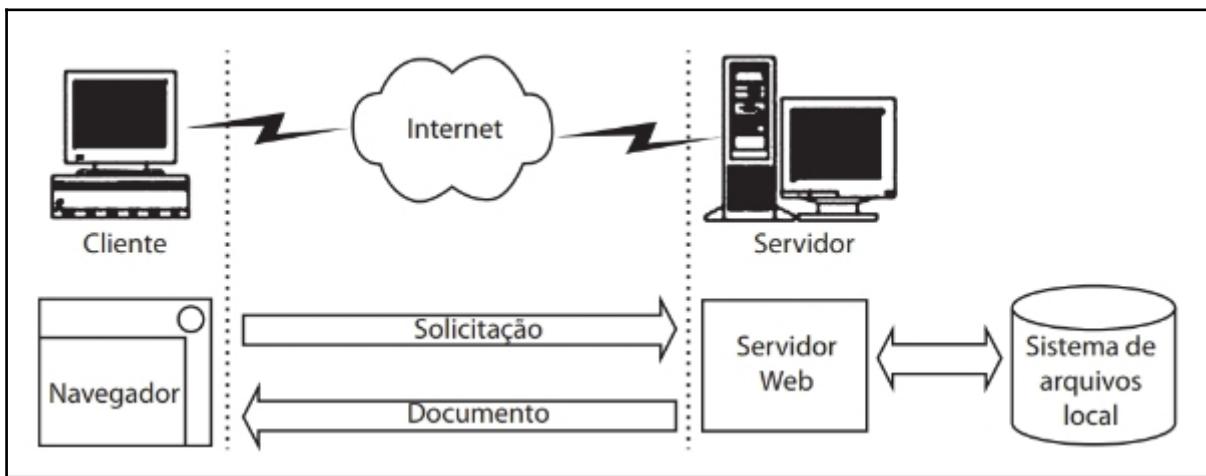
2.12 APLICAÇÕES WEB

Segundo Winckler e Pimenta (2002), uma aplicação web pode ser definida como um software que utiliza a Web como ambiente de execução, pode ser classificada como um Site Web ou um sistema Web (CONALLEM, 2000, apud WINCKLER; PIMENTA, 2002, p. 11). Um Site Web é um sistema hipermídia distribuído, cujo propósito é permitir pesquisa e acesso direto a documentos e informações através da internet para computadores clientes. O acesso ocorre através de um software chamado *browser* (navegador), instalado no computador do cliente, utilizando a infraestrutura da internet, sob o protocolo HTTP (Hyper Text Transfer Protocol). O conteúdo disponível em um Site Web é estático, definido por um arquivo ou documento pré-formatado.

Uma aplicação web é um sistema web que permite aos usuários executarem lógica de negócio e persistir dados através da interação com a interface da aplicação. Diferentemente de sites web, o conteúdo de uma aplicação web é dinamicamente construído, baseado na interação do usuário com as páginas, através do *browser* (WINCKLER; PIMENTA, 2002).

Segundo Miletto e Bertagnolli (2014), um sistema web pode ser considerado um sistema distribuído, implementado sob a arquitetura cliente-servidor, como ilustrado na Figura 33. Na figura, o lado cliente é acessa a aplicação a partir de um navegador web (*browser*); através da internet, são feitas solicitações ao servidor web remoto, que abriga a aplicação e é responsável por interpretar a solicitação, e responde com o respectivo documento. Além disso, a aplicação pode interagir com o sistema de arquivos local através da operação de um banco de dados ou de arquivos propriamente ditos.

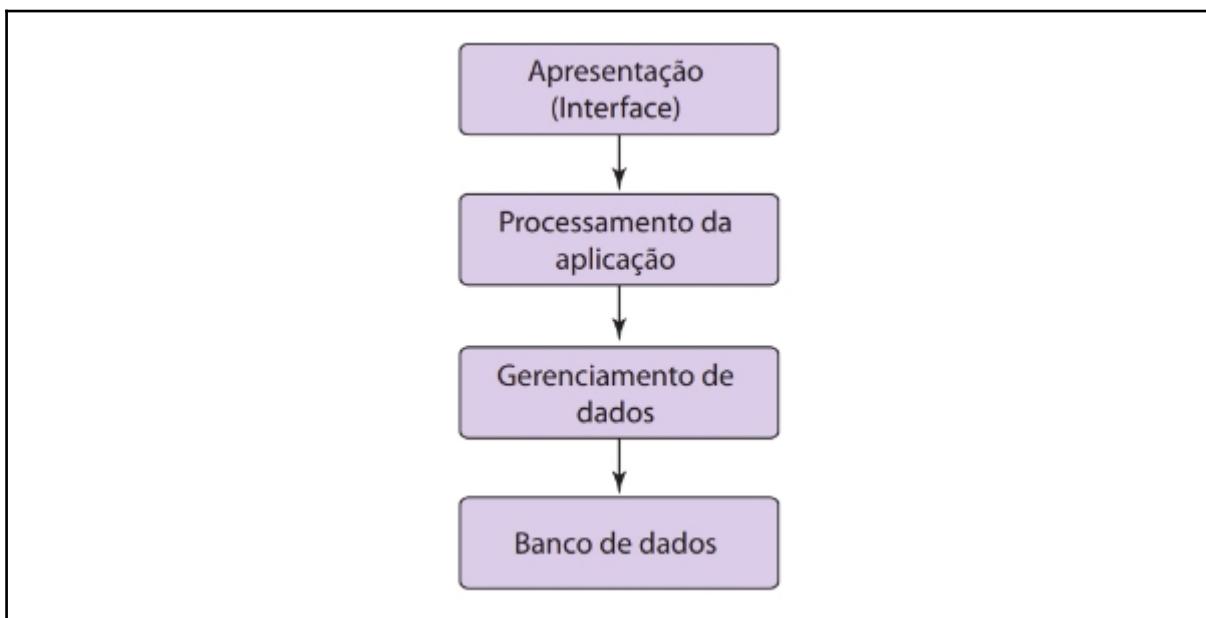
Figura 33: Implementação cliente-servidor de uma aplicação Web



Fonte: (WINCKLER; PIMENTA, 2002, p. 12)

Ainda segundo Miletto, Franco e Bertagnolli (2014), é considerada uma boa prática na construção de sistemas web, estruturar a implementação em camadas com responsabilidades específicas, conforme representado na Figura 34. Essa separação em camadas fornece robustez e consistência, além de reduzir o acoplamento entre os componentes.

Figura 34: Camadas de uma aplicação web



Fonte: Miletto, Franco e Bertagnolli (2014, p. 6)

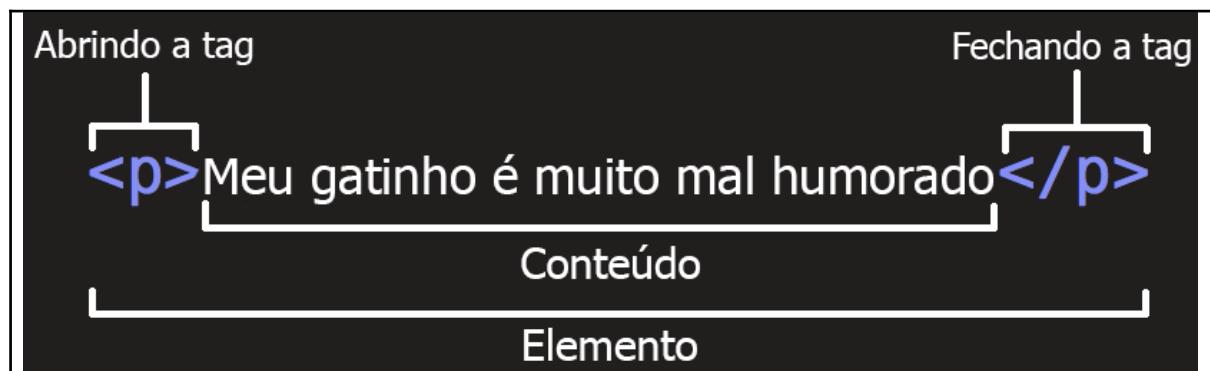
2.12.1 Construção

Cada camada de uma aplicação web pode ser construída através do uso de uma gama de tecnologias. Fundamentalmente, a camada de apresentação é executada no lado do cliente e é construída com o uso da tecnologia HTML, estilizada com CSS e pode ter o comportamento definido pela linguagem *JavaScript*. As camadas de processamento da aplicação, gerenciamento de dados e banco de dados, por sua vez, ficam localizadas no lado do servidor. A camada de processamento da aplicação é construída com o uso de alguma linguagem de programação do lado do servidor, como PHP, por exemplo, que com o uso de linguagem SQL (ver seção 2.10) realiza a manipulação de dados (MILETTO; FRANCO; BERTAGNOLLI, 2014).

2.12.1.1 HTML

De acordo com MDN Web Docs (2020c), HTML (Linguagem de Marcação de Hipertexto, ou *Hyper Text Markup Language*, em inglês) é o código utilizado para estruturar uma página web e seu conteúdo. Consiste em uma série de elementos utilizados para delimitar ou agrupar partes do conteúdo para que apareça ou atua de determinada maneira. Um elemento é composto por, respectivamente, uma *tag* de abertura, o conteúdo, e uma *tag* de fechamento, conforme ilustrado na figura 35.

Figura 35: Estrutura de um elemento HTML



Fonte: MDN Web Docs (2020c).

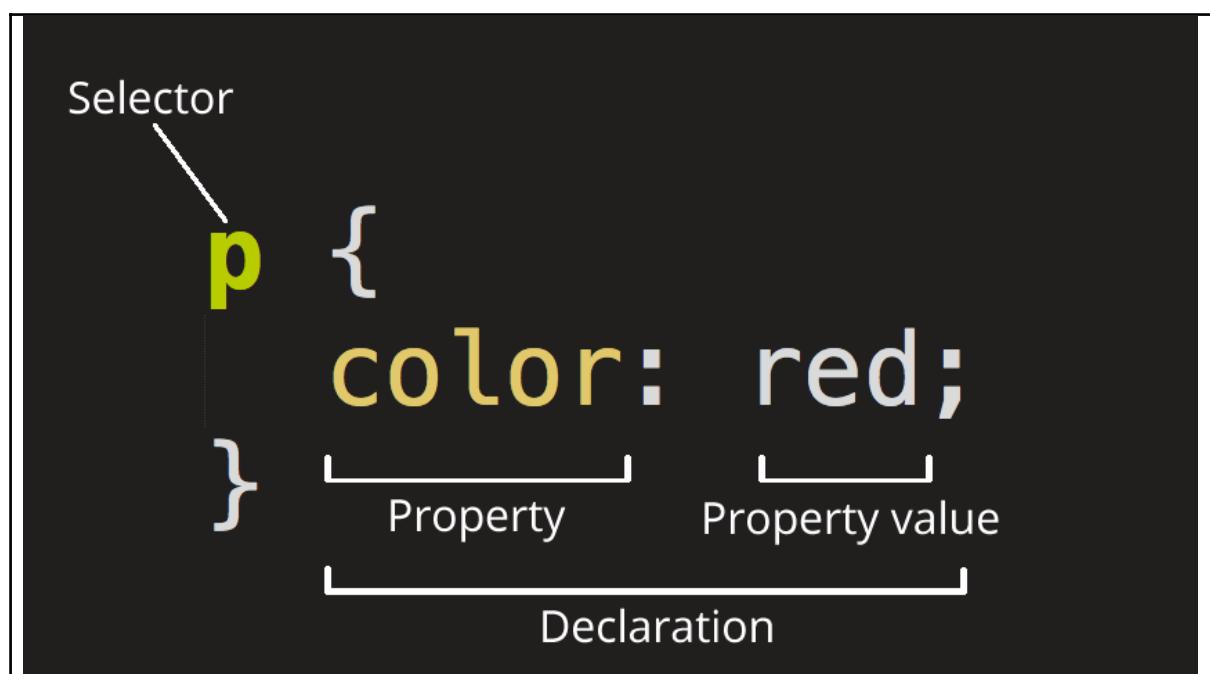
2.12.1.2 CSS

Segundo MDN Web Docs (2020a), CSS (*Cascade Style Sheet*, ou Folha de Estilo em Cascata, em português), é o código utilizado para estilizar uma página web. Assim como o HTML, CSS é uma linguagem de folhas de estilos que permite aplicar estilos seletivamente a elementos em documentos, como deixar a cor de um texto vermelha, por

exemplo. Para que um arquivo CSS atue sobre uma página web, ele deve ser aplicado ao documento HTML. Um arquivo CSS é composto uma série de conjunto de regras para definir o estilo de uma página web. Conforme apresentado na Figura 36, um conjunto de regras é definido por:

1. Seletor (*selector*): indica o elemento HTML ao qual o estilo será aplicado;
2. Declaração (*declaration*): define uma propriedade (*property*) e um valor (*property value*) que serão aplicados ao seletor;

Figura 36: estrutura de um conjunto de regras CSS



Fonte: MDN Web Docs (2020a).

2.12.1.3 JavaScript

JavaScript é uma linguagem de programação multi-paradigma utilizada principalmente do lado do cliente para criar páginas web dinâmicas, mas também é utilizado do lado do servidor, através da plataforma *Node.js*. Do lado do cliente, fornece interação com uma série de APIs, que permite que a linguagem manipule o conteúdo de páginas web, manipule dados, interaja com o dispositivo que executa o navegador, entre outras funcionalidades (MDN WEB DOCS, 2020e).

Em websites modernos, é comum que se recuperem dados individuais de um servidor para atualizar partes de uma página sem precisar recarregá-la completamente. Isso é possível através de uma técnica denominada Ajax (*Asynchronous JavaScript and*

XML) que permite o envio de requisições (*requests*) para um servidor, que deve retornar o conteúdo que será utilizado para atualizar parte de uma página web (MDN WEB DOCS, 2020b, tradução minha)

2.12.1.4 PHP

PHP é um acrônimo recursivo para Hypertext Preprocessor, trata-se de uma linguagem de script *open source* de uso geral, amplamente utilizada para desenvolvimento *web*. Assim como o *JavaScript* no lado do cliente, o PHP também permite aplicar lógica de programação em páginas HTML. Porém, no PHP, o processamento do PHP ocorre do lado do servidor, onde os scripts são interpretados e retornam o conteúdo de acordo com a definição do algoritmo, sem revelar a lógica da aplicação. (PHP, 2020a).

Para uso no desenvolvimento *web*, são necessários um interpretador PHP (CGI ou um módulo do servidor web), um servidor web e um navegador web. Esse método é sua aplicação mais comum, porém ainda é possível utilizar scrips PHP por linha de comando, sem necessidade de um servidor ou navegador envolvido, ou ainda escrever aplicações desktop. Apesar do PHP não ser a linguagem mais indicada para construção de aplicações desktop, através da extensão do PHP, PHP-GTK (PHP, 2020b).

Além disso, o PHP possui suporte a uma ampla variedade de banco de dados, bem como sockets e comunicação com diversos serviços através de protocolos como IMAP, SNMP, PHP3, HTTP, etc. Esses e outros recursos são disponibilizados através de extensões instaláveis. Vale frisar que o PHP não se limita ao processamento de páginas HTML. Com a linguagem é possível gerar imagens, arquivos PDF, XML, manipular o sistema de arquivos, e incontáveis demais funcionalidades (PHP, 2020b).

Segundo PHP (2020a) a estrutura fundamental de uma página ou trecho de código PHP são diretivas que marcam o início e final das instruções, como exemplificado na Figura 37.

Figura 37: Exemplo de um bloco de código PHP entre código HTML.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Exemplo</title>
  </head>
  <body>

    <?php
      echo "Olá, eu sou um script PHP!";
    ?>

  </body>
</html>
```

Fonte: PHP (2020a).

2.12.1.5 Frameworks

“Um *framework* é uma estrutura genérica estendida para se criar uma aplicação ou subsistema mais específico”. (SOMMERVILLE, 2011, p. 300). De acordo com MDN Web Docs (2020f, tradução minha), do lado do servidor, frameworks web fornecem ferramentas e bibliotecas para simplificar operações comuns no desenvolvimento *web*. A adoção de um *framework* para desenvolvimento não é obrigatória, mas é altamente recomendada, uma vez que simplifica tarefas do dia-a-dia de um desenvolvedor. Entre as funções frequentemente fornecidas por um *framework*, pode-se citar: encapsulamento de conexões HTTP e interpretação dos métodos (GET, POST, PUT, PATCH, DELETE, etc.), roteamento de requisições, abstração e simplificação do acesso ao banco de dados e renderização de dados.

2.12.1.5.1 Lumen

De acordo com Lumen.laravel.com (2020b, tradução minha), Lumen é uma framework PHP derivada da popular *framework* MVC Laravel, desenvolvida e otimizada especialmente para construção de micro-serviços e API's altamente velozes e elegantes. A *framework* apresenta estrutura extremamente semelhante e as mesmas funcionalidades apreciadas no Laravel, como Eloquent, Cache, validações, rotas, *migrations*, *middlewares* e afins, porém de forma minimalista. Isso porque Laravel inclui diversas funcionalidades

para construção de aplicações MVC que não se fazem necessárias quando se está construindo uma API.

Os requisitos para uso do Lumen em sua versão mais recente, 8.x, são, segundo Lumen.laravel.com (2020a, tradução minha): PHP versão igual ou superior a 7.3; Extensão PHP OpenSSL, extensão PHP PDO e extensão PHP Mbstring. Além disso, a *framework* utiliza o software Composer para gerenciamento de dependências, desse modo, esse é mais um dos requisitos.

2.12.1.5.1.1 Eloquent ORM

De acordo com Laravel.com (2020b, tradução minha), o Eloquent ORM incluso no Laravel é uma biblioteca que fornece uma implementação simples de *ActiveRecord* para interação com o banco de dados. Cada tabela do banco de dados possui um *Model* correspondente, que deve ser usado para interação com a tabela. A biblioteca utiliza uma série de convenções a fim de agilizar e facilitar a interação com o banco de dados na aplicação. Para desfrutar dessas funcionalidades, a classe *Modelo* deve estender a classe *Model* da biblioteca.

Por padrão, os nomes dos *Models* devem ser apresentados no singular, com as palavras iniciadas em letra maiúscula e sem separação por qualquer caractere. Essa nomenclatura é interpretada e convertida para o nome da tabela no banco de dados, que será escrita no plural, com todas as letras minúsculas e palavras separadas por *underline* (_). As chaves primárias são por padrão denominadas *id*. Além disso, a biblioteca também fornece um padrão para determinar os horários de criação e atualização dos registros, correspondendo aos campos *created_at* e *updated_at*. Esses padrões podem ainda ser customizados através de sobrescrita de métodos e atributos da classe mãe. (LARAVEL.COM, 2020b, tradução minha)

2.12.1.5.1.2 Migrations

Outra funcionalidade fornecida pelo Laravel são as *Migrations*. Pode-se dizer que são equivalentes a controle de versão de uma base de dados, fornecendo uma ferramenta para criação e alteração dos esquemas de banco de dados através de código. Isso permite à equipe de desenvolvimento alterar a base de dados e compartilhar facilmente essas alterações através do próprio código-fonte da aplicação, dispensando a

necessidade de compartilhamento de códigos SQL sempre que for efetuada alguma modificação no banco de dados. (LARAVEL.COM, 2020a, tradução minha).

2.12.1.5.2 Express

NodeJs (ou, somente *Node*, com menos formalidade) é um ambiente em tempo de execução e multiplataforma que permite a criação de aplicativos e ferramentas do lado do servidor sob linguagem *JavaScript*. Entre as principais vantagens oferecidas, destacam-se: performance excelente, código escrito em *JavaScript* (dispensa necessidade de aprendizado de uma outra linguagem para o *backend*), acesso a diversos pacotes (disponíveis no NPM), compatível com diversos sistemas operacionais, comunidade e ecossistemas muito ativos (MDN WEB DOCS, 2020d).

Express é o framework Node mais popular e a biblioteca originária de uma série de outros frameworks Node. É bastante minimalista, em contrapartida oferece bastante flexibilidade aos desenvolvedores para modelar as soluções. Além disso, fornece uma série de bibliotecas para trabalhar com cookies, sessões, login de usuários, parâmetros de URL, verbos HTTP, cabeçalhos, etc (MDN WEB DOCS, 2020d).

2.12.1.5.3 VueJS

Conforme Galdino (2020) é um *framework* progressivo e flexível, baseado na linguagem *JavaScript*, para a construção de interfaces de usuário. É ideal para prototipagem rápida pois oferece uma maneira fácil e flexível de ligação de dados reativos e componentes reutilizáveis, mas também pode suportar aplicações web complexas e escaláveis. Além disso, fornece boa integração com as diversas bibliotecas *JavaScript*.

De acordo com Picollo (2020), a arquitetura de uma aplicação construída sob *VueJS* é baseada em componentes criados com a sintaxe de HTML, CSS e *JavaScript* em um único arquivo de extensão .vue, o que facilita o isolamento e a manutenção de funcionalidades. Assim, cada componente constitui um escopo isolado dos demais, tanto em lógica quanto nos estilos.

2.13 RESTFUL WEBSERVICES

Para entender o que é um RESTful *Web Service*, é necessário, primeiramente, compreender o que é um *web service*. De acordo com Breitman (2005), não há uma definição única para o termo *Web Service*, porém a autora destaca algumas definições consideradas mais esclarecedoras. Com base na identificação de pontos comuns entre elas, pode-se afirmar que um *web service* é um tipo de aplicação *web*, identificado através de uma URI, baseado no padrão XML, que fornece interfaces através das quais outras aplicações podem utilizar ou invocar os métodos e funcionalidades implementadas.

Segundo Salvadori (2015), um *web service* pode ser implementado através do padrão SOAP (*Simple Object Access Protocol*), que utiliza notação XML para a troca de informações; ou através de REST, que em sua maioria utiliza JSON para a troca de mensagens. Dessa forma, pode-se dizer que em uma definição mais recente, um *web service* não opera restritivamente sob XML, na verdade define um formato padrão para a comunicação com as aplicações clientes.

Por fim, De acordo com Machado, Franco e Prestes (2016), REST (Representational State Transfer) é uma abordagem para o desenvolvimento de aplicações com baixo acoplamento, escalabilidade e simplicidade arquitetural, viabilizada pela adoção de um conjunto de restrições. As principais restrições REST são:

“As aplicações devem respeitar o modelo cliente/servidor; devem existir intermediários entre o cliente e o servidor e, nesses mediadores, ser possível a utilização de cache, por exemplo, o proxy do HTTP; qualquer tipo de recurso deve ser definido por meio de *Uniform Resource Identifiers* (URIs), da mesma forma como acontece com os links na Web”; deve-se delimitar um conjunto de métodos, por exemplo, HTTP GET, POST, PUT, etc.; não se deve possuir um estado para a aplicação (*stateless*) (MACHADO; FRANCO; PRESTES, 2016, p. 193).”

Recomenda-se o uso de uma abordagem REST quando um serviço necessita ser incorporado em uma página Web, através de Ajax, por exemplo.

2.14 AMBIENTE

Conforme discutido no capítulo 2.3, as ferramentas compõem uma das camadas da engenharia de software e dão apoio aos processos e métodos. De acordo com Sommerville (2011), uma plataforma de desenvolvimento é o conjunto formado por hardware, sistema operacional e demais softwares de apoio, onde um software é desenvolvido. Pode incluir, por exemplo, sistemas gerenciadores de banco de dados, editores de código, compiladores; ferramentas de teste, ferramentas gráficas para

construção de diagramas, ferramentas de depuração e ferramentas para organização de código, por exemplo.

2.14.1 MySQL Workbench

MySQL Workbench é uma ferramenta gráfica para trabalhar com bancos de dados e servidores MySQL, mantida pela mesma mantenedora do MySQL Server, a Oracle. Oferece versões para os sistemas operacionais Windows, Linux e MacOS. Inclui funcionalidades de apoio ao desenvolvimento, modelagem, administração e migração de bancos de dados.

Permite o gerenciamento de conexões a servidores MySQL, onde podem ser executadas consultas diretamente de seu editor SQL embutido (Figura 38). Fornece uma interface gráfica para criação de modelos ER (Figura 39), engenharia reversa e direta, bem como uma interface simples para criação e edição de tabelas e suas propriedades. Possibilita também a migração a partir de outros SGBDs, como Microsoft SQL Server e PostgreSQL, ou a partir de versões mais antigas do MySQL para versões mais recentes.

Inclui também funções de administração de servidor, como gerenciamento de usuários, backups e restaurações, inspeção de dados de auditoria, visualização da saúde do banco de dados e monitoramento de performance do servidor MySQL. O MySQL Workbench é distribuído em duas versões: a Community Edition (gratuita) e a versão Comercial, a qual inclui suporte a serviços avançados de backup, *firewall* e auditoria (MYSQL, 2020, tradução minha).

Figura 38: Editor SQL Embutido do MySQL Workbench

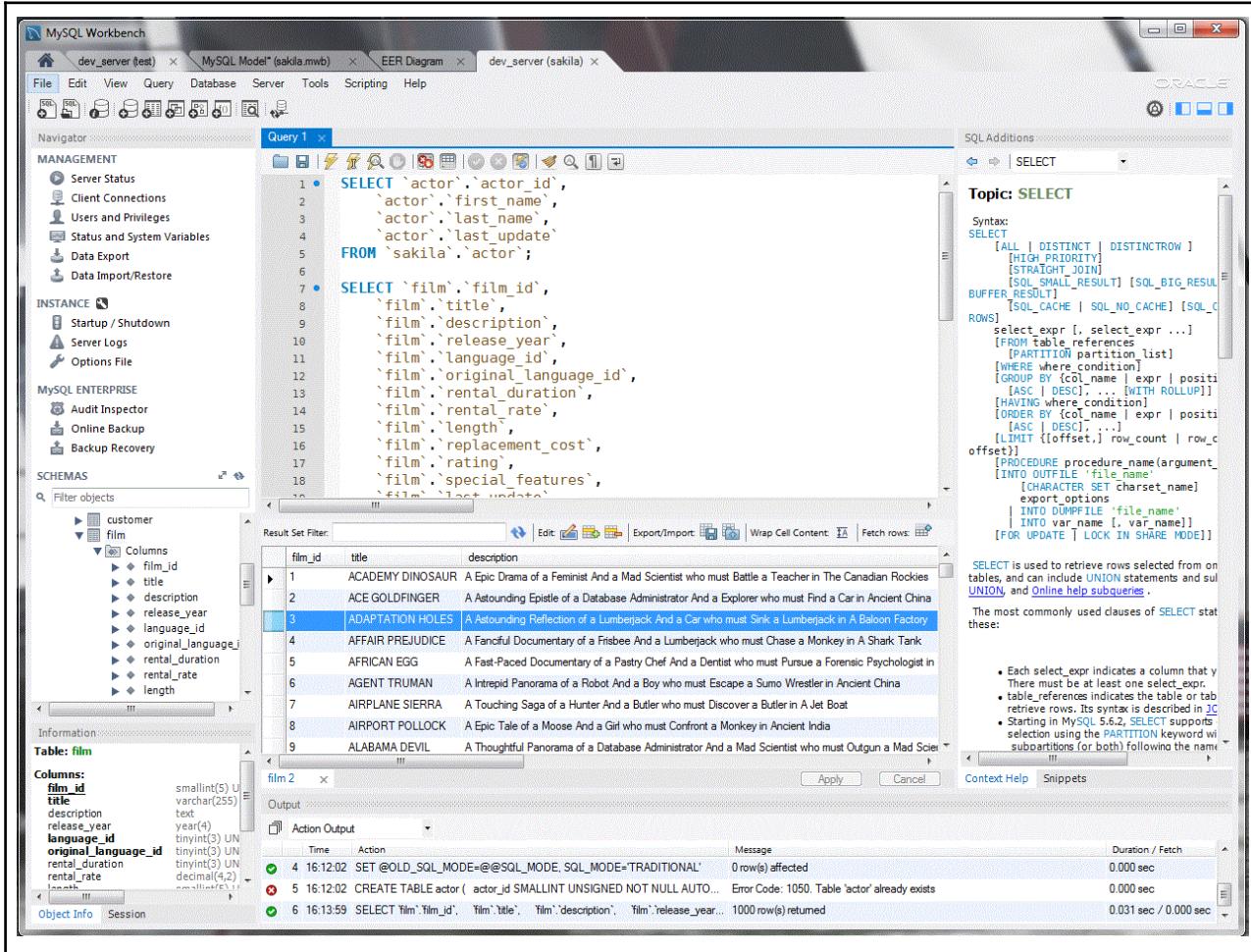
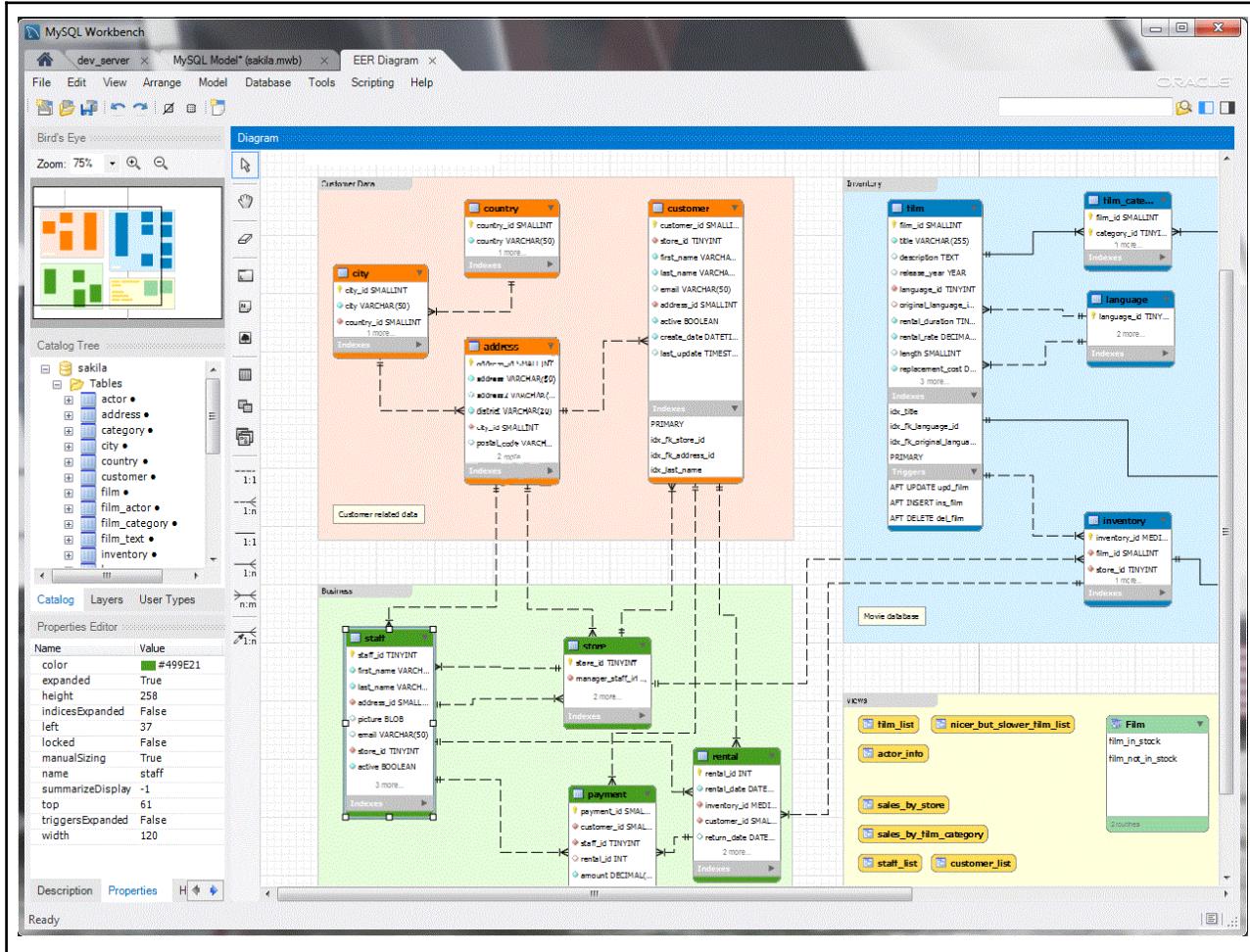


Figura 39: Interface para criação de modelos ER do MySQL Workbench



Fonte: (MYSQL, 2020)

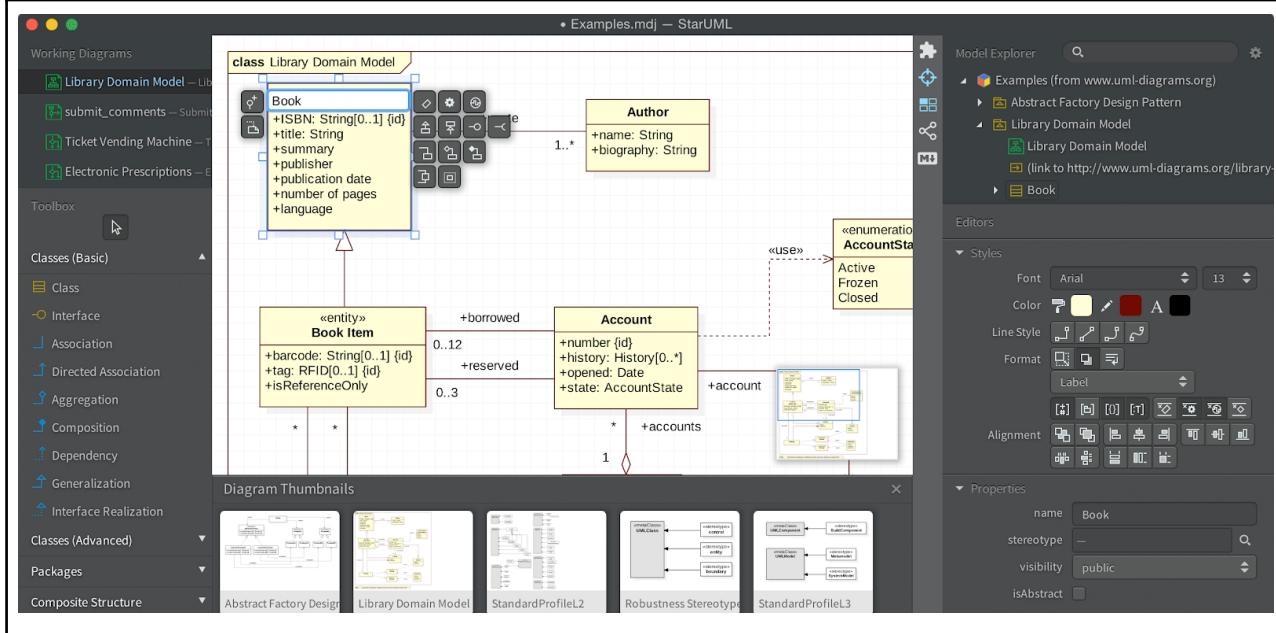
2.14.2 StarUML

StarUML é uma ferramenta gráfica para modelagem de diagramas, em especial UML. Oferece suporte aos padrões da versão 2 da Linguagem UML e seus diagramas, além de diagramas adicionais como Entidade-Relacionamento, Diagrama de Fluxo de Dados e Fluxograma. A Figura 40 apresenta a interface do software.

Os diagramas desenvolvidos podem ser exportados para PDF, HTML ou imagem. O software permite também que os diagramas desenvolvidos sejam agrupados em um pacote com todos os demais diagramas de um projeto, salvando em um arquivo de extensão .mdj. Além disso, é facilmente integrável com extensões de terceiros para execução de outras tarefas especializadas, cujo desenvolvimento é viabilizado através de uma API aberta. O software está disponível para as plataformas Linux, MacOs e

Windows. (STARUML.IO, 2020, tradução minha).

Figura 40: Interface do StarUML



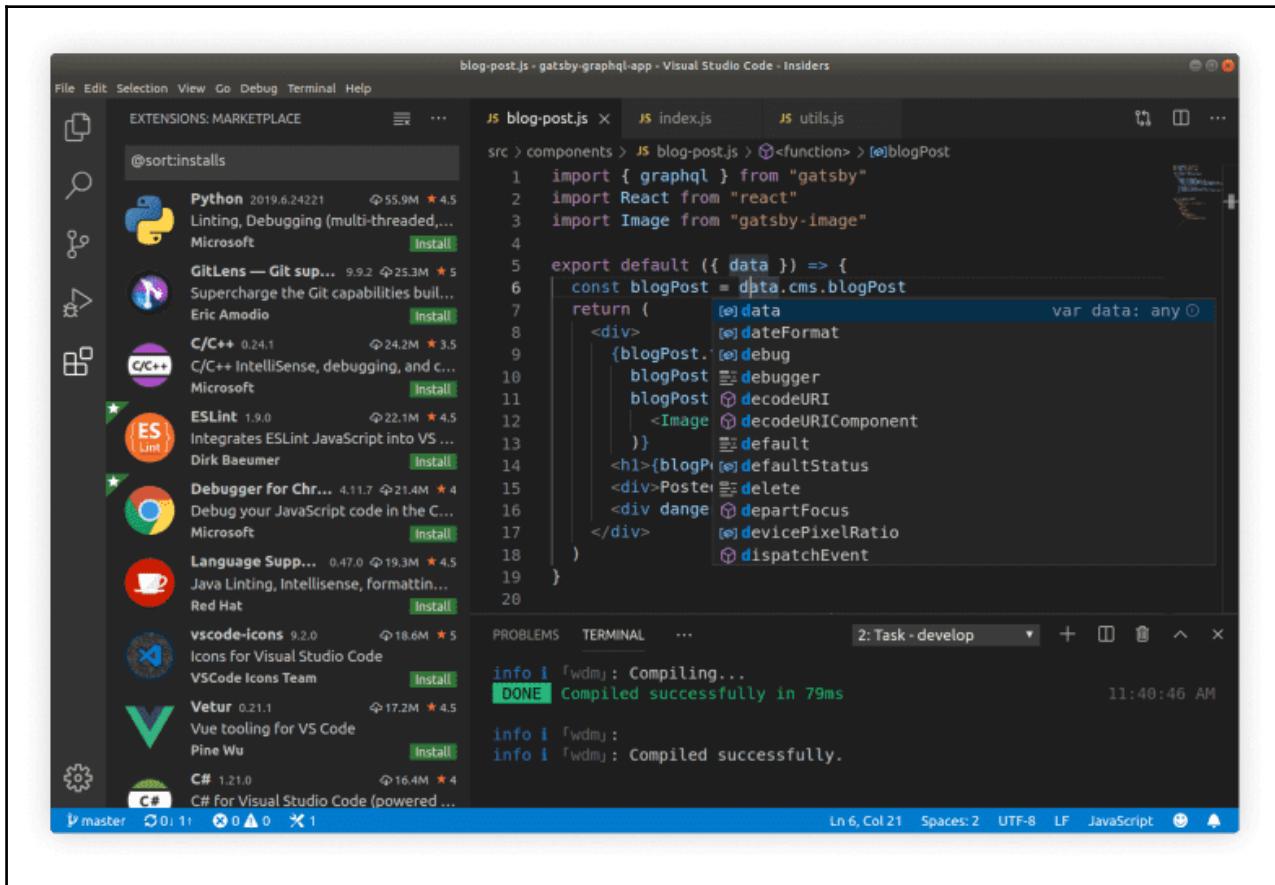
Fonte: (STARUML.IO, 2020)

2.14.3 Visual Studio Code

Visual Studio Code (ou VS Code), é um editor de código-fonte fornecido pela Microsoft, desenvolvido para as plataformas desktop Windows, MacOS e Linux. Oferece suporte nativo para JavaScript, TypeScript e NodeJs, mas permite a adição de suporte para outras linguagens através da instalação de extensões. Além disso, o software apresenta alto índice de customização através da sua biblioteca de extensões, que podem incluir indentadores de código, revisores de estilo de código-fonte, atalhos de teclado, depuradores, conexão remota com janelas do editor rodando em outro computador para edição de código remoto, etc. (CODE, 2020)

A interface do software apresentada na Figura 41 mostra na coluna à esquerda a biblioteca de extensões, onde as extensões disponíveis podem ser encontradas e instaladas sem precisar sair da plataforma. No lado direito, na parte inferior, pode-se observar um terminal integrado, que permite a execução de linhas de comando. E, por fim, na parte superior do lado direito, encontra-se o editor de código-fonte em si.

Figura 41: Interface do editor VS Code



Fonte: (CODE, 2020)

3 MATERIAL E MÉTODOS

Este trabalho caracteriza-se como uma pesquisa qualitativa, na qual apresenta-se o desenvolvimento de um protótipo de aplicativo multiplataforma que permite a busca e divulgação de produtores locais de alimentos. Tendo como base o notável aumento pela busca por hábitos alimentares mais saudáveis, bem como o fomento da economia local, considerando o uso de software como potenciais soluções para os mais diversos problemas encontrados em uma sociedade, mais especificamente o uso de aplicativos como solução para problemas pontuais.

3.1 PESQUISA BIBLIOGRÁFICA

Buscando compreender os domínios dos problemas em comum enfrentados na busca por hábitos alimentares mais saudáveis e pelo fomento da economia local, iniciou-se uma pesquisa bibliográfica com o objetivo de compreender o cenário atual relacionado a estes dois assuntos, e de que forma o uso de software poderia preencher as lacunas identificadas.

Na sequência, foi realizada também uma pesquisa bibliográfica acerca dos assuntos relacionados ao desenvolvimento de software e especificamente de aplicativos, a fim de construir uma base de conhecimento sólida para o desenvolvimento da solução consistente e efetiva.

3.2 IDENTIFICAÇÃO DE STAKEHOLDERS

A partir da escolha de um problema específico para propor uma solução, com base nos resultados da pesquisa bibliográfica acerca da problemática, foram identificados os *stakeholders*, isto é, os indivíduos impactados pela solução proposta.

3.3 ANÁLISE DE REQUISITOS

Como a proposta da solução baseou-se somente na pesquisa bibliográfica, buscou-se identificar requisitos mínimos funcionais e não funcionais para a construção de um protótipo com funcionalidades genéricas que atendam às necessidades dos *stakeholders* dentro dos domínios do problema.

3.4 PLANEJAMENTO

Avaliando os requisitos, planejou-se a solução, adotando para o desenvolvimento do protótipo o modelo orientado a cronograma, tendo em vista os prazos do projeto, a forma de elicitação de requisitos e a característica de protótipo atribuída à proposta de solução.

Na primeira etapa, será desenvolvido um modelo de alto nível, onde serão desenvolvidos os diagramas de caso de uso e de classe. Com base no documento de requisitos, juntamente do Diagrama de Caso de Uso e na compreensão das dependências e criticidade de cada caso de uso, será possível ordenar por prioridade os conjuntos funcionais que deveriam compor o protótipo. Dessa forma, cada uma das iterações dedicar-se-á a entregar um artefato de software que atenda ao caso de uso mais prioritário ainda não tratado.

Cada iteração responsável pela entrega de uma funcionalidade correspondente a um caso de uso inclui, em respectiva ordem, as atividades de modelagem, construção e entrega, compostas pelas ações de:

1. Modelagem: especificação do caso de uso e desenvolvimento de diagrama de sequência correspondente ao caso de uso;
2. Construção: documentação dos casos de teste, codificação e execução dos casos de teste;
3. Entrega: testes de integração e entrega do incremento.

Ao final das iterações, pretende-se obter um protótipo funcional, possibilitando disponibilizá-lo para testes em ambiente semelhante ao de produção, com usuários reais, que forneçam *feedbacks* a respeito do produto auxiliando para sua melhoria.

3.5 MODELAGEM

Para desenvolvimento dos diagramas mencionados a seguir, será utilizado o software StarUML, devido a amplitude dos recursos voltados à linguagem UML, sendo esse seu foco principal.

3.5.1 Diagrama de Casos de Uso

Após o levantamento de requisitos e planejamento, será desenvolvido o diagrama de caso de uso, baseado nos requisitos funcionais. Com isso, será possível visualizar quais funcionalidades deveriam compor o protótipo.

A partir da primeira iteração, o caso de uso correspondente será especificado e descrito com o uso do editor de texto Google Docs, fornecendo informações de como espera-se que ocorra a interação com o protótipo.

3.5.2 Diagrama de Classes

Visando identificar as classes e respectivos atributos e funções, necessários para o funcionamento da aplicação, será desenvolvido o diagrama de classes. O diagrama fornece também uma visão do modelo relacional, onde os atributos correspondem aos atributos, as classes às entidades e as relações entre classes às relações entre entidades. O diagrama pode ser adaptado conforme necessário nas iterações seguintes.

3.5.3 Diagrama de Sequência

Com base nos artefatos da primeira iteração, somado à descrição do caso de uso, será desenvolvido o diagrama de sequência. Com isso, cria-se uma visão do fluxo de processos que devem ser codificados para a viabilização do caso de uso.

3.6 CONSTRUÇÃO

A construção, realizada a cada iteração responsável pela entrega de um incremento correspondente a um caso de uso, incorpora três principais ações, em respectiva ordem: identificação e documentação dos casos de teste, construção e execução dos casos de teste.

3.6.1 Documentação dos Casos de Teste

Sabendo-se do comportamento esperado pelo sistema, informação fornecida pelos artefatos produzidos até o momento, os casos de teste são identificados e documentados,

para que ao final da construção do caso de uso os mesmos sejam executados a fim de garantir o correto atendimento do caso de uso. Para a documentação, será utilizado o editor de texto Google Docs.

3.6.2 Codificação

Para a codificação, optou-se por construir a aplicação utilizando arquitetura cliente-servidor, separada em três camadas: cliente, aplicação e dados.

O papel do cliente é desempenhado por uma aplicação construída com o uso das tecnologias HTML e CSS, sob a framework VueJs, a qual é baseada na linguagem de programação JavaScript. Aplica-se também à camada de cliente, o conceito de Progressive Web App, tecnologia que permite que a mesma aplicação seja acessada tanto através de um navegador web, como seja instalada nos dispositivos e se comporte de maneira semelhante a um aplicativo nativo. O uso de VueJs na camada de cliente auxilia a implementação do conceito de PWA. A arquitetura escolhida para a construção da aplicação cliente foi a arquitetura MVC.

A camada de aplicação será centralizada em um *webservice*, construído utilizando a linguagem de programação JavaScript, sob a *framework* Express, aplicando o padrão API REST em uma arquitetura de transação.

Por fim, a responsabilidade da camada de dados é desempenhada por um banco de dados PostgreSQL, cuja estrutura é definida no código da camada de aplicação, com o uso da biblioteca JavaScript Knex. Optou-se por utilizar essa biblioteca pois fornece uma camada de abstração entre a aplicação e a base de dados, permitindo que a mesma estrutura seja utilizada em diversas plataformas de banco de dados sem a necessidade de construção de scripts DDL específicos para cada tipo de banco de dados. Dessa forma, é possível migrar a estrutura da base de dados da aplicação sem grandes complicações, visto que está definida em um código comum aplicável a diferentes tipos de bases de dados, além de permitir o versionamento da estrutura.

Definidos os papéis, a codificação da solução será realizada na seguinte sequência:

1. Quando necessário, codificação da estrutura da base de dados necessária para o caso de uso, utilizando *migrations* da biblioteca JavaScript Knex;

2. Quando necessário, codificação dos recursos REST que serão consumidos no caso de uso;
3. Codificação da interface gráfica através da qual o cliente poderá efetivar o caso de uso, bem como as funções que deverão consumir recursos da API REST.

3.6.3 Execução dos Casos de Teste

Serão executados casos de teste conforme descritos na documentação, para garantia de que o recurso foi implementado com sucesso. Caso a execução do teste tenha identificado algum problema, retoma-se a ação de construção, buscando identificar e solucioná-lo.

3.7 ENTREGA

Na etapa de entrega, serão realizados os testes de integração, incluindo o novo incremento de software. Caso sejam identificados erros nos testes de integração, retoma-se a ação de construção, buscando identificá-lo e corrigi-lo. Caso contrário, o incremento de software é entregue e obtém-se uma versão mais completa do protótipo.

4 RESULTADOS E DISCUSSÕES

Nessa etapa demonstram-se os procedimentos realizados para a conclusão deste estudo, considerando diretamente os objetivos apresentados no capítulo introdutório do presente trabalho.

4.1 PESQUISA

A pesquisa bibliográfica teve seu início em março de 2020, na cadeira de Trabalho de Conclusão de Curso I. O tema havia sido previamente definido no final do ano de 2019, porém até o início efetivo da pesquisa fora realizado apenas especulações acerca do assunto. Ainda era necessário coletar material para identificar o tema central e os problemas a serem abordados, bem como delimitar os métodos e tecnologias a serem usadas na solução.

A pesquisa relacionada ao tema escolhido, Locavorismo, baseou-se principalmente em artigos científicos e publicações em revistas, através da qual identificou-se o problema onde a aplicação de um software poderia impactar positivamente e as principais características do movimento que guiaram o levantamento de requisitos, junto de conhecimentos prévios sobre softwares semelhantes. Além disso, o estudo de métodos e tecnologias permitiu a identificação e delimitação da abordagem para planejamento e construção da solução.

4.2 IDENTIFICAÇÃO DOS STAKEHOLDERS

Com base na pesquisa bibliográfica, identificou-se a dificuldade de conectar produtores locais e consumidores como uma das lacunas existentes dentro do movimento Locavorista que dificulta sua expansão. Desse modo, o produtor local foi identificado como principal *stakeholder* por ser diretamente beneficiado pela aplicação, que possibilitaria a divulgação de seu perfil e produção. O consumidor interessado em adquirir produtos de procedência local também pode ser considerado um *stakeholder*, visto que a aplicação necessita do engajamento de ambas as partes para atingir seus objetivos: a divulgação e busca de produtores locais.

4.3 ANÁLISE DE REQUISITOS

O levantamento e análise de requisitos foi exclusivamente baseado na pesquisa bibliográfica do tema, sem consulta aos *stakeholders*. Portanto, os requisitos funcionais e não funcionais elicitados estão relacionados a requisitos genéricos de aplicações de software e ao que identificou-se serem requisitos para cumprimento dos objetivos dentro dos domínios do tema abordado.

Dessa forma, os quadros a seguir apresentam os requisitos funcionais e não funcionais identificados.

Quadro 5: Requisitos funcionais

R.F	Descrição
RF1	O sistema deve permitir o cadastro de usuários ou login por Facebook (<i>login social</i>) com os seguintes dados: nome, e-mail e senha.
RF2	O sistema deve permitir adicionar uma foto ou usar a foto do Facebook (quando realizado <i>login social</i>)
RF3	Para produtores, o sistema deve permitir informar os dados de contato como telefone, celular, <i>whatsapp</i> , etc.
RF4	Para produtores, o sistema deve permitir incluir uma descrição de apresentação ao perfil
RF5	Os produtos devem ser categorizados por seu método de produção: orgânicos, convencionais, coloniais, artesanais, etc
RF6	O aplicativo deve permitir que consumidores adicionem produtores aos seus favoritos
RF7	Um usuário produtor tem acesso às mesmas funcionalidades de um usuário consumidor
RF8	A diferença entre um usuário consumidor de um usuário produtor é uma opção que pode ser habilitada no cadastro do usuário
RF9	Para um usuário ser classificado como produtor e aparecer nas buscas, ele deve concluir seu cadastro incluindo contato, descrição e endereço.
RF10	O aplicativo deve permitir que o usuário crie uma lista de palavras-chave de interesse, para então localizar produtos e produtores compatíveis. Ex: maçã, abacate.
RF11	Ao entrar no aplicativo, caso o usuário já tenha definido seus produtores favoritos, deverá ser apresentada a lista. Caso já tenha definido seus interesses, deverá ser exibida uma lista de produtores e produtos compatíveis com seus critérios. Deve ser mostrado o produto, os produtores e a distância em que se encontram.
RF12	Caso o usuário não tenha cadastrado favoritos e palavras-chave de

	interesse, deverá ser exibido um aviso para selecionar seus favoritos.
RF13	No resultado da busca, deverão existir botões para adicionar/excluir o produtor aos favoritos e ver detalhes.
RF14	O aplicativo deve oferecer acesso alteração de dados do cadastro, alteração de foto, alteração de senha, alteração dos produtos que deseja vender, esquecer os dados e sair.
RF15	Uma opção para visualizar o resultado da busca através de mapa deverá estar disponível
RF16	O sistema deve permitir aos produtores cadastrar seus produtos informando nome, foto(s), descrição, categoria e palavras-chave
RF17	O cadastro de produtos não é obrigatório para que o produtor apareça nos resultados da busca
RF18	O produtor pode associar palavras-chave ao seu perfil.
RF19	O aplicativo deve permitir aos usuários consumidores avaliar os produtos com nota de 1 a 5, podendo informar descrição
RF20	O aplicativo deve permitir aos usuários consumidores avaliar os produtores com nota de 1 a 5, podendo informar descrição
RF21	A média das avaliações determinará a reputação do produtor e do produto e deve ser exibida nos detalhes dos itens
RF22	Para realizar a busca de produtos/produtores, o usuário não precisa estar cadastrado
RF23	O sistema deve conter uma sessão para sugestões aos desenvolvedores, como forma de solicitar funcionalidades que julga relevantes e fornecer feedback.
RF24	O sistema deve fornecer uma interface para o usuário buscar produtos e produtores
RF25	O sistema deve fornecer uma interface para que o produtor gerencie seus endereços (pode haver mais de um)
RF26	Cada endereço do produtor pode conter seus próprios canais de contato
RF27	É obrigatório que o produtor cadastre ao menos um endereço
RF28	Um produtor pode enviar imagens e anexá-las aos detalhes de seu perfil, além de sua foto de perfil

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 6: Requisitos não funcionais

R.N.F.	Descrição
RNF1	O sistema deve ser apresentado em uma interface amigável para dispositivos móveis
RNF2	A localização do usuário deve se basear na geolocalização do dispositivo

	de acesso.
RNF3	As senhas armazenadas devem estar criptografadas

Fonte: (DADOS DA PESQUISA, 2020)

4.4 PLANEJAMENTO

O processo de software adotado para o desenvolvimento foi o modelo orientado a cronograma, dados os prazos, riscos envolvidos, pouco conhecimento do ambiente de aplicação do projeto e a característica de protótipo atribuída à solução proposta. Os requisitos funcionais e não funcionais contribuíram na definição da arquitetura do software e tomada de decisão de linguagens de programação e *frameworks* a serem utilizadas na construção. Feito isso, o desenvolvimento foi dividido em iterações, onde as primeiras foram dedicadas à modelagem do protótipo, gerando diversos artefatos de documentação. As iterações seguintes dedicaram-se à codificação e testes do protótipo. Vale ressaltar que a cada iteração, com a descoberta de detalhes sobre o projeto, mantém-se aberta a possibilidade de retomada e adequação da documentação.

4.5 ARQUITETURA, LINGUAGENS E FRAMEWORKS

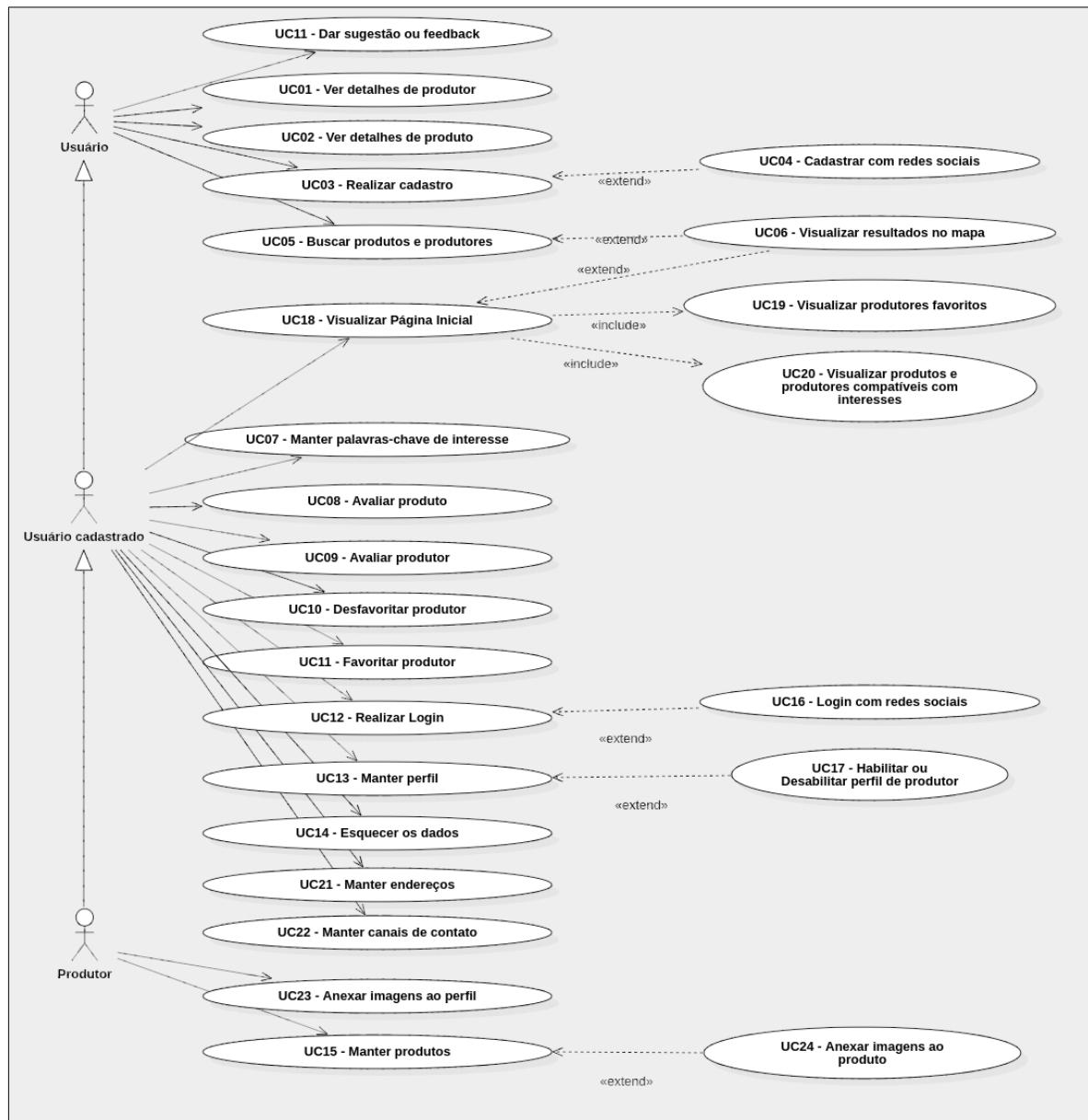
A arquitetura da aplicação foi dividida em três camadas: cliente, aplicação e dados. A camada de cliente foi desenvolvida com uso da *framework* VueJs, baseada na linguagem de programação JavaScript, junto de HTML e CSS. A camada de aplicação, por sua vez, foi implementada como uma API REST, codificada em PHP, sob a *framework* Lumen. Por fim, utilizou-se banco de dados MySQL na camada de dados.

4.6 MODELAGEM

4.6.1 Diagrama de Caso de Uso

O diagrama de caso de uso fornece uma visão de alto nível das funcionalidades que compõem o sistema, sendo inteligível inclusive para clientes e usuários sem conhecimento de desenvolvimento de software e artefatos de documentação mais avançados. Serve também como base para o desenvolvimento do restante da documentação do sistema. O diagrama de caso de uso obtido está expresso na Figura 42

Figura 42: Diagrama de Caso de Uso



Fonte: (DADOS DA PESQUISA, 2020)

Para cada caso de uso foi desenvolvida uma descrição, indicando o comportamento esperado pela funcionalidade e demais informações relacionadas ao caso de uso. As descrições de caso de uso estão dispostas nos quadros a seguir.

Quadro 7: Descrição do Caso de uso UC01

Caso de uso	UC01 - Ver detalhes de produtor
Descrição	Fornece ao usuário a visualização de detalhes do perfil do

	produtor.
Pré-condições	O usuário clicou sobre o perfil do produtor nos detalhes da busca, em sua tela inicial ou na lista de produtores favoritos.
Fluxo básico	<p>1. O usuário clica sobre o perfil do produtor</p> <p>2. O sistema exibe uma tela com os detalhes do produtor, exibindo, além de um botão para favoritar produtor, as informações cadastradas:</p> <ol style="list-style-type: none"> 1. Nome; 2. Foto de perfil; 3. Descrição; 4. Imagens; 5. Categorias de produtos que produz; 6. Tags; 7. Endereço(s) e respectivos contato(s); 8. Avaliação média; 9. Avaliações; 10. Produtos

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 8: Descrição do Caso de uso UC02

Caso de uso	UC02 - Ver detalhes de produto
Descrição	Fornece ao usuário a visualização de detalhes do produto selecionado.
Pré-condições	O usuário clicou sobre o produto nos detalhes da busca, em sua tela inicial ou no perfil do produtor.
Fluxo básico	<p>3. O usuário clica sobre a miniatura do produto;</p> <p>4. O sistema exibe uma tela com os detalhes do produto:</p> <ol style="list-style-type: none"> 1. Nome; 2. Descrição; 3. Categoria; 4. Valor; 5. Imagens;

	<p>6. Tags;</p> <p>7. Produtor;</p> <p>8. Avaliação média;</p> <p>9. Avaliações.</p>
--	--

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 9: Descrição do caso de uso UC03

Caso de uso	UC03 - Realizar Cadastro
Descrição	Fornece uma interface que permite ao usuário realizar o cadastro na aplicação.
Pré-condições	O usuário não pode estar logado.
Fluxo básico	<p>5. Na tela de login, o usuário seleciona a opção “Ainda não sou cadastrado”;</p> <p>6. O sistema exibe um formulário de cadastro, contendo campos para: nome, e-mail, senha e confirmação de senha;</p> <p>7. O usuário preenche os campos no formulário;</p> <p>8. O usuário clica no botão “cadastrar”;</p> <p>9. O sistema valida se a senha e a confirmação de senha coincidem;</p> <p>10. O sistema verifica se o e-mail informado já está associado a algum usuário;</p> <p>11. O e-mail informado ainda não está cadastrado;</p> <p>12. O sistema grava as informações do usuário, finalizando o cadastro;</p> <p>13. O sistema redireciona o usuário para a tela inicial.</p>
Fluxos alternativos e exceções	<p>Cenário alternativo 1:</p> <p>6. O sistema identifica que o valor informado no campo de senha não coincide com o valor informado no campo de confirmação;</p> <p>7. O sistema apresenta uma mensagem informando que os campos não coincidem.</p>

	<p>Cenário alternativo 2:</p> <p>7. O e-mail informado já está associado a um usuário;</p> <p>8. O sistema apresenta uma mensagem informando que o e-mail informado já está em uso.</p>
Pós-condições	O usuário possui cadastro e está logado no sistema.

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 10: Descrição do Caso de uso UC04

Caso de uso	UC04 - Cadastrar com redes sociais
Descrição	Fornece uma interface que permite ao usuário realizar o cadastro na aplicação com seu perfil no Facebook ou Google.
Pré-condições	Não estar logado no sistema.
Fluxo básico	<p>1. O usuário acessa o sistema;</p> <p>2. O sistema exibe uma tela onde o usuário pode fazer login ou cadastrar-se;</p> <p>3. O usuário opta pela opção de cadastro;</p> <p>4. O usuário cobra-se usando Facebook;</p> <p>5. O sistema coleta os dados fornecidos pela API do Facebook;</p> <p>6. O sistema finaliza o cadastro do usuário;</p> <p>7. O sistema redireciona o usuário para a tela inicial.</p>
Fluxo alternativo de eventos	<p>4. O usuário se cobra usando sua conta Gooogle;</p> <p>5. O sistema coleta os dados fornecidos pela API do Google</p> <p>6. O sistema finaliza o cadastro do usuário;</p> <p>7. O sistema redireciona o usuário para a tela inicial.</p>
Exceções no fluxo normal de eventos	No passo 4 o usuário não possui conta na plataforma selecionada. Desse modo, não é possível realizar o cadastro
Pós-	O usuário possui cadastro e está logado no sistema.

condições	
-----------	--

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 11: Descrição do Caso de uso UC05

Caso de uso	UC05 - Buscar produtos e produtores
Descrição	Permite ao usuário buscar produtos e produtores com base em palavras-chave.
Pré-condições	Nenhuma.
Fluxo básico	<p>1. O usuário digita as palavras-chave no campo de Busca;</p> <p>2. O usuário seleciona o tipo de localização para referência:</p> <ol style="list-style-type: none"> 1. Cidade/Estado 2. Um dos endereços cadastrados (caso usuário esteja logado) 3. Localização no mapa <p>3. O usuário clica em buscar;</p> <p>4. O sistema localiza os produtos e produtores que contenham em suas <i>tags</i> as palavras-chave fornecidas pelo usuário;</p> <p>5. O sistema exibe em duas listas separadas uma em cada guia os resultados de produtores e produtos, ordenando-os pela distância caso tenha sido selecionado um endereço cadastrado ou a localização no mapa.</p>
Fluxo alternativo de eventos	<p>Cenário alternativo 2:</p> <p>4. O sistema não identifica nenhum produtor ou produto com as palavras-chave informadas;</p> <p>5. O sistema exibe uma mensagem na tela, informando que nenhum resultado foi encontrado, de acordo com a guia que o usuário está visualizando.</p> <p>Cenário alternativo 3:</p> <p>5. O sistema exibe em duas listas separadas uma em cada guia os resultados de produtores e produtos na cidade selecionada pelo usuário.</p>

Pós-condições	O usuário visualiza duas guias contendo a lista de produtores e produtos.
---------------	---

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 12: Descrição do Caso de uso UC06

Caso de uso	UC06 - Visualizar resultados no mapa
Descrição	Permite ao usuário visualizar os resultados da busca ou seus produtores favoritos em um mapa, com as localizações marcadas.
Pré-condições	O usuário fez uma busca, ou está na tela inicial.
Fluxo básico	<ol style="list-style-type: none"> 1. Na tela inicial, na aba “produtores” o usuário clica em “Ver no mapa”; 2. O aplicativo gera um mapa, centrado na localização atual do usuário, exibindo a localização de seus produtores favoritados.
Fluxo alternativo de eventos	<p>Cenário alternativo 1:</p> <ol style="list-style-type: none"> 2. Não há nenhum item no resultado da busca e, portanto, nenhum outro ponto no mapa se não a localização do usuário. <p>Cenário alternativo 2:</p> <ol style="list-style-type: none"> 2. O sistema não conseguiu identificar a localização atual do usuário; 3. O sistema centraliza o mapa em um ponto intermediário entre os resultados. <p>Cenário alternativo 3:</p> <ol style="list-style-type: none"> 2. Não há nenhum produtor favoritado e portanto, nenhum outro ponto no mapa se não a localização do usuário. <p>Cenário alternativo 4:</p> <ol style="list-style-type: none"> 1. Nos resultados da busca, na aba “produtores” o usuário clica em “Ver no mapa”; 2. O aplicativo gera um mapa, centralizado na localização

	<p>atual do usuário, exibindo a localização dos resultados da busca.</p> <p>Cenário alternativo 5:</p> <ol style="list-style-type: none"> 1. Nos resultados da busca, na aba “produtos” o usuário clica em “Ver no mapa”; 2. O aplicativo gera um mapa, centralizado na localização atual do usuário, exibindo a localização dos resultados da busca.
Pós-condições	O usuário vê pontos de produtos e produtores marcados no mapa com sua localização no centro.

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 13: Descrição do Caso de uso UC07

Caso de uso	UC07 - Manter palavras-chave de interesse
Descrição	Permite ao usuário gerenciar palavras-chave de interesse.
Pré-condições	O usuário está cadastrado.
Fluxo básico	<ol style="list-style-type: none"> 1. O usuário acessa o menu “Meus interesses”; 2. O sistema lista todas as palavras-chave de interesse, exibindo um botão para removê-las; 3. O sistema exibe também uma caixa de texto, onde o usuário pode digitar uma palavra-chave para adicionar a lista de interesses; 4. O usuário digita uma palavra-chave e clica em “adicionar”; 5. O sistema grava a palavra-chave informada pelo usuário; 6. O sistema atualiza a lista de palavras-chave do usuário.
Fluxo alternativo de eventos	Nenhum
Pós-condições	Nenhum

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 14: Descrição do Caso de uso UC08

Caso de uso	UC08 - Avaliar produto
Descrição	Permite ao usuário dar uma avaliação para os produtos.
Pré-condições	O usuário está cadastrado; O usuário está visualizando a tela de detalhes do produto.
Fluxo básico	<ol style="list-style-type: none"> 1. O usuário seleciona a opção “Avaliar produto”; 2. O sistema exibe um formulário com uma escala de 0 a 5 e um campo de observação; 3. O usuário preenche as informações do formulário e clica em “Enviar avaliação”; 4. O sistema registra a avaliação do produto; 5. O sistema atualiza a lista de avaliações do produto e a avaliação média.
Fluxo alternativo de eventos	Nenhum
Pós-condições	A avaliação média do produto foi atualizada.

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 15: Descrição do Caso de uso UC09

Caso de uso	UC09 - Avaliar produtor
Descrição	Permite ao usuário dar uma avaliação para os produtores.
Pré-condições	O usuário está cadastrado; O usuário está visualizando a tela de detalhes do produtor.
Fluxo básico	<ol style="list-style-type: none"> 1. O usuário seleciona a opção “Avaliar produtor”; 2. O sistema exibe um formulário com uma escala de 0 a 5 e um campo de observação; 3. O usuário preenche as informações do formulário e clica

	<p>em “Enviar avaliação”;</p> <p>4. O sistema registra a avaliação do produtor;</p> <p>5. O sistema atualiza a lista de avaliações do produtor e a avaliação média.</p>
Fluxo alternativo de eventos	Nenhum
Pós-condições	A avaliação média do produtor foi atualizada.

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 16: Descrição do Caso de uso UC10

Caso de uso	UC10 - Desfavoritar produtor
Descrição	Permite ao usuário desfavoritar o produtor e com isso fazer com que seus produtos deixem de ser listados na tela inicial.
Pré-condições	O usuário está cadastrado; O usuário está na lista de produtores em sua tela inicial, ou no resultado da busca, ou visualizando os detalhes do produtor.
Fluxo básico	<p>1. O usuário clica no botão de favorito, que está marcado pois o produtor está favoritado;</p> <p>2. O sistema remove o produtor da lista de favoritos do usuário.</p>
Fluxo alternativo de eventos	Nenhum
Pós-condições	Nenhuma

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 17: Descrição do Caso de uso UC11

Caso de uso	UC11 - Favoritar produtor
-------------	----------------------------------

Descrição	Permite ao usuário favoritar o produtor e com isso fazer com que seus produtos sejam listados na tela inicial;
Pré-condições	O usuário está cadastrado; O usuário está na lista de produtores, ou no resultado da busca, ou visualizando os detalhes do produtor.
Fluxo básico	<p>3. O usuário clica no botão de favorito, que está desmarcado pois o produtor ainda não é um favorito;</p> <p>4. O sistema adiciona o produtor à lista de favoritos do usuário.</p>
Fluxo alternativo de eventos	Nenhum
Pós-condições	Nenhuma

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 18: Descrição do Caso de uso UC12

Caso de uso	UC12 - Realizar Login
Descrição	Fornece uma interface que permite ao usuário realizar o login na aplicação.
Pré-condições	O usuário não pode estar logado. O usuário deve possuir cadastro.
Fluxo básico	<p>1. Na tela de login, o sistema exibe um formulário contendo dois campos para o usuário informar, respectivamente, login e senha;</p> <p>2. O usuário preenche os campos no formulário;</p> <p>3. O usuário clica no botão “Entrar”;</p> <p>4. O sistema localiza o usuário, identificando-o através da combinação de e-mail e senha;</p> <p>5. O sistema redireciona o usuário para sua tela inicial.</p>

Fluxos alternativos e exceções	Cenário alternativo 1: 5. O sistema não localiza o usuário; 6. O sistema redireciona o usuário para a tela de login, exibindo a mensagem “E-mail ou senha incorretos”
Pós-condições	O usuário está logado no sistema.

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 19: Descrição do Caso de uso UC13

Caso de uso	UC13 - Manter perfil
Descrição	Permite ao usuário editar seus dados pessoais.
Pré-condições	O usuário está cadastrado.
Fluxo básico	<p>1. O usuário seleciona a opção “Ver perfil”;</p> <p>2. O sistema mostra uma tela com os detalhes do perfil do usuário, de acordo com seu nível: usuário normal ou produtor;</p> <p>3. O usuário clica na opção “Editar perfil”;</p> <p>4. Caso o perfil seja de usuário simples. o sistema abre campos, mostrando os valores anteriores de seu perfil, com opções para editar:</p> <ul style="list-style-type: none"> 1. Foto; 2. Nome; 3. E-mail; 4. Tipo do perfil; <p>5. O usuário altera suas informações e clica em “Salvar”;</p> <p>6. O sistema armazena as novas informações do usuário.</p>
Fluxo alternativo de eventos	<p>4. Caso o perfil seja de produtor. o sistema abre campos, mostrando os valores anteriores de seu perfil, com opções para editar:</p> <ul style="list-style-type: none"> 1. Foto; 2. Nome;

	<p>3. Email;</p> <p>4. Descrição;</p> <p>5. Tipo do perfil;</p> <p>6. Tags associadas;</p> <p>7. Tipos de produtos que produz;</p> <p>5. O usuário altera suas informações e clica em “Salvar”;</p> <p>6. O sistema armazena as novas informações do usuário.</p>
Pós-condições	Nenhum

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 20: Descrição do Caso de uso UC14

Caso de uso	UC14 - Esquecer os dados
Descrição	Permite ao usuário excluir seu perfil e todos os dados a ele relacionados, respeitando leis de privacidade.
Pré-condições	O usuário está cadastrado.
Fluxo básico	<p>1. O usuário acessa a opção “Perfil” do menu;</p> <p>2. O usuário seleciona a opção “Excluir conta”;</p> <p>3. O sistema abre uma tela com aviso de que a operação não pode ser desfeita e um campo para o usuário digitar sua senha;</p> <p>4. O usuário digita sua senha;</p> <p>5. O sistema valida a senha fornecida pelo usuário;</p> <p>6. O sistema apaga todos os dados relacionados ao usuário.</p>
Fluxo alternativo de eventos	<p>Cenário alternativo 1:</p> <p>4. O usuário cancela a operação;</p> <p>5. O sistema cancela a operação e fecha a janela.</p> <p>Cenário alternativo 2:</p> <p>6. A senha fornecida pelo usuário é inválida;</p>

	7. O sistema exibe uma mensagem informando que a senha é inválida e a solicita novamente até que seja validada ou cancelada.
Pós-condições	O perfil do usuário e todas as informações a ele associadas não existem mais.

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 21: Descrição do Caso de uso UC15

Caso de uso	UC15 - Manter produtos
Descrição	Permite ao produtor cadastrar, editar e excluir seus itens de produção.
Pré-condições	O perfil de produtor está habilitado.
Fluxo básico	<p>1. O produtor seleciona a opção “Gerenciar Produtos”;</p> <p>2. O sistema lista os produtos que o produtor já cadastrou, cada um com um botão para:</p> <ol style="list-style-type: none"> 1. Enviar imagens 2. Editar 3. Excluir <p>3. O sistema exibe também uma opção para o produtor cadastrar;</p> <p>4. O produtor seleciona a opção “Cadastrar”;</p> <p>5. O sistema exibe um formulário para cadastro do produto, contendo os campos:</p> <ol style="list-style-type: none"> 1. Nome 2. Descrição 3. Preço (campo opcional) 4. Categoria 5. Foto principal <p>6. O produtor preenche os campos do formulário e clica em “Salvar”;</p> <p>7. O sistema armazena os dados do produto.</p>
Fluxo alternativo de eventos	Cenário alternativo 1: <p>4. O produtor clica na opção de “Editar” em um dos produtos da lista;</p>

	<p>5. O sistema exibe um formulário para cadastro do produto, contendo os campos já preenchidos com as informações atuais:</p> <ol style="list-style-type: none"> 1. Nome 2. Descrição 3. Preço (campo opcional) 4. Categoria 5. Foto principal <p>6. O produtor altera os dados do formulário conforme necessário;</p> <p>7. O produtor clica em “Salvar”;</p> <p>8. O sistema atualiza os dados do produto.</p> <p>Cenário alternativo 2:</p> <ol style="list-style-type: none"> 4. O produtor clica na opção “Excluir” em um dos produtos da lista; 5. O sistema exibe uma caixa para confirmação da operação; 6. O produtor confirma a operação; 7. O sistema apaga os dados do produto e todas as demais informações a ele relacionadas.
Pós-condições	Nenhuma

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 22: Descrição do Caso de uso UC16

Caso de uso	UC16 - Login com redes sociais
Descrição	Fornece uma interface que permite ao usuário realizar o login na aplicação utilizando sua conta em redes sociais.
Pré-condições	O usuário está desconectado.
Fluxo básico	<ol style="list-style-type: none"> 1. Na tela de login, o sistema exibe o formulário de login normal, e outros dois botões para login com o Facebook e conta Google; 2. O usuário seleciona a opção de login usando Facebook; 3. O sistema coleta os dados fornecidos pela API do Facebook;

	<p>4. O sistema valida se existe cadastro relacionado aquele perfil;</p> <p>5. O sistema realiza o login e redireciona o usuário para a tela inicial.</p>
Fluxo alternativo de eventos	<p>Cenário alternativo 1:</p> <p>6. O sistema não localiza o usuário;</p> <p>7. O sistema redireciona o usuário para a tela de login, exibindo a mensagem “Usuário não localizado”.</p> <p>Cenário alternativo 2:</p> <p>2. O usuário seleciona a opção de login usando Google;</p> <p>3. O sistema coleta os dados fornecidos pela API do Google;</p> <p>4. O sistema valida se existe cadastro relacionado aquele perfil;</p> <p>5. O sistema realiza o login e redireciona o usuário para a tela inicial.</p>
Pós-condições	O usuário está logado no sistema.

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 23: Descrição do Caso de uso UC17

Caso de uso	UC17 - Habilitar perfil de produtor
Descrição	Fornece uma interface que permite ao usuário habilitar seu perfil de produtor, permitindo que seja localizado em busca.
Pré-condições	O usuário está conectado.
Fluxo básico	<p>1. Na tela de manutenção do perfil, o sistema exibe um botão para ativar/desativar o perfil de produtor;</p> <p>2. O usuário ainda não está com o perfil de produtor ativado;</p> <p>3. O usuário clica no campo, ativando o perfil de produtor;</p> <p>4. O sistema habilita o campo de descrição, <i>tags</i> e categorias.</p>
Fluxo	<p>Cenário alternativo 1:</p> <p>2. O usuário já está com o perfil de produtor ativado;</p>

alternativo de eventos	3. O usuário clica no campo, desativando o perfil de produtor; 4. O produtor confirma a ação; 5. O sistema desativa o campo de descrição, tags e categorias de produtos, e as opções de anexo de imagens ao perfil e gestão de produtos no menu.
Pós-condições	Nenhuma

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 24: Descrição do Caso de uso UC18

Caso de uso	UC18 - Visualizar página inicial
Descrição	Fornece a tela inicial para qualquer tipo de usuário, cadastrado ou não.
Pré-condições	Nenhuma
Fluxo básico	1. O usuário está conectado; 2. O usuário possui ao menos um endereço cadastrado; 3. O sistema exibe uma aba com seus produtores favoritos e outra aba com resultados compatíveis com seus interesses.
Fluxo alternativo de eventos	1. O usuário está desconectado; 2. O sistema exibe uma mensagem orientando-o a realizar o cadastro para desfrutar de mais funcionalidades da aplicação, junto de um botão para a tela de cadastro.
Pós-condições	Nenhuma

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 25: Descrição do Caso de uso UC19

Caso de uso	UC19 - Visualizar produtores favoritos
Descrição	Fornece ao usuário a lista de dos produtores que favoritou.

Pré-condições	O usuário deve estar cadastrado.
Fluxo básico	<ol style="list-style-type: none"> 1. O usuário está conectado; 2. O sistema exibe uma lista paginada com a lista de seus produtores favoritos ordenados alfabeticamente; 3. O sistema exibe também um botão para “Ver no mapa”.
Fluxo alternativo de eventos	<ol style="list-style-type: none"> 2. O usuário ainda não favoritou nenhum produtor; 3. O sistema exibe uma mensagem indicando que ele deve favoritar produtores para visualizar em sua tela inicial.
Pós-condições	Nenhuma
Exceções ao fluxo normal de eventos	<ul style="list-style-type: none"> • Caso o usuário não tenha favoritado produtores, o sistema exibe uma mensagem indicando que ele deve favoritar produtores para visualizar em sua tela inicial; • Caso o usuário não tenha cadastrado palavras-chave, o sistema exibe uma mensagem indicando que ele deve adicionar palavras-chave para visualizar resultados compatíveis em sua tela inicial.

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 26: Descrição do Caso de uso UC20

Caso de uso	UC20 - Visualizar produtos e produtores compatíveis com interesses
Descrição	Fornece ao usuário a lista de produtos e produtores compatíveis com as palavras-chave de seu interesse.
Pré-condições	O usuário deve estar cadastrado.
Fluxo básico	<ol style="list-style-type: none"> 1. O usuário está conectado; 2. O sistema exibe um botão para alternar a visualização entre produtos e produtores; 3. Na aba de produtores, o sistema lista os produtores que

	<p>possuem alguma das <i>tags</i> marcadas como interesse pelo usuário;</p> <ol style="list-style-type: none"> 4. O usuário possui ao menos um endereço cadastrado; 5. O sistema lista 6 produtos e 6 produtores que possuem alguma das <i>tags</i> marcadas como interesse pelo usuário na cidade de seu endereço principal.
Fluxo alternativo de eventos	<p>Cenário alternativo 1:</p> <ol style="list-style-type: none"> 2. O usuário ainda não cadastrou nenhuma palavra-chave; 3. O sistema exibe uma mensagem indicando que ele deve adicionar palavras-chave para visualizar resultados compatíveis em sua tela inicial. <p>Cenário alternativo 2:</p> <ol style="list-style-type: none"> 4. O usuário não cadastrou nenhum endereço; 5. Na aba de produtos, o sistema lista 6 produtos e 6 produtores que possuem alguma das <i>tags</i> marcadas como interesse pelo usuário aleatoriamente;
Pós-condições	Nenhuma

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 27: Descrição do Caso de uso UC21

Caso de uso	UC21 - Manter endereços
Descrição	Fornece ao usuário cadastrado uma interface para gerenciar seus endereços. Para um produtor, é necessário cadastrar ao menos um endereço.
Pré-condições	O usuário deve estar cadastrado e logado.
Fluxo básico	<ol style="list-style-type: none"> 1. O usuário seleciona a opção “Gerenciar Endereços” 2. O sistema lista os endereços que o usuário já cadastrou, cada um com um botão para: <ol style="list-style-type: none"> 1. Editar 2. Excluir 3. Editar contatos 4. Definir como endereço principal 3. O sistema exibe também uma opção para o usuário

	<p>cadastrar novo endereço;</p> <ol style="list-style-type: none"> 4. O usuário seleciona a opção “Novo endereço”; 5. O sistema exibe um formulário para cadastro do endereço, contendo os campos: <ol style="list-style-type: none"> 1. Nome 2. Rua 3. Número 4. Bairro 5. Cidade 6. Estado 7. Mapa para selecionar a localização 6. O usuário preenche os campos do formulário e clica em “Salvar”; 7. O sistema armazena o novo endereço e redireciona o usuário para a tela de cadastro de canal de contato.
Fluxo alternativo de eventos	<p>Cenário alternativo 1:</p> <ol style="list-style-type: none"> 4. O usuário clica na opção de “Editar” em um dos endereços da lista; 5. O sistema exibe um formulário para alteração do endereço, contendo os mesmos campos do formulário de cadastro, preenchido com os dados atuais; 6. O usuário altera os dados do formulário conforme necessário; 7. O usuário clica em “Salvar”; 8. O sistema armazena os novos dados de endereço. <p>Cenário alternativo 2:</p> <ol style="list-style-type: none"> 4. O usuário clica na opção “Excluir” em um dos endereços da lista; 5. O sistema exibe uma caixa para confirmação da operação; 6. O usuário confirma a operação; 7. O sistema apaga os dados do endereço e todas as demais informações a ele relacionadas. <p>Cenário alternativo 3:</p> <ol style="list-style-type: none"> 4. O usuário clica na opção “Definir como principal” em um dos endereços da lista; 5. O sistema atualiza o endereço, definindo como principal.

Pós-condições	Nenhuma
---------------	---------

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 28: Descrição do Caso de uso UC22

Caso de uso	UC22 - Manter canais de contato
Descrição	Fornece ao usuário cadastrado uma interface para gerenciar seus meios de contato, associados aos endereços. Para um produtor, é necessário cadastrar ao menos um canal de contato.
Pré-condições	O usuário deve estar cadastrado e logado. O usuário deve ter clicado em “Editar endereços” na lista de endereço, ou é redirecionado após o cadastro de um endereço.
Fluxo básico	<ol style="list-style-type: none"> 1. O sistema exibe uma lista dos meios de contato associados ao endereço, com um botão para edição e exclusão em cada um. Além de um botão para cadastrar novo contato; 2. O usuário seleciona a opção “Novo contato”; 3. O sistema exibe um formulário para cadastro do contato contendo os campos: Tipo do contato e valor; 4. O usuário preenche os campos do formulário e clica em “Salvar”; 5. O sistema armazena o novo contato e redireciona o usuário novamente para a lista de contatos cadastrados.
Fluxo alternativo de eventos	<p>Cenário alternativo 1:</p> <ol style="list-style-type: none"> 2. O usuário clica na opção de “Editar” em um dos contatos da lista; 3. O sistema exibe um formulário para alteração do contato, contendo os mesmos campos do formulário de cadastro, preenchido com os dados atuais; 4. O usuário altera os dados do formulário conforme necessário; 5. O usuário clica em “Salvar”; 6. O sistema atualiza os dados do contato e redireciona o usuário para a lista de contatos.

	<p>Cenário alternativo 2:</p> <ol style="list-style-type: none"> 2. O usuário clica na opção “Excluir” em um dos contatos da lista; 3. O sistema exibe uma caixa para confirmação da operação; 4. O usuário confirma a operação; 5. O sistema apaga os dados do contato.
Pós-condições	Nenhuma

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 29: Descrição do Caso de uso UC23

Caso de uso	UC23 - Anexar imagens ao perfil
Descrição	Fornece ao usuário produtor uma forma de anexar imagens ao seu perfil, de modo a dar mais informações sobre sua produção.
Pré-condições	O usuário produtor deve estar cadastrado e logado.
Fluxo básico	<ol style="list-style-type: none"> 1. O produtor seleciona a opção “imagens do perfil”; 2. O sistema exibe uma tela contendo as imagens já anexadas, e em cada uma delas uma opção para editar a descrição ou excluir o registro; 3. O sistema exibe também um campo onde o produtor pode selecionar uma ou mais imagens e enviá-las; 4. O produtor clica no campo de <i>upload</i> e seleciona as imagens que deseja enviar; 5. O produtor clica no botão “Enviar”; 6. O sistema envia as imagens e atualiza a lista de imagens cadastradas.
Fluxo alternativo de eventos	<p>Cenário alternativo 1:</p> <ol style="list-style-type: none"> 4. O produtor clica na opção para editar a descrição da imagem; 5. O sistema exibe a descrição da imagem em um campo de texto, onde o produtor pode editar o valor;

	<p>6. O produtor clica no botão “salvar”; 7. O sistema atualiza a descrição da imagem.</p> <p>Cenário alternativo 2:</p> <p>4. O usuário clica na opção “Excluir” em uma imagem; 5. O sistema exibe uma caixa para confirmação da operação; 6. O usuário confirma a operação; 7. O sistema apaga a imagem.</p>
Pós-condições	Nenhuma

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 30: Descrição do Caso de uso UC24

Caso de uso	UC24 - Anexar imagens ao produto
Descrição	Fornece ao usuário produtor uma forma de anexar imagens ao produto, de modo a dar mais informações para os interessados.
Pré-condições	O usuário produtor deve estar cadastrado e logado. O produto deve estar previamente cadastrado.
Fluxo básico	<p>1. Na tela de edição do produto, o produtor seleciona a opção “Imagens”;</p> <p>2. O sistema exibe uma tela contendo as imagens já anexadas, e em cada uma delas uma opção para editar a descrição ou excluir o registro;</p> <p>3. O sistema exibe também um campo onde o produtor pode selecionar uma ou mais imagens e enviá-las;</p> <p>4. O produtor clica no campo de <i>upload</i> e seleciona as imagens que deseja enviar;</p> <p>5. O produtor clica no botão “Enviar”;</p> <p>6. O sistema envia as imagens e atualiza a lista de imagens cadastradas.</p>
	<p>Cenário alternativo 1:</p> <p>4. O produtor clica na opção para editar a descrição da imagem;</p>

Fluxo alternativo de eventos	<p>5. O sistema exibe a descrição da imagem em um campo de texto, onde o produtor pode editar o valor;</p> <p>6. O produtor clica no botão “salvar”;</p> <p>7. O sistema atualiza a descrição da imagem.</p> <p>Cenário alternativo 2:</p> <p>4. O usuário clica na opção “Excluir” em uma imagem;</p> <p>5. O sistema exibe uma caixa para confirmação da operação;</p> <p>6. O usuário confirma a operação;</p> <p>7. O sistema apaga a imagem.</p>
Pós-condições	Nenhuma

Fonte: (DADOS DA PESQUISA, 2020)

Quadro 31: Descrição do Caso de uso UC25

Caso de uso	UC25 – Dar sugestão ou feedback
Descrição	Fornece ao usuário, conectado ou não, uma interface para enviar sugestões à equipe de desenvolvimento do protótipo.
Pré-condições	Nenhuma.
Fluxo básico	<ol style="list-style-type: none"> 1. O usuário acessa o menu “Feedback”; 2. O sistema exibe um formulário contendo nome, e-mail e um campo para o usuário digitar a sugestão; 3. Caso o usuário já esteja conectado, o nome e e-mail são preenchidos com as informações do usuário logado; 4. O usuário preenche o formulário; 5. O sistema envia a sugestão para a equipe de desenvolvimento.
Fluxo alternativo de eventos	
Pós-	Nenhuma

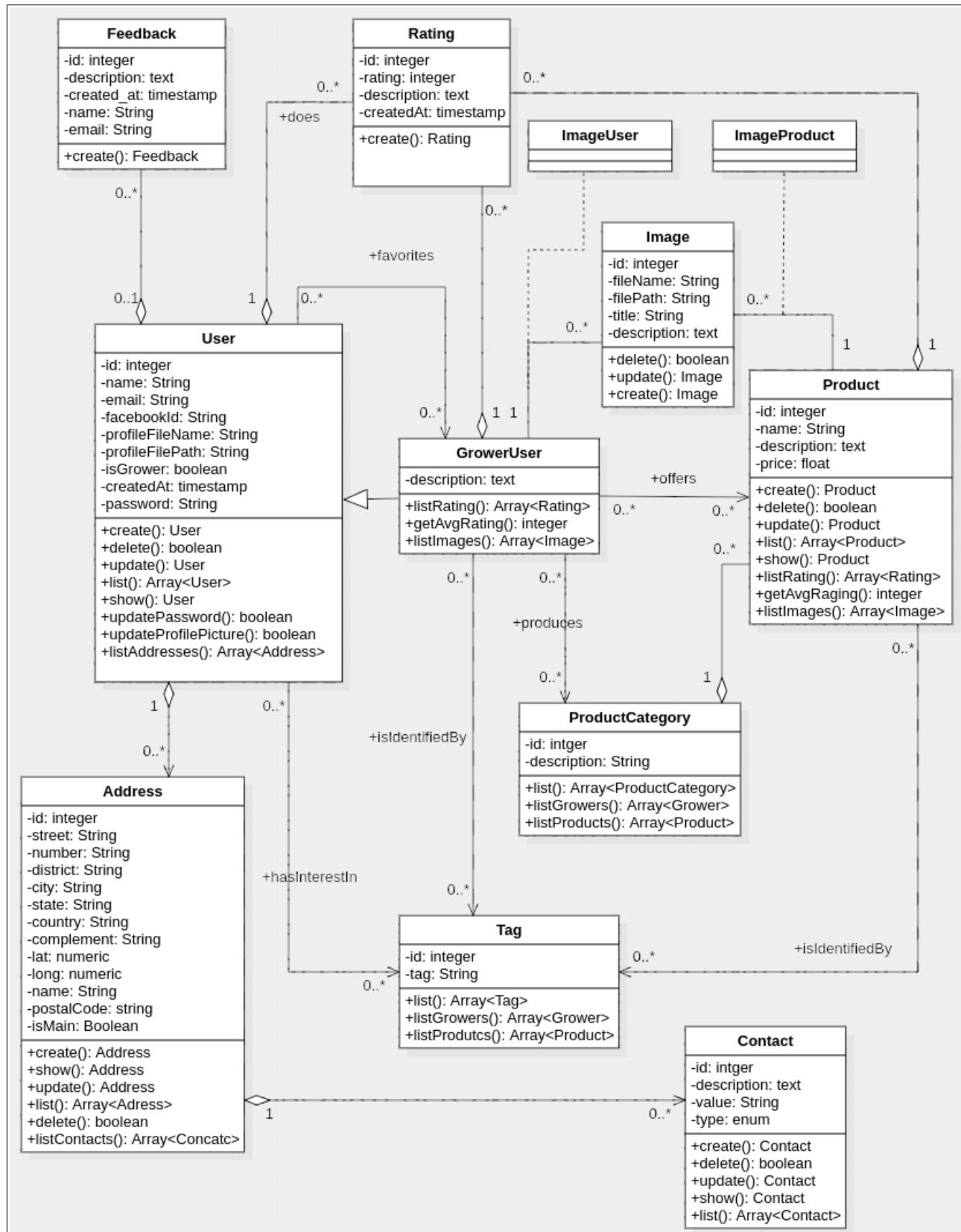
condições	
-----------	--

Fonte: (DADOS DA PESQUISA, 2020)

4.6.2 Diagrama de Classes

Após o desenvolvimento do diagrama de caso de uso, foi desenvolvido o diagrama de classe, a fim de compreender quais classes irão compor o sistema e como irão se relacionar, bem como os atributos que cada classe deverá possuir. O diagrama de classe também fornece uma visão das entidades, seus atributos e relacionamentos quando em um banco de dados, desempenhando papel semelhante ao modelo ER.

Figura 43: Diagrama de classes



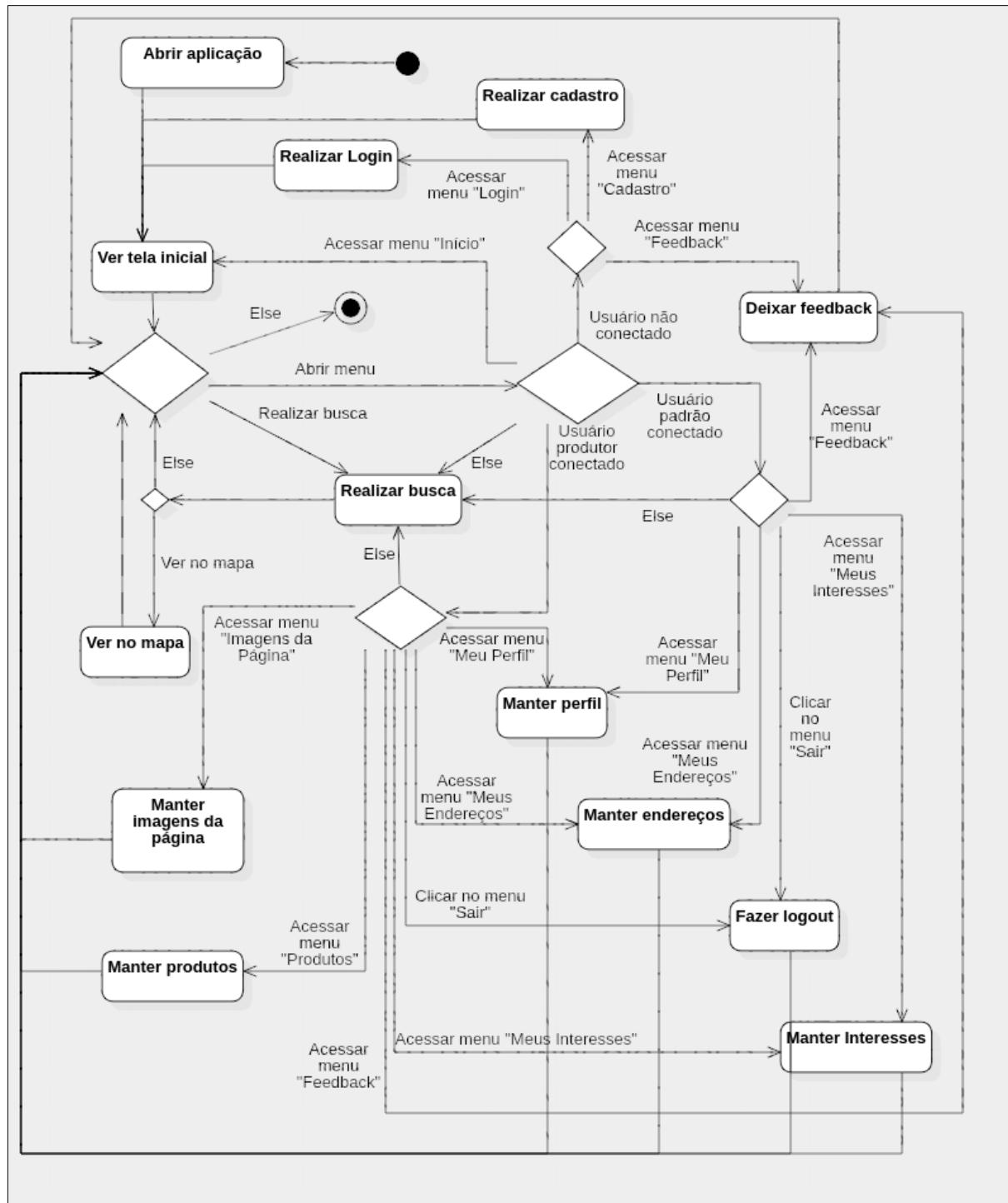
Fonte: (DADOS DA PESQUISA, 2020)

Optou-se por nomear as classes em inglês, seguindo a convenção de nomenclatura definida pelo Eloquent ORM, integrado à *framework* utilizada na construção da API.

4.6.3 Diagrama de Atividade

O diagrama de atividade, muito semelhante a um fluxograma, é o diagrama UML utilizado para demonstrar fluxos de ações em um sistema. Nesse projeto, o diagrama foi utilizado para modelar e demonstrar o fluxo de atividades da aplicação cliente. Com base no diagrama desenvolvido foi definida a interface e funcionamento da aplicação.

Figura 44: Diagrama de atividade do fluxo completo da aplicação



Fonte: (DADOS DA PESQUISA, 2020)

Pode-se observar que a qualquer momento é possível realizar uma busca. Também observa-se que o usuário produtor compartilha todas as ações de um usuário padrão, tendo algumas permissões a mais. Além disso, as únicas ações disponíveis ao usuário não cadastrado é realizar busca, visualizar tela inicial, realizar login, realizar

cadastro e enviar *feedback*. Percebe-se a ordem cílica das ações uma vez que o fim do processo só ocorre quando o usuário opta por não realizar nenhuma ação.

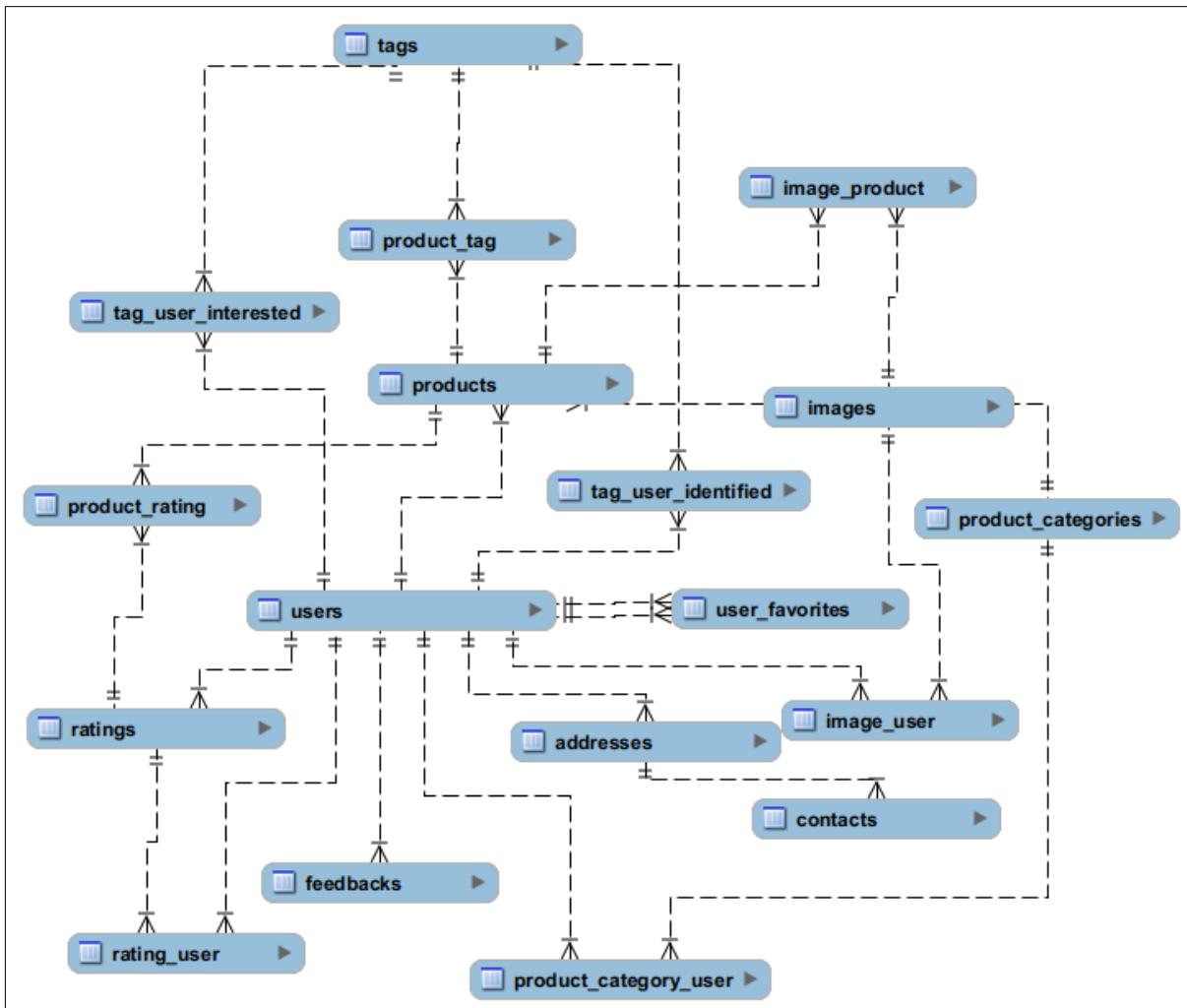
4.7 CONSTRUÇÃO

Após finalizada a modelagem do protótipo, deu-se início à implementação. Esta, por sua vez, ocorreu em iterações, tendo início na configuração do projeto, construção da base de dados e as seguintes dedicaram-se a entregar um componente funcional que implemente ao menos um caso de uso. Os resultados são demonstrados a seguir.

4.7.1 Banco de Dados

Após a configuração de cada componente da arquitetura geral, iniciou-se a construção do banco de dados, pois este será a base de toda a aplicação. Com base no diagrama de classes, foi possível identificar as tabelas e atributos do banco de dados. Com isso, a construção do banco de dados ocorreu através da funcionalidade *migrations*, provida pela *framework* Lumen. Com o auxílio da ferramenta de engenharia reversa do software MySQL Workbench, foi gerado o diagrama ER da base de dados obtida.

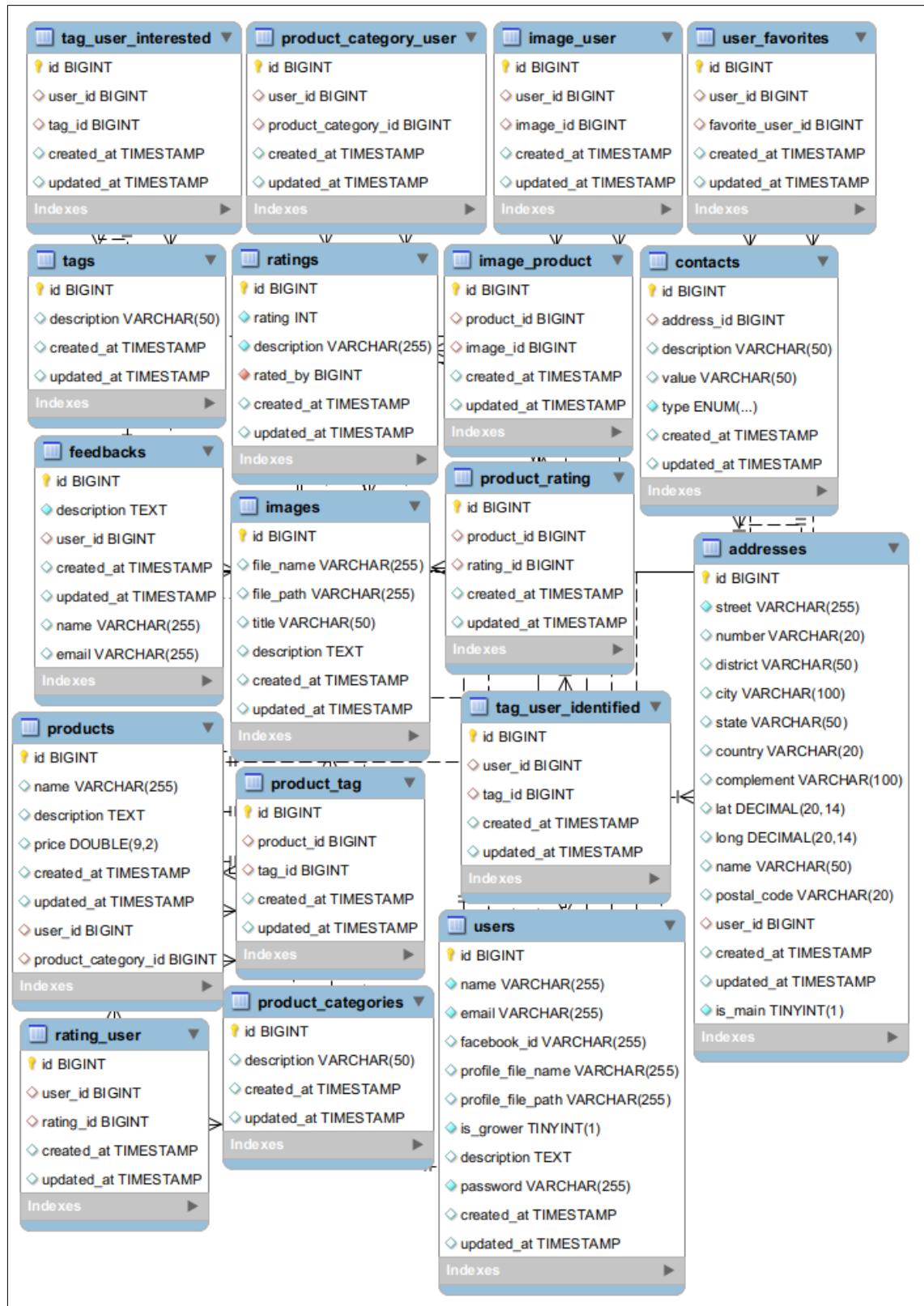
Figura 45: Visão dos relacionamentos entre entidades do banco de dados



Fonte: (DADOS DA PESQUISA, 2020)

Nessa visão, foram suprimidos os atributos de cada entidade a fim de possibilitar a demonstração de todas as entidades em uma única figura de forma legível. Na Figura 46 são demonstradas as entidades com seus atributos, dificultando a leitura das relações.

Figura 46: Visão expandida das entidades apresentadas do modelo ER



Fonte: (DADOS DA PESQUISA, 2020)

4.7.2 Codificação

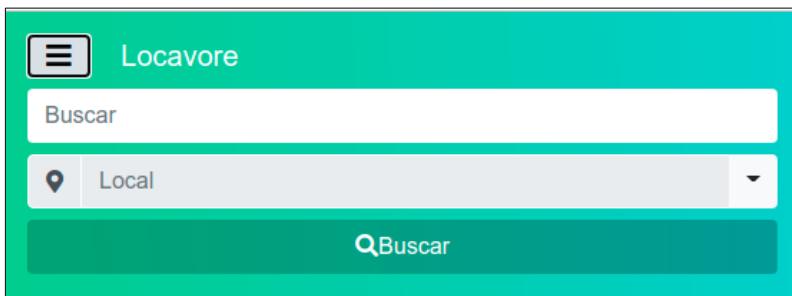
Devido ao modelo orientado a cronograma escolhido para o desenvolvimento e a característica de protótipo, entendeu-se que os casos de uso deveriam ser priorizados para o desenvolvimento, a fim de ao esgotar-se o tempo hábil para codificação do projeto, tenha-se ao menos um protótipo que demonstre os objetivos principais do trabalho. Para isso, foi necessário compreender a totalidade de casos de uso e dependência entre eles. Desse modo, os casos de uso foram priorizados ordenados para execução.

Cada caso de uso exigiu construção da funcionalidade de interação na camada de cliente e da funcionalidade na camada de aplicação, para persistência ou recuperação dos resultados no banco de dados, desse modo a atuação ocorreu em ambas as camadas de forma alternada, sendo testadas de forma informal durante o desenvolvimento, com base na descrição do caso de uso a ser atendido. As interfaces que atendem aos casos de uso acima mencionados são listadas a seguir. Tendo em vista a intenção de implementar a aplicação com uma interface responsiva, as Figuras representam a interface em sua visão para celular.

4.7.2.1 Menu

O menu fornece ao usuário navegação dentro da aplicação e está presente em todas as páginas, inclusive no Cadastro e Login.

Figura 47: Menu compactado

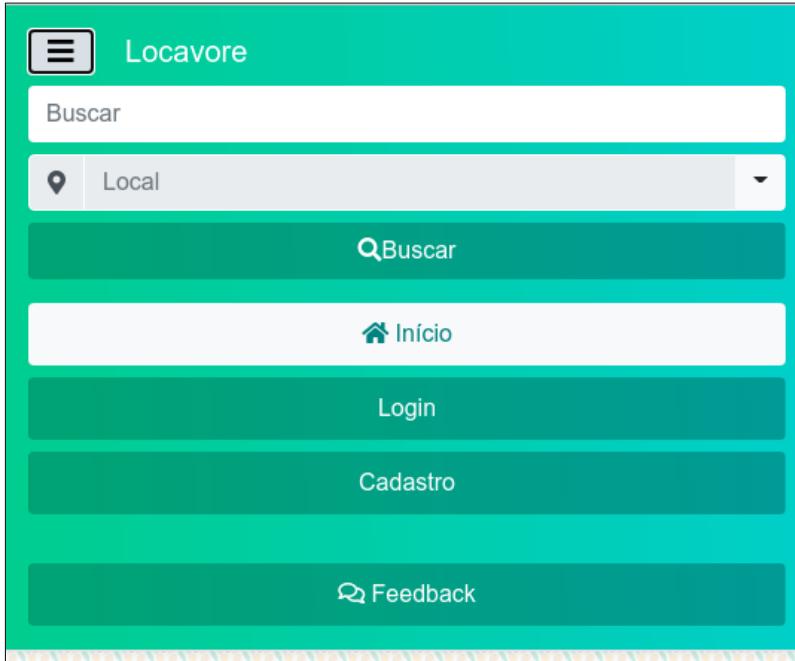


Fonte: (DADOS DA PESQUISA, 2020)

Em qualquer visão do menu, há o formulário de busca que possibilita ao usuário realizar busca a partir de qualquer página, e ao menu “Feedback” que fornece acesso à tela onde o usuário pode enviar um *Feedback* para a equipe. Ao clicar no ícone de barra no canto superior esquerdo, o menu expande as opções, exibindo acesso às interfaces do

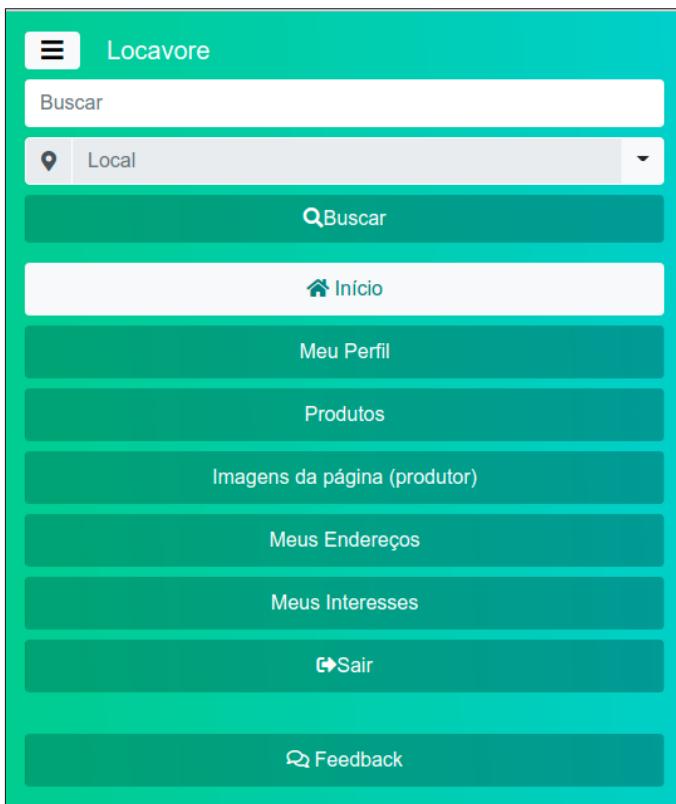
protótipo de acordo com o nível de acesso: usuário desconectado, usuário produtor e usuário padrão.

Figura 48: Visão expandida do menu para usuário desconectado



Fonte: (DADOS DA PESQUISA, 2020)

Figura 49: Visão expandida do menu para usuário produtor



Fonte: (DADOS DA PESQUISA, 2020)

Com exceção dos menus “Imagens da página (produtor)” e “Produtos”, os demais menus estão visíveis na visão expandida para o usuário padrão.

4.7.2.2 Cadastro

Fornece ao usuário uma interface para realizar seu cadastro, atendendo ao caso de uso UC03 – Realizar cadastro. Ao clicar no link “já possuo cadastro” o usuário é redirecionado para a tela de Login.

Figura 50: Interface de Cadastro de usuário

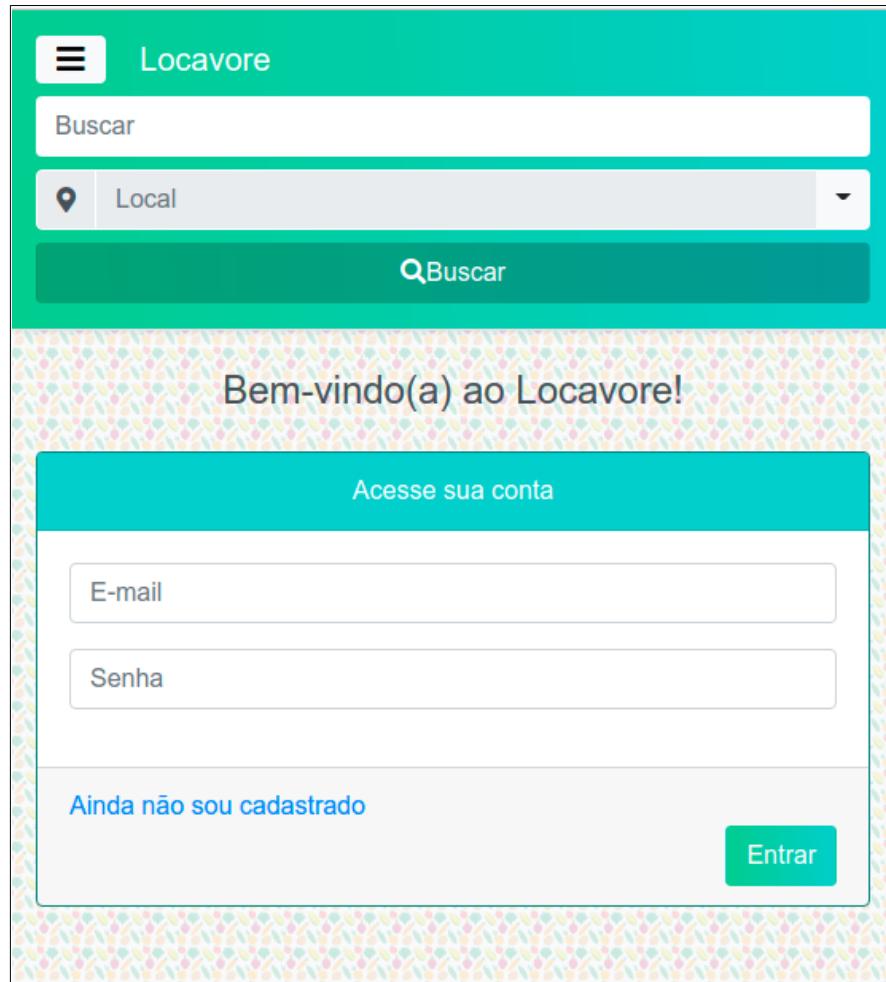
A imagem mostra a interface de usuário para o cadastro de um novo usuário. No topo, há uma barra azul com o logo "Locavore" e uma barra de busca com o placeholder "Buscar". Abaixo disso, uma barra amarela contém uma caixa de seleção com o placeholder "Local" e um botão "Buscar". O formulário principal, intitulado "Bem-vindo(a) ao Locavore!", exibe campos para "Nome", "E-mail", "Senha" e "Confirmar senha". À direita do formulário, há um link "Já possuo cadastro" e um botão verde "Cadastrar".

Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.3 Login

Fornece ao usuário não conectado interface para logar no sistema. Clicando no link “Ainda não sou cadastrado” o usuário é redirecionado para a interface de Cadastro. Essa interface atende ao caso de uso UC12 – Realizar Login

Figura 51: Interface de Login

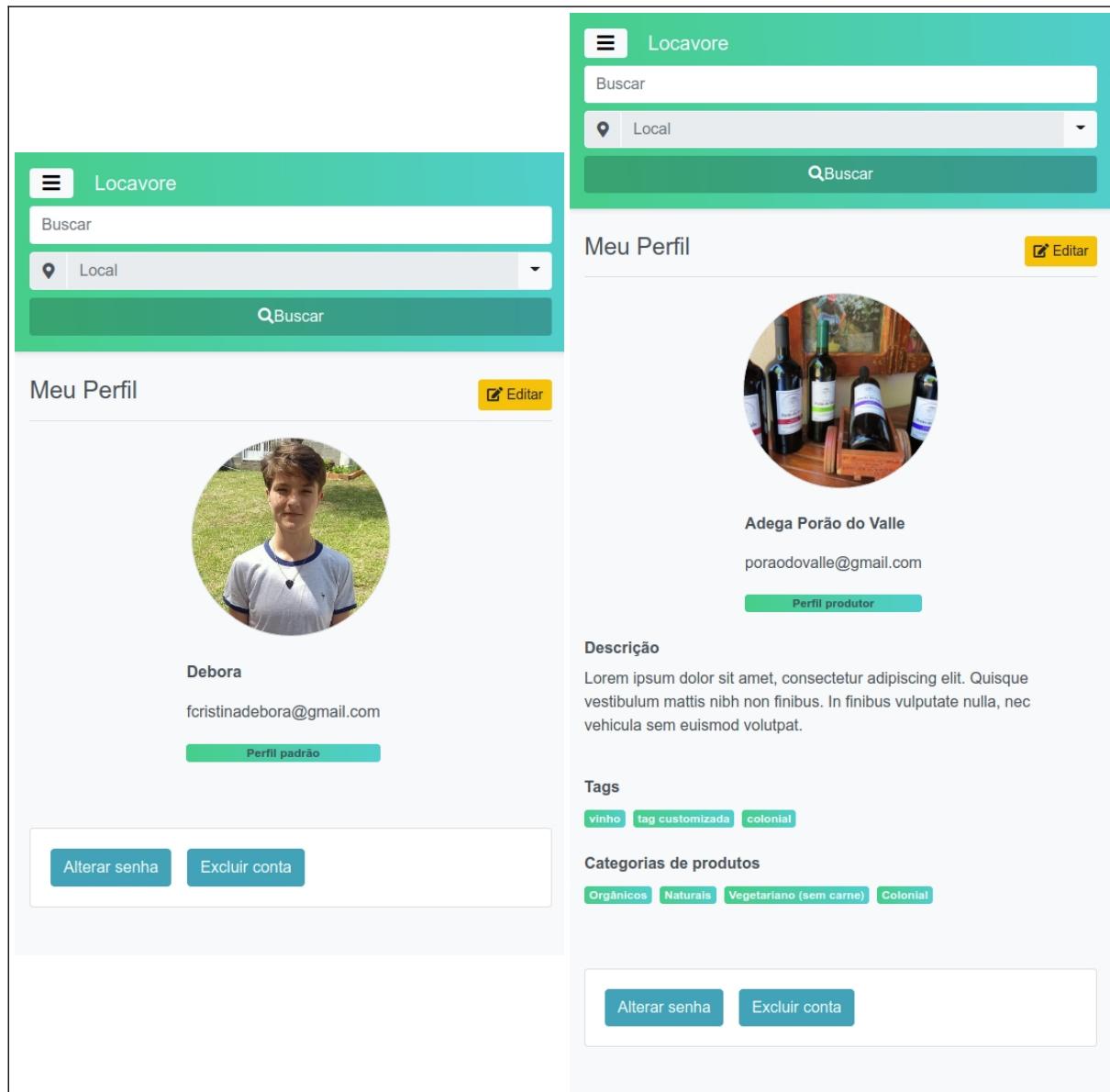


Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.4 Meu Perfil

Atendendo ao caso de uso UC13 – Manter perfil. Ao acessar o menu “Meu perfil” o usuário visualiza uma página com seus dados de perfil. Há duas visões distintas para usuários padrão e produtores.

Figura 52: À esquerda, perfil do usuário padrão, à direita, perfil do usuário produtor.



Fonte: (DADOS DA PESQUISA, 2020)

Clicando no botão “editar” habilita-se um formulário que permite ao usuário alterar suas informações. Na visão do usuário padrão, são ocultados os campos “Descrição”, “tags” e “categorias de produtos”, porém, caso seja alterado o valor do campo “perfil” para “Produtor” o sistema habilita os campos exclusivos para produtor. Essa função atende ao caso de uso UC17 – Habilitar ou desabilitar perfil de produtor.

Figura 53: Formulário para alteração dos dados do perfil, visão do produtor.

The screenshot shows a user interface for profile editing. At the top, there's a navigation bar with a search bar and a location dropdown. Below it, the title "Meu Perfil" is displayed, along with "Cancelar" and "Salvar" buttons. A circular profile picture placeholder is shown, with a file upload section below it. The main form area contains fields for "Nome" (Adega Porão do Vale), "E-mail" (poraovalle@gmail.com), and "Perfil" (Produtor). There's also a rich text editor for "Descrição" containing placeholder text. The "Tags" section lists "vinho", "tag customizada", and "colonial". Under "Categorias de produtos", there's a list of checked and unchecked options: Orgânicos, Naturais, Artesanais, Veganos (sem origem animal), Vegetariano (sem carne), and Colonial. At the bottom, there are "Alterar senha" and "Excluir conta" buttons.

Fonte: (DADOS DA PESQUISA, 2020)

Ao clicar no botão “Alterar senha” abre-se um *frame* com os campos para alteração da senha do usuário. Ao preencher todas as informações de forma válida, o sistema altera a senha do usuário, complementando o caso de uso UC13 – Manter perfil.

Figura 54: Frame para alteração de senha

O frame contém o seguinte formulário:

- Alterar senha**
- Senha atual**: campo de texto.
- Nova senha**: campo de texto.
- Confirmar senha**: campo de texto.
- Cancelar** e **✓ Salvar** (botão verde com ícone de checkmark).

Fonte: (DADOS DA PESQUISA, 2020)

Ao clicar no botão “Excluir conta” abre-se um *frame* onde é exibido um aviso e solicitada a senha do usuário para autorizar a exclusão, atendendo ao caso de uso UC14 – Esquecer os dados.

Figura 55: Frame para exclusão da conta

O frame contém o seguinte formulário:

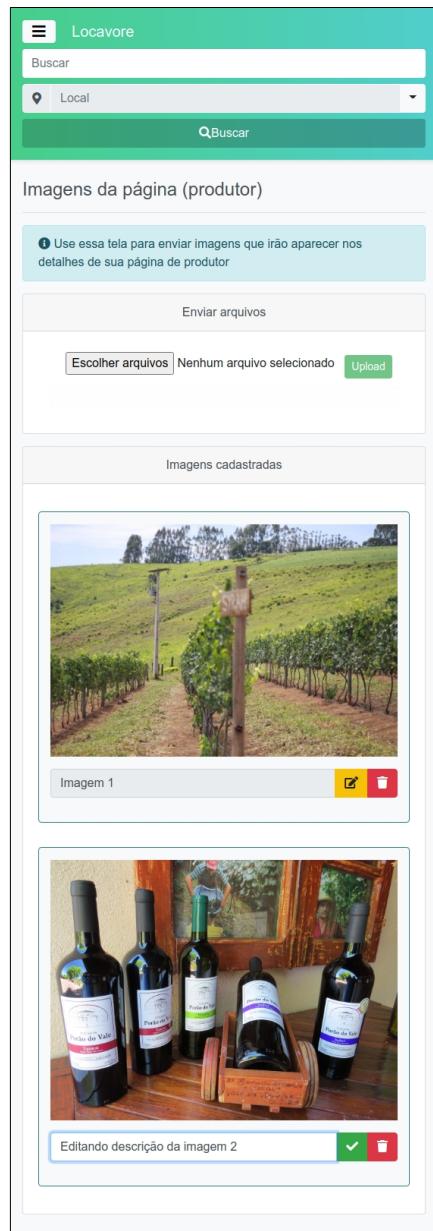
- Excluir conta**
- Atenção!** Ao excluir sua conta, todos seus dados e demais informações associadas ao seu perfil serão excluídos. **Essa operação não poderá ser desfeita**
- Para prosseguir com a operação, informe sua senha: campo de texto.
- Cancelar** e **✓ Excluir conta** (botão vermelho com ícone de checkmark).

Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.5 Imagens da página (produtor)

Através dessa interface, o produtor realiza o caso de uso UC23 – Anexar imagens ao perfil. É possível enviar mais de uma imagem na mesma ação, posteriormente pode-se editar a descrição ou excluir o registro. Na Figura 56, a descrição da segunda imagem está sendo editada.

Figura 56: Interface para anexo de imagens ao perfil



Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.6 Meus produtos

Permite realizar o caso de uso UC15 – Manter produtos; Ao acessar o menu “Meus produtos” são listados os produtos já cadastrados pelo produtor. Ao clicar no botão “Novo” o usuário é direcionado ao formulário de cadastro do produto. Clicando no ícone de “Editar” no produto listado, o mesmo formulário é carregado, porém com os dados já preenchidos, permitindo a edição das informações.

Figura 57: Lista de produtos cadastrados

The screenshot shows a mobile application interface for managing products. At the top, there is a header with a menu icon and the text "Locavore". Below the header is a search bar labeled "Buscar". Underneath the search bar is a dropdown menu for "Local". A large green button labeled "Buscar" with a magnifying glass icon is positioned below the dropdown. The main content area is titled "Meus produtos" and features a "Novo" button. Two product entries are displayed:

- [cod. 6] Vinho colonial suave 1L
Valor: R\$ 22,00
Detalhes:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
- [cod. 5] Vinho colonial seco 1L
Valor: R\$ 20,00
Detalhes:
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut

Each product entry has three small circular icons on the right side: a red one with a white minus sign, a yellow one with a white checkmark, and a blue one with a white plus sign.

Fonte: (DADOS DA PESQUISA, 2020)

Figura 58: Formulário de edição de produto

The screenshot shows a user interface for editing a product. At the top, there's a navigation bar with a menu icon, the brand name 'Locavore', a search bar labeled 'Buscar', a dropdown menu set to 'Local', and a search button with a magnifying glass icon labeled 'Buscar'. Below this, the main content area has a header 'Meus produtos > Editar' with 'Cancelar' and 'Salvar' buttons. The form fields include:

- Nome ***: Vinho colonial seco 1L
- Categoria ***: Colonial
- Valor (R\$)**: 20
- Detalhes**: A rich text editor containing placeholder text about labor rights.
- Tags**: vinho, colonial, tag

Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.7 Anexar imagens ao produto

Clicando no ícone de imagens em um dos produtos listados na lista de produtos, o usuário é direcionado para uma interface semelhante à interface para anexo de imagens ao perfil do produtor, sendo essa a interface para anexo de imagens ao produto.

Figura 59: Interface para anexo de imagens ao produto



Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.8 Meus endereços

Possibilita a realização do caso de uso UC21 – Manter endereços. Ao acessar o menu “Meus endereços” o usuário é direcionado para a lista de seus endereços cadastrados, onde um deles estará marcado como principal. Clicando no botão “Novo” o usuário acessa o formulário de cadastro do endereço onde, ao preencher o CEP, o sistema realiza uma requisição a uma API externa que retorna a rua, bairro, cidade e

estado relacionados ao CEP informado. Clicando no botão de edição o usuário é direcionado ao mesmo formulário, com os dados do endereço selecionado para edição. Observa-se no formulário que há um mapa, onde o usuário deve localizar sua posição e clicar no mapa para selecionar, essa interface permite ao sistema identificar a latitude e longitude do endereço, para ser usado como base nos cálculos de distância.

Figura 60: Lista de endereços cadastrados para o usuário



Fonte: (DADOS DA PESQUISA, 2020)

Figura 61: Formulário para edição do endereço

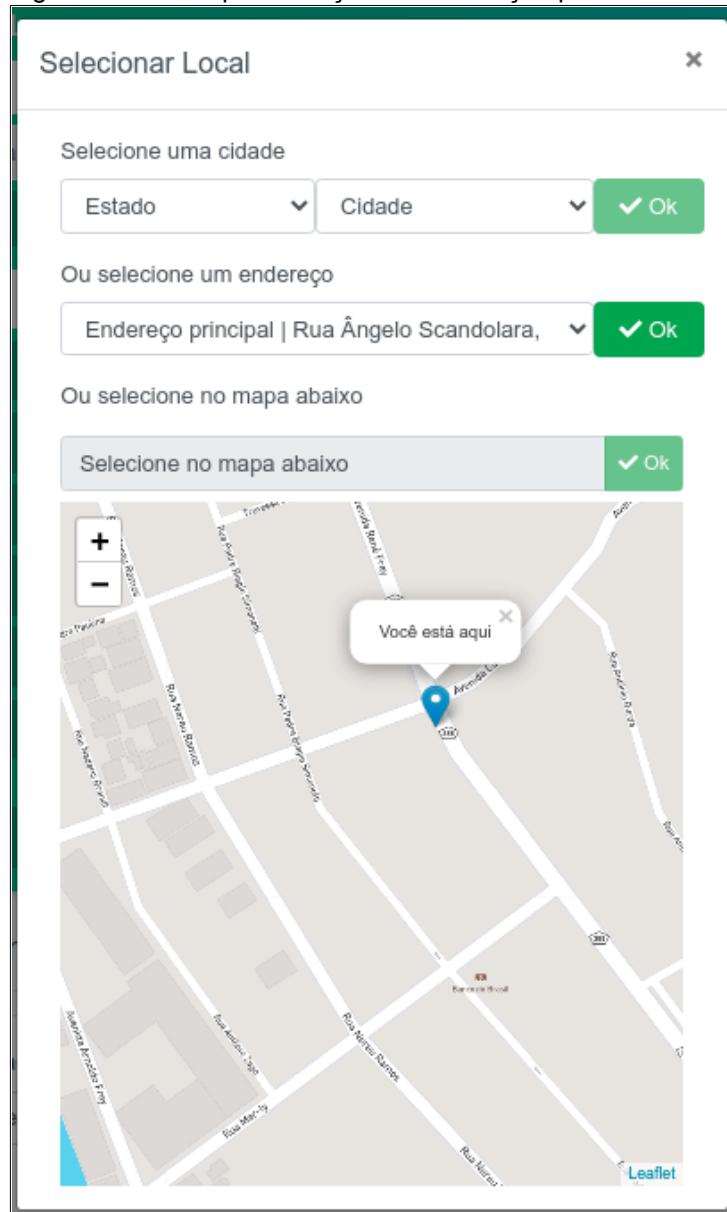
Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.9 Busca

Para realizar a busca por produtos ou produtores, não há obrigatoriedade de informar nenhum filtro. Entretanto, o informe dos filtros refinará os resultados. Como já mencionado, o formulário de busca está disponível em todas as páginas do protótipo, no intuito de facilitar o acesso do usuário à principal funcionalidade da aplicação.

Ao clicar no botão ao lado do campo “Localização”, o sistema abre um *frame* onde são exibidas três opções de localização para o usuário basear sua busca: cidade e estado selecionados, um de seus endereços cadastrados (indisponível para usuários desconectados) ou a seleção de um ponto no mapa. A diferença de tratamento para cada um dos casos é descrita na especificação do caso de uso UC05 – Buscar produtos e produtores.

Figura 62: Frame para seleção da localização parâmetro da busca



Fonte: (DADOS DA PESQUISA, 2020)

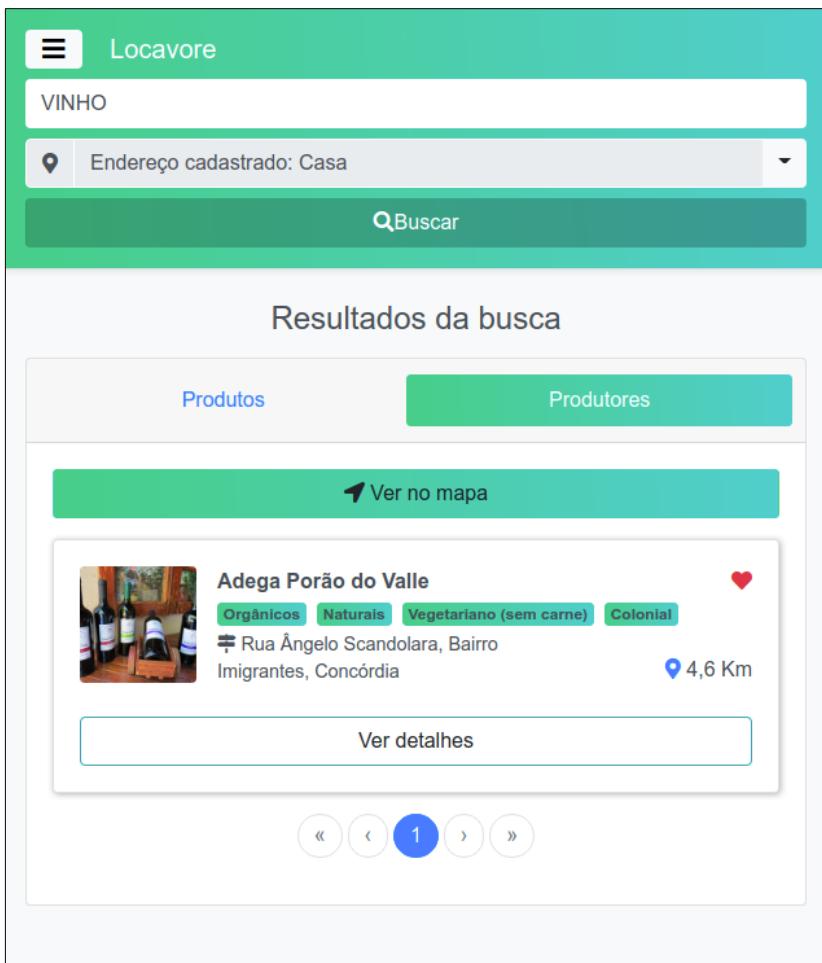
Ao clicar no botão “Buscar” o software aplica os filtros de busca e direciona o usuário para a página com os resultados compatíveis, realizando o caso de uso UC05 – Buscar produtos e produtores.

Figura 63: Resultado da busca por produtos

The screenshot shows the Locavore application interface. At the top, there is a header with a menu icon and the text "Locavore". Below the header, there is a search bar containing the text "VINHO". Underneath the search bar is a dropdown menu showing "Endereço cadastrado: Casa". A green "Buscar" button is located below the dropdown. The main content area is titled "Resultados da busca". It contains two tabs: "Produtos" (selected) and "Produtores". A green "Ver no mapa" button is positioned above the product list. The first product listed is "Vinho colonial suave 1L" from "Adega Porão do Valle" at "Rua Ângelo Scandolara, Bairro Imigrantes, Concórdia". The price is R\$ 22,00 and the distance is 4,6 Km. Below the product details are three hashtags: "#vinho", "#colonial", and "#tag". A "Ver detalhes" button is present. The second product listed is "Vinho colonial seco 1L" from the same location and with the same price and distance. It also has the same hashtags and a "Ver detalhes" button. At the bottom of the results page, there is a navigation bar with icons for previous, next, and last pages.

Fonte: (DADOS DA PESQUISA, 2020)

Figura 64: Resultados da busca por produtores



Fonte: (DADOS DA PESQUISA, 2020)

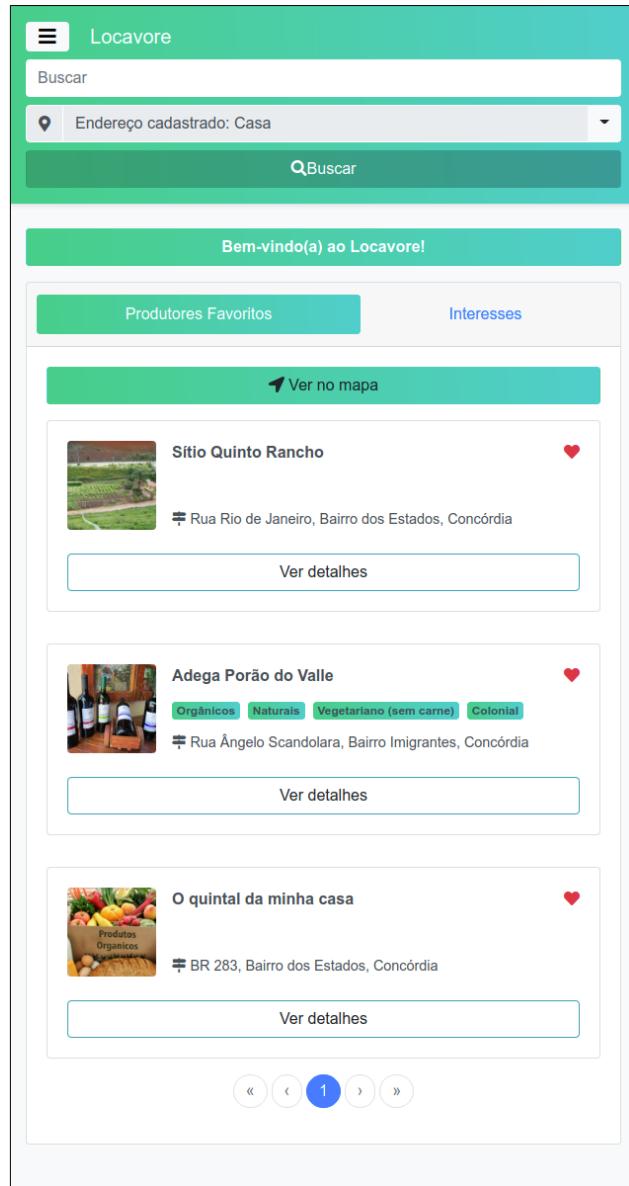
Em ambos os resultados pode-se observar a distância calculada entre o endereço selecionado e o endereço do produtor. Também é possível observar o ícone de coração preenchido no canto superior direito do card do produtor, indicando que ele está favoritado. Clicando no ícone, o produtor é desfavoritado e o ícone de coração fica sem preenchimento, clicando novamente o produtor é favoritado e o ícone volta a estar preenchido. Assim, são atendidos os casos de uso UC11 – Favoritar produtor e UC10 – Desfavoritar produtor. Esses ícones são exibidos também na tela de detalhes do produtor e na página inicial, onde são listados os produtores favoritos.

4.7.2.10 Início

Na tela inicial, realizam-se os casos de uso UC18 – Visualizar página inicial, UC19 – Visualizar produtores favoritos e UC 06 – Visualizar produtos e produtores compatíveis com interesses, conforme demonstrado nas Figuras 65 e 66. Da mesma forma como na

busca, na lista de produtores favoritos é possível desmarcá-los como favoritos e acessar os detalhes, bem como vê-los no mapa. Na lista de resultados compatíveis com interesses, o cálculo de distância é baseada no endereço principal do usuário.

Figura 65: Tela inicial - Lista de produtores favoritos



Fonte: (DADOS DA PESQUISA, 2020)

Figura 66: Tela inicial – Lista de resultados compatíveis com interesses

The screenshot shows the Locavore app's main interface. At the top, there is a navigation bar with a menu icon and the word "Locavore". Below it is a search bar with the placeholder "Buscar" and a dropdown menu set to "Endereço cadastrado: Casa". A green "Buscar" button with a magnifying glass icon is positioned below the search bar. A welcome message "Bem-vindo(a) ao Locavore!" is displayed in a teal header bar. Below this, there are two tabs: "Produtos Favoritos" (in blue) and "Interesses" (in green). The "Produtos" section displays two items:

- Vinho colonial seco 1L** (Colonial) - R\$ 20,00
Adega Porão do Valle
Rua Ângelo Scandolara, Bairro Imigrantes, Concórdia - 4,6 Km
#vinho #colonial #tag
[Ver detalhes](#)
- Vinho colonial suave 1L** (Colonial) - R\$ 22,00
Adega Porão do Valle
Rua Ângelo Scandolara, Bairro Imigrantes, Concórdia - 4,6 Km
#vinho #colonial
[Ver detalhes](#)

Below the products, there is a "Produtores" section featuring a producer profile:

- Adega Porão do Valle**
Orgânicos, Naturais, Vegetariano (sem carne), Colonial
Rua Ângelo Scandolara, Bairro Imigrantes, Concórdia - 4,6 Km
[Ver detalhes](#)

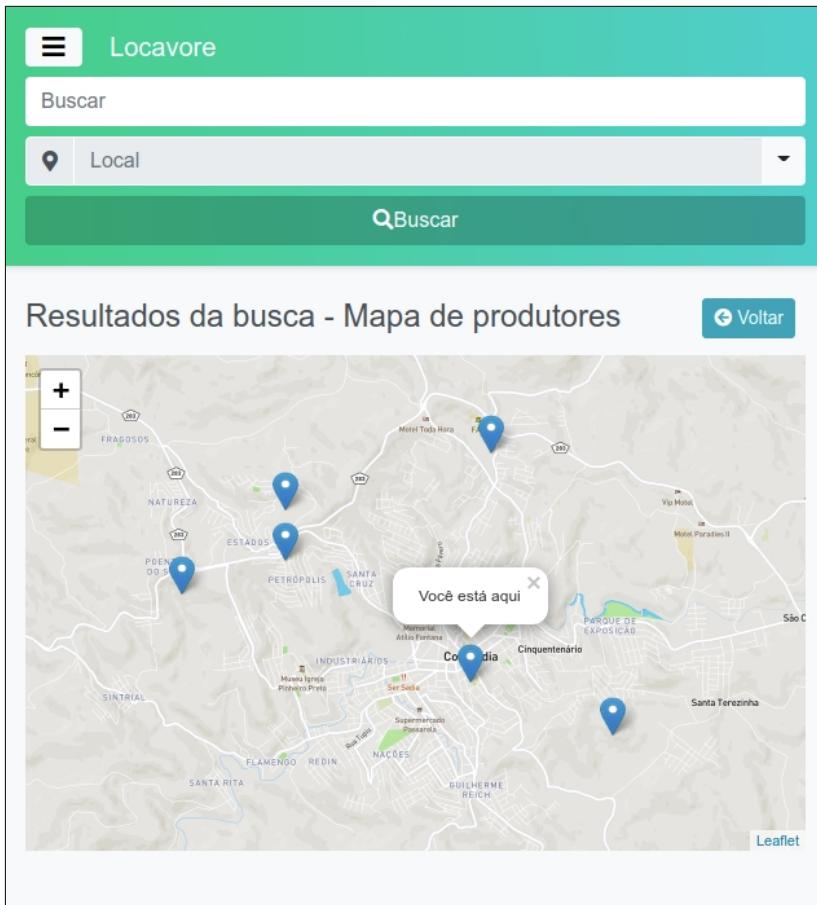
Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.11 Ver no mapa

No resultado da busca e na lista de produtores favoritos, há um botão “Ver no mapa” que ao receber um clique, redireciona o usuário para uma tela onde, conforme a tela de origem (produtores buscados, produtores favoritos ou produtos buscados) serão dispostos os itens no mapa. A exemplo da figura 67, onde os produtores resultantes da

busca são dispostos no mapa (observar que não foi aplicado qualquer filtro). Essa tela atende ao caso de uso UC06 – Visualizar resultados no mapa.

Figura 67: Resultados da busca dispostos no mapa



Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.12 Ver detalhes de produtor

O caso de uso UC01 – Ver detalhes do produtor descreve que na tela inicial, no resultado da busca, bem como na tela de detalhes do produto haverá um botão ou um link que ao ser clicado redireciona o usuário para a tela de detalhes do produtor. Essa tela também atende ao caso de uso UC09 – Avaliar produtor, ao disponibilizar um formulário para avaliação. A interface exibida ao acessar essa tela corresponde à figura 68.

Figura 68: Tela de detalhes do produtor

The screenshot displays the Locavore platform's product details page. At the top, there is a header with a search bar and a dropdown menu showing 'Endereço cadastrado: Casa'. Below the header, the title 'Detalhes do produtor' is displayed, along with a 'Voltar' button. The main content area is divided into several sections:

- Produtor:** Adega Porão do Valle. Below the name are four categories: **Locais**, **Locais**, **Esportes e lazer**, and **Locais**. A small image of wine bottles is shown.
- Detalhes:** A placeholder text block: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque vestibulum mattis nibh non finibus. In finibus vulputate nulla, nec vehicula sem euismod volutpat."
- Galeria de Imagens:** A large image of a vineyard with a wooden sign in the foreground, labeled 'Imagen 1'. Below it is a smaller thumbnail image.
- Tags:** **Locais**, **Locais customizada**, **Ecológico**.
- Endereço(s):** Endereço principal: Rua Angelo Scandolara, 89549, Bairro Imigrantes, Concórdia, Santa Catarina, 89711-162.
- Avaliação:**
 - Avaliar Produtor:** A rating of 4.3 stars is shown.
 - Avaliação:** A rating of 5 stars is shown.
 - Descreva sua avaliação:** An empty text input field.
 - Avaliar:** A green button with a checkmark icon.
 - Avaliação média:** 4.3 stars.
- Avaliações:**
 - Debora Cristina:** 5 stars, 23/10/2020. Review: Muito bom! Nada a reclamar. Excelente atendimento e produtos.
 - Arthur Ipsum:** 5 stars, 20/10/2020. Review: Produtos excelentes, somente o local é de difícil acesso.
 - Joana Lorem:** 5 stars, 19/09/2020. Review: Muito bom! Nada a reclamar. Excelente atendimento e produtos.

Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.13 Ver detalhes de produto

Nos produtos exibidos na tela inicial, no resultado da busca e na lista de produtos do produtor, há um botão “ver detalhes”, que ao ser clicado redireciona o usuário para a tela de detalhes do produto, onde podem ser vistas informações completas do produto, atendendo ao caso de uso UC02 – Ver detalhes de produto e ao UC08 – Avaliar produto.

Figura 69: Interface de detalhes do produto

Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.14 Manter canais de contato

Para cada endereço é possível associar mais de um meio de contato, principalmente para o produtor, onde diferentes endereços podem corresponder a diferentes pontos de venda com diferentes contatos. No menu “Meus endereços”, ao clicar no ícone de contato no endereço cadastrado, são listados os contatos associados,

oferecendo a possibilidade de cadastrar novo contato, editar um existente ou excluir. Essa interface atende ao caso de uso UC22 – Manter canais de contato.

Figura 70: Lista de contatos associados a um endereço

The screenshot shows the Locavore application interface. At the top, there's a header with a menu icon and the text "Locavore". Below the header is a search bar labeled "Buscar". Underneath the search bar is a dropdown menu set to "Endereço cadastrado: Casa". A green search button with a magnifying glass icon and the word "Buscar" is located below the dropdown. The main content area is titled "Meus endereços - Contatos". Inside this area, there's a contact entry: "Tipo: Telefone | Valor: (49) 99999-9999". To the right of this entry are two small buttons: a red one with a trash icon and a yellow one with a checkmark icon. At the bottom of the content area are two buttons: "Voltar" (blue with white text) and "+ Novo" (green with white text).

Fonte: (DADOS DA PESQUISA, 2020)

Figura 71: Formulário de edição de contato

This screenshot shows the Locavore application interface again. The top section is identical to Figura 70. The main content area is titled "Meus endereços - Contatos" and includes an "Editar" link. An edit mode overlay is displayed over the contact list. The overlay has two input fields: "Tipo*" (with a dropdown menu showing "Telefone") and "Valor*" (with the value "(49) 99999-9999"). At the top of the overlay are two buttons: "Cancelar" (yellow with a red X) and "Salvar" (green with a checkmark). The rest of the interface is identical to Figura 70.

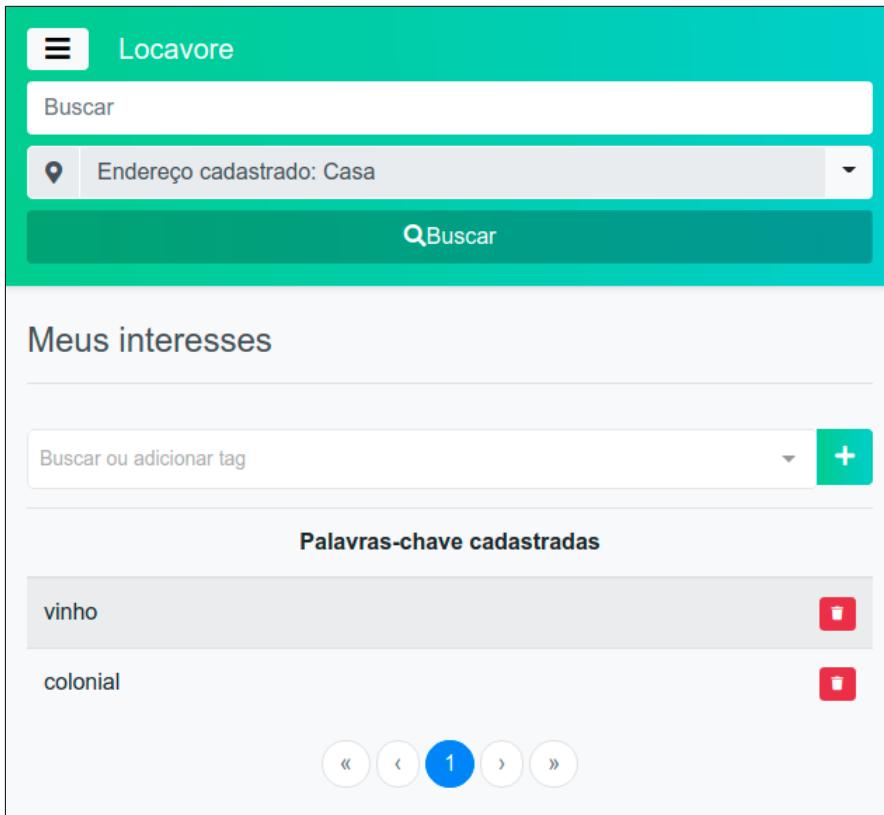
Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.15 Meus interesses

Para visualizar resultados compatíveis com seus interesses na tela inicial, é necessário selecionar as *tags* pelas quais se interessa, realizando o caso de uso UC07 – Manter palavras-chave de interesse. Acessando o menu “Meus interesses” o usuário

visualiza uma lista com as palavras-chave que já cadastrou, junto de uma caixa de texto na parte superior, onde pode adicionar nova.

Figura 72: Lista e cadastro de palavras-chave de interesses



Fonte: (DADOS DA PESQUISA, 2020)

4.7.2.16 Dar sugestão ou *feedback*

Por fim, a tela de *feedback*, acessível a todos os usuários, logados ou não, permite enviar mensagens a equipe. A finalidade principal dessa tela é fornecer um canal de contato para o usuário, por meio do qual deseja-se receber mensagens que contribuam para a melhoria contínua do produto e atendimento das expectativas e necessidades do usuário, atendendo assim o caso de uso UC25 – Dar sugestão ou *feedback*.

Figura 73: Formulário para envio de *feedback* para a equipe de desenvolvimento

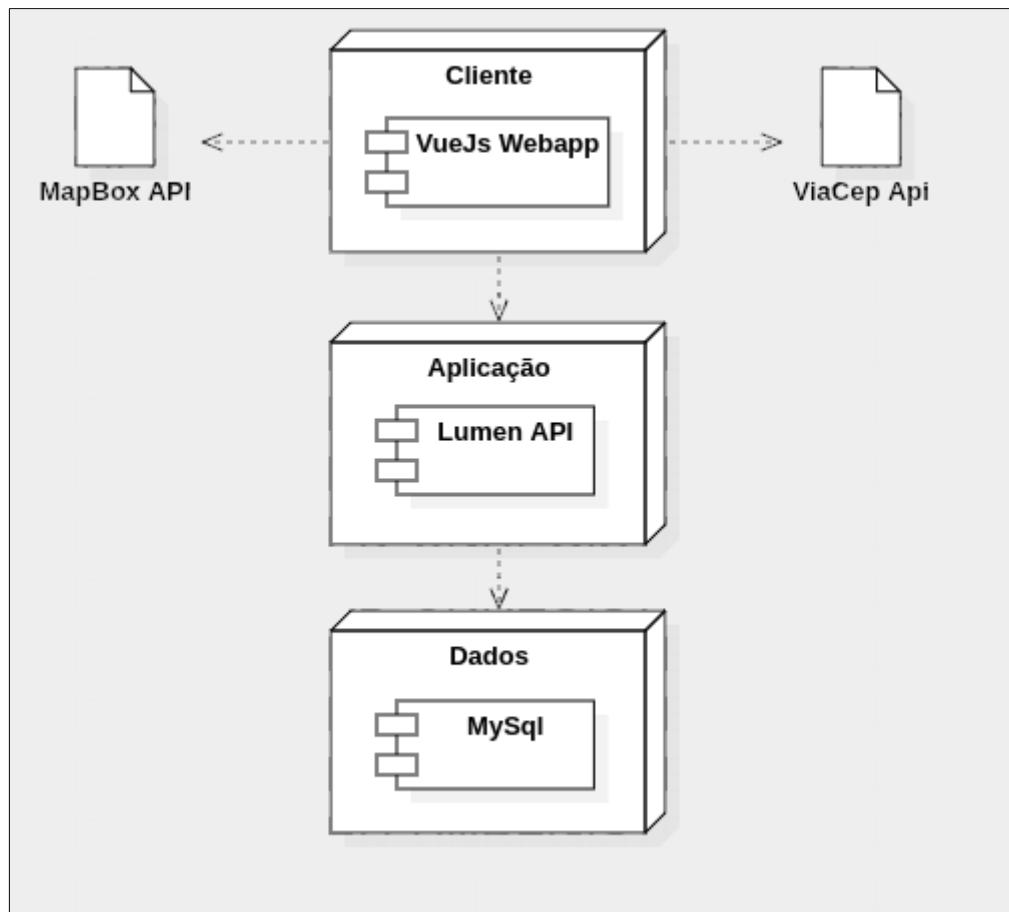
The screenshot shows a mobile application interface for 'Locavore'. At the top, there's a navigation bar with a menu icon and the word 'Locavore'. Below it is a search bar with the placeholder 'Buscar'. Underneath the search bar is a dropdown menu showing 'Endereço cadastrado: Casa'. A green button labeled 'Buscar' with a magnifying glass icon is positioned below the dropdown. The main content area is titled 'Feedback' and contains a text input field with the placeholder: 'Deixe aqui suas avaliações, sugestões de melhorias ou funcionalidades, problemas encontrados, etc. Sua mensagem será recebida por nossa equipe :)' (Leave here your evaluations, suggestions for improvements or features, problems found, etc. Your message will be received by our team :)).' Below this, there are two input fields: 'Seu nome' (Your name) and 'Seu e-mail' (Your email). At the bottom, there is a large text area labeled 'Sua mensagem*' (Your message*) with a red asterisk, intended for the user's feedback. A green 'Enviar' (Send) button with a checkmark icon is located at the top right of the feedback section.

Fonte: (DADOS DA PESQUISA, 2020)

4.8 DIAGRAMA DE COMPONENTES

Em UML, um diagrama de componentes é uma ferramenta para demonstração da estrutura física dos componentes de um sistema e relativamente flexível quando se trata do nível de abstração representado. Nesse sentido, para fins de documentação e consultas futuras, foram desenvolvidos três diagramas de componentes, representando, respectivamente, a arquitetura geral da aplicação, a arquitetura de um módulo (que se aplica aos demais) da aplicação cliente, e a arquitetura de um módulo (também aplicável aos demais) da aplicação responsável pela camada de persistência, a API.

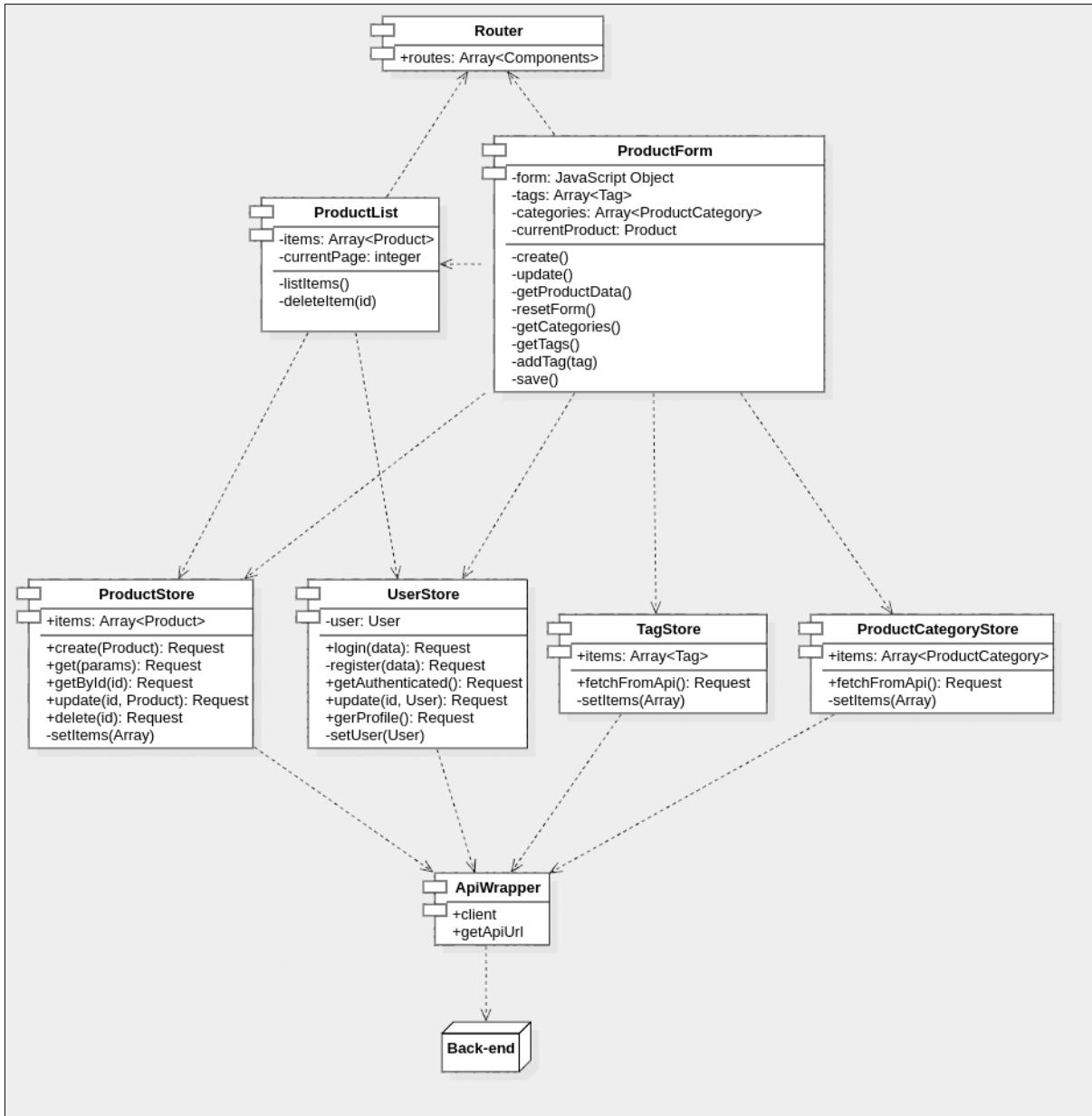
Figura 74: Diagrama de componentes representando a arquitetura geral de componentes da aplicação



Fonte: (DADOS DA PESQUISA, 2020)

No diagrama acima apresentado, cada camada da aplicação está encapsulada em um nó, que pode ser disponibilizado inclusive em diferentes servidores. Dentro de cada nó, há o nome da *framework* ou *software* utilizados para a implementação. Além disso, a aplicação cliente depende de outras duas fontes externas, a API MapBox, utilizada para gerar os mapas exibidos na aplicação, e a API ViaCep, utilizada para consulta de endereços associados ao CEP, melhorando a usabilidade e padronizando os dados de endereço.

Figura 75: Diagrama de componentes do módulo de cadastro de produtos na aplicação cliente

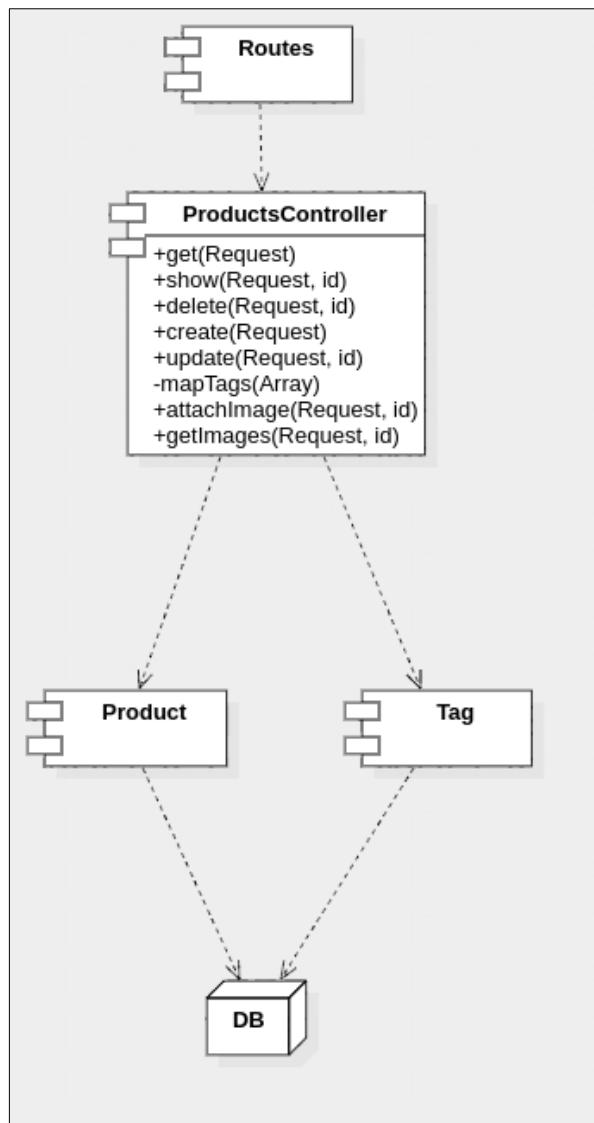


Fonte: (DADOS DA PESQUISA, 2020)

O diagrama acima representa a implementação do módulo de cadastro de produtos, sem incluir o *upload* de imagens. Nele, alguns aspectos devem ser destacados. A iniciar pelo componente Router, que inclui os componentes responsáveis pelo encapsulamento das telas do sistema (nesse caso, *ProductList* e *ProductForm*) e os exibe ao usuário de acordo com a URL requisitada. É comum em aplicações VueJs que as interações com a camada de persistência sejam encapsuladas por módulos chamados de *store*, implementados através de uma biblioteca Javascript chamada *vuex*. Essa biblioteca permite encapsular módulos que podem ser compartilhados por toda a aplicação, evitando duplicidade de código. Por fim, o cliente HTTP responsável pelo envio de

requisições ao *backend* (API) foi também encapsulado em um módulo que pode ser incluído pelos componentes do sistema quando for necessário comunicar-se com a API. Essa arquitetura se aplica a qualquer módulo do sistema que possua interação com a camada de persistência.

Figura 76: Diagrama de componentes do módulo de cadastro de produtos na API



Fonte: (DADOS DA PESQUISA, 2020)

Na camada de aplicação, exemplifica-se a persistência do mesmo módulo representado na camada cliente. Nesse caso, a arquitetura é evidentemente mais simples, implementada em *transaction script*. A requisição passa pelo componente *Routes* que identifica a qual *controller* e função aquela URL e método HTTP correspondem, direcionando para o definido. Por convenção, cada *controller* se responsabiliza pelas funções de um *model*, normalmente as funções CRUD, e demais funções que houverem.

Um componente *controller* tem como responsabilidade controlar as ações de um *model* (nesse caso, *Product* e *Tag*), cada função pública do *controller* desempenha um *transaction script*, que utiliza o modelo, para encapsular funções básicas de interação com o banco de dados e modelos, como relacionamento, criação, pesquisa e exclusão de dados, providas pela biblioteca Eloquent.

5 CONCLUSÃO

5.1 QUANTO A PESQUISA BIBLIOGRÁFICA

A pesquisa bibliográfica compreendeu a maior parte da realização do trabalho, o que evidencia sua importância para um trabalho de pesquisa, além de sua contribuição para a parte teórica e prática do trabalho. Tanto aspectos voltados ao tema do trabalho, como a pesquisa de métodos e técnicas para construção da solução foram de suma importância para fornecer uma base sólida e fundamentada para desenvolver o projeto com maior segurança.

O tema abordado se trata de um movimento relativamente novo e, por consequência, com pouco material disponível. Os materiais consultados estão ligados a pesquisas principalmente de ciências humanas, analisando o locavorismo como um movimento. Nota-se pouco consenso quanto às delimitações do tema, bem como algumas contradições entre autores e até mesmo alguns questionamentos quanto aos seus benefícios. Apesar disso, foi possível identificar alguns aspectos centrais e principais pontos defendidos pelos adeptos do movimento.

Quanto à pesquisa de aspectos voltados à engenharia de *software*, o principal ponto percebido foi no aspecto pessoal, o confronto entre material teórico e métodos e técnicas adquiridos ao longo da vivência acadêmica e profissional e perceber as diferenças entre conhecimento técnico e conhecimento científico, além de reconhecer a aplicabilidade da teoria na prática. Além disso, a pesquisa bibliográfica proporcionou maior contato com produções em engenharia de software, através da qual foi possível perceber a disciplina como bastante flexível, principalmente nas diferenças de visão e abordagens dos autores. Isso evidencia a necessidade de avanço nas pesquisas e necessidade de produção de materiais acerca de assuntos atualizados da disciplina. Entretanto, julgo como um reflexo da característica mutável da disciplina e do grande avanço tecnológico vivenciado, além de se tratar de uma disciplina extremamente jovem se comparada a disciplinas tradicionais como psicologia, direito e afins.

Tal estudo contribui para o curso de Engenharia de Software devido ao relacionamento de teorias fundamentais da disciplina com aspectos tecnológicos mais recentes, para os quais houve fundamentação teórica e consequentemente produção de conhecimento. Além disso, o tema abordado tem sua contribuição social e ao relacioná-lo

com o desenvolvimento de software, destaca-se a função social da engenharia de software como agente de solução de problemas na sociedade.

5.2 QUANTO AO ESTADO ATUAL

Com base na pesquisa bibliográfica acerca do locavorismo, foi possível identificar que se trata de um movimento voltado muito mais a práticas de consumo consciente e sustentável do que o simples consumo de alimento produzidos próximo ao local de consumo. Por se tratar de um movimento orgânico, adquire características conforme se populariza e evolui ao longo do tempo.

Entre os principais objetivos do movimento destacam-se a busca por uma alimentação mais saudável e redução do impacto ambiental, viabilizado através do consumo de alimentos locais. Tal hábito implica em dar preferência ao consumo de produtos provenientes de pequenos produtores, visto que o impacto ambiental desse tipo de cultura é menor do que de latifúndios. Isso contribui para o desenvolvimento da cultura local, além de favorecer a agricultura familiar. Há uma dúvida no que diz respeito ao argumento de que o consumo de alimentos locais reduz a pegada de carbono, porém não se anulam os demais benefícios.

Um aspecto interessante identificado durante a pesquisa é de que, considerados os princípios do locavorismo, não somente o consumo local de alimentos identifica-se como seguindo ao movimento, mas pode ser considerado como o consumo de qualquer produto de procedência na região, cidade ou proximidades do consumidor, que siga os princípios de redução de impacto ao ambiente, favorecimento da economia local e sustentabilidade.

Entre as principais lacunas dificultadoras da expansão do movimento está a dificuldade de conectar produtores com consumidores, nesse sentido encontra-se a aplicabilidade de uma solução de software como a proposta no presente trabalho.

5.3 QUANTO AOS RESULTADOS OBTIDOS

No que diz respeito aos resultados, a pesquisa foi bem-sucedida e obteve-se um protótipo funcional que atenda ao objetivo geral proposto, aplicando métodos fundamentados de engenharia de software. Entretanto, não foi possível atender à totalidade dos objetivos específicos. Foram contemplados completamente os dois

primeiros objetivos, sendo o terceiro contemplado parcialmente, restando o quarto objetivo.

A aplicação foi implementada visando uma interface amigável priorizando dispositivos móveis, porém no que diz respeito à implementação de uma PWA não foi possível atender, pois requer a disponibilização do protótipo em um domínio sob protocolo HTTPS que foi inviabilizado devido ao cronograma. Da mesma forma, por limitações de cronograma e não disponibilização da aplicação em um servidor de hospedagem, não foi possível disponibilizá-lo para testes de aceitação pelo público-alvo.

Outro ponto notável é a diferença na metodologia prevista em comparação à metodologia resultante. Apesar disso, a metodologia prevista foi mantida com o intuito de evidenciar adaptações no processo durante o decorrer do projeto, o que é comum em projetos de software. A primeira diferença foi que, inicialmente, previu-se que a cada iteração de construção do caso de uso mais prioritário, o mesmo seria descrito. Porém identificou-se que seria impossível compreender a prioridade do caso de uso sem descrevê-lo. Na sequência, fora desenvolvido um diagrama de atividade a fim de compreender o fluxo de ações do protótipo, o que não estava previsto. A etapa de construção demonstrou bastante diferença em comparação ao planejamento.

A cada iteração, não houve desenvolvimento do diagrama de sequência e não foram formalizados os testes, apesar disso a aplicação foi testada informalmente conforme o desenvolvimento das funcionalidades. Junto da mudança na estratégica de modelagem, a etapa de construção sofreu uma alteração na linguagem e *framework* escolhidas para construção da camada de aplicação, optando-se pelo uso da *framework* Lumen, sob linguagem PHP, devido à maior familiaridade com as ferramentas e consequente ganho de tempo. Uma particularidade das iterações de desenvolvimento foi de que a primeira iteração dedicou-se exclusivamente à construção da base de dados através de *migrations*. Por fim, não foi possível disponibilizar para teste dos usuários.

Apesar das alterações no cronograma, foi obtido um protótipo funcional, como mencionado. Somente os casos de uso de login e cadastro com redes sociais não foram atendidos. A efetividade no atendimento das necessidades dos *stakeholders* é inconclusiva, porém, como o protótipo se mostrou funcional, é possível em uma próxima etapa disponibilizá-lo para testes de aceitação e identificação de sua efetividade. Uma percepção relevante é de que apesar de um protótipo não requerer necessariamente implementação de funcionalidades de apoio (como cadastro de produtos, endereços e afins), os dados associados às principais funcionalidades o requeriam. Por isso,

pessoalmente, um protótipo funcional se mostrou um trabalho mais extenso do que imaginado.

Por fim, pode-se dizer que apesar das alterações na metodologia e não atendimento da totalidade dos objetivos específicos, o protótipo apresenta funcionalidades genéricas que possibilitam evolução para atendimento das funcionalidades específicas voltadas ao tema caso se mostre necessário. Além disso, apresenta potencial de melhoria e desenvolvimento como um produto de software funcional. Sendo assim, pode-se afirmar que o protótipo obtido tem seu valor para a engenharia de software pois provê uma solução para um problema específico, contribuindo para um movimento com foco na sustentabilidade econômica e ambiental, tema que vem ganhando força atualmente.

5.4 TRABALHOS FUTUROS

Para trabalhos futuros, pode-se mencionar o esforço para atendimento total dos terceiro e quarto objetivos específicos, o que possibilitará a evolução do protótipo para um produto de software. Nesse sentido, menciona-se também a possibilidade de desenvolvimento do protótipo como produto de software aplicado a um modelo de negócio.

Para isso, recomenda-se a realização de algumas tarefas de melhoria do produto, sendo elas: desenvolvimento de validações de formulários, desenvolvimento de proteção de acesso às rotas da aplicação, refatoração do código-fonte, desenvolvimento de aspectos ligados à tecnologia PWA, documentação da API, formalização de testes, construção de testes automatizados e disponibilização da plataforma em homologação ou produção.

Além disso, podem ser integradas novas funcionalidades, como processamento de transações pelo software, ou chat entre usuários. Com a plataforma em produção também será possível identificar outros pontos de melhoria através dos *feedbacks* de usuários e estudo do produto.

REFERÊNCIAS

AZEVEDO, Eliane. **O Ativismo Alimentar na Perspectiva do Locavorismo**. Ambiente & Sociedade. São Paulo, v. 18, n. 3, p. 81-98, jul.-set 2015. Disponível em: <<https://doi.org/10.1590/1809-4422ASOC740V1832015>>. Acesso em: 18 abr. 2020.

BARBOZA, Fabrício Felipe Meleto; FREITAS, Pedro Henrique Chagas. **Modelagem e desenvolvimento de banco de dados**. Porto Alegre: SAGAH, 2018. E-book.

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. 3 ed. Rio de Janeiro: Elsevier, 2015. E-book.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML, guia do usuário**. Rio de Janeiro: Campus, 2000.

BREITMAN, Katin Koogan. **Web semântica: a internet do futuro**. Rio de Janeiro: LTC, 2005. E-book.

CARDOSO, Virgínia; CARDOSO, Giselle. **Sistema de banco de dados: uma abordagem introdutória e aplicada**. São Paulo: Saraiva, 2012. E-book.

CARVALHO, André C. P. L. F. de; LORENA, Ana Carolina. **Introdução à computação: Hardware, Software e Dados**. Rio de Janeiro: LTC, 2017. E-book.

CODE, Visual Studio. **Documentation for Visual Studio Code: Getting Started**. Disponível em: <<https://code.visualstudio.com/docs>>. Acesso em: 3 set 2020;

DEVMEDIA. **UML para Java**. Disponível em: <<https://www.devmedia.com.br/uml-para-java/2711>>. Acesso em: 5 nov. 2020.

FOWLER, Martin. **Patterns of enterprise application architecture**. Addison-Wesley, 2002. E-book.

GALDINO, Fabricio. **VueJs Tutorial**. 2017. Disponível em: <<https://www.devmedia.com.br/vue-js-tutorial/38042>>. Acesso em: 22 jun. 2020.

HALWEIL, Brian. **The Argument for Local Food**. World Watch, p. 20-27, Mai.-jun, 2003. Disponível em: <<https://www.iatp.org/documents/the-argument-for-local-food>> Acesso em: 22 abr. 2020.

LARAVEL.COM. **Database: Migrations**. Disponível em: <<https://laravel.com/docs/8.x/migrations>>. Acesso em: 8 nov. 2020a.

_____. **Eloquent: Getting Started**. Disponível em: <<https://laravel.com/docs/8.x/eloquent>>. Acesso em: 8 nov. 2020b.

LIMA, Matheus. **Introdução aos Progressive Web Apps**. 2017. Disponível em: <<https://medium.com/tableless/introdu%C3%A7%C3%A3o-aos-progressive-web-apps-ad47ba24cddb>>. Acesso em 21 jun. 2020.

LUMEN.LARAVEL.COM. **Installation**. Disponível em: <<https://lumen.laravel.com/>>. Acesso em: 5 nov. 2020.

_____. **Lumen**. Disponível em: <<https://lumen.laravel.com/docs/8.x>>. Acesso em: 5 nov. 2020.

MACHADO, Felipe Nery Rodrigues. **Projeto e implementação de banco de dados**. 3 ed. São Paulo: Érica, 2014. E-book.

MACHADO, Rodrigo Prestes; FRANCO, Márcia H. I.; BERTAGNOLLI; Silvia de Castro. **Desenvolvimento de software III**: programação de sistemas web orientada a objetos em Java. Porto Alegre: Bookman, 2016. E-book.

MANZANO, José Augusto N. G. **Mysql 5.5 interativo**: guia essencial de orientação e desenvolvimento. São Paulo: Érica, 2011. E-book.

MDN WEB DOCS. **CSS básico**. Disponível em:
<https://developer.mozilla.org/pt-BR/docs/Aprender/Getting_started_with_the_web/CSS_basico>. Acesso em: 22 jun. 2020a;

_____. **Fetching data from the server**. Disponível em:
<https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data>. Acesso em: 22 jun. 2020b.

_____. **HTML básico**. Disponível em:
<https://developer.mozilla.org/pt-BR/docs/Aprender/Getting_started_with_the_web/HTML_basico>. Acesso em: 22 jun. 2020c.

_____. **Introdução Express/Node**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introdu%C3%A7%C3%A3o>. Acesso em: 22 jun. 2020d.

_____. **JavaScript**. Disponível em:
<<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>>. Acesso em: 22 jun. 2020e.

_____. **Server-side web frameworks**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First_steps/Web_frameworks>. Acesso em: 22 jun. 2020f.

MYSQL. **MySQL Workbench Manual: 1 General Information**. Disponível em:
<<https://dev.mysql.com/doc/workbench/en/wb-intro.html>>. Acesso em: 11 set 2020;

MILANI, André. **MySQL**: guia do programador. São Paulo: Novatec, 2006. E-book.

PAULA FILHO, Wilson de Pádua. **Engenharia de software**: fundamentos, métodos e padrões. 3 ed. Rio de Janeiro: LTC, 2009. E-book.

PHP. **O que é o PHP? - Manual**. Disponível em:
<https://www.php.net/manual/pt_BR/intro-whatis.php>. Acesso em: 5 nov. 2020.

_____. **O que o PHP pode fazer? - Manual.** Disponível em: <https://www.php.net/manual/pt_BR/intro-whatcando.php>. Acesso em: 5 nov. 2020.

PICOLLO, Lucas. **Vue JS:** o que é, como funciona e vantagens. Disponível em: <<https://blog.geekhunter.com.br/vue-js-so-vejo-vantagens-e-voce/>>. Acesso em: 22 jun. 2020.

PRESSMAN, Roger S; MAXIM, Bruce R. **Engenharia de Software:** uma abordagem profissional. 8 ed. Porto Alegre: AMGH, 2016. E-book.

RAINER JR, R. Kelly; CEGIELSKI, Casey G. **Introdução a Sistemas de Informação:** apoiando e transformando negócios na era da mobilidade. 5 ed. Rio de Janeiro: Elsevier, 2016. E-book.

ROCK CONTENT. **Entenda o que são Progressive Web Apps (PWAs) e veja os melhores exemplos do mercado.** 2019. Disponível em: <<https://rockcontent.com/blog/progressive-web-apps/>>. Acesso em 21 jun. 2020.

RUDY, Kathy. **Locavores, Feminism, and the Question of Meat.** The Journal of American Culture, v. 35, n. 1, p. 26-36, mar. 2012. Disponível em: <<https://doi.org/10.1111/j.1542-734X.2011.00795.x>>. Acesso em: 23 abr. 2020.

SALVADORI, Ivan Luiz. **Desenvolvimento de Web APIs RESTFul Semânticas Baseadas em JSON.** 2015. Dissertação (Mestrado em Ciência da Computação) - Departamento de Informática e Estatística. Universidade Federal de Santa Catarina, Florianópolis, 2015. Disponível em: <<https://repositorio.ufsc.br/bitstream/handle/123456789/132469/333102.pdf?sequence=1&isAllowed=y>>. Acesso em: 22 jun. 2020

SBROCCO, José Henrique Teixeira de Carvalho; MACEDO, Paulo Cesar de. **Metodologias Ágeis:** engenharia de software sob medida. São Paulo: Érica, 2012. E-book.

SCHACH, Stephen R. **Engenharia de software:** os paradigmas clássico e orientado a objetos. 7 ed. Porto Alegre: AMGH, 2010. E-book.

SOMMERVILLE, Ian. **Engenharia de Software.** 9 ed. São Paulo: Pearson Prentice Hall, 2011. E-book.

STARUML.IO. **StarUML.** Disponível em: <<http://staruml.io/>>. Acesso em: 3 set 2020;

THOMPSON, Edward Jr; HARPER, Alethea Marie; KRAUS, Sibella. **Think Globally - Eat Locally.** San Francisco Foodshed Assessment, 2008. E-book.

TRINDADE, Patrícia Esteves; AFFINI, Letícia Passos. **APONTAMENTO A CERCA DO PROGRESSIVE WEB APPS.** In: III Jornada Internacional GEMInIS (JIG 2018). São Paulo, 2019. Disponível em: <<https://www.doity.com.br/anais/jig2018/trabalho/81964>>. Acesso em 21 jun. 2020.

UML.org. **What Is UML**. 2020. Disponível em: <<https://www.uml.org/what-is-uml.htm>>. Acesso em: 13 jun. 2020.

WAZLAWICK, Raul Sidnei. **Engenharia de software**: conceitos e práticas. Rio de Janeiro: Elsevier, 2013. E-book.

WINCKLER, Marco; PIMENTA, Marcelo Soares. **Avaliação de Usabilidade de Sites Web**. In: Escola de Informática da SBC SUL (ERI 2002) ed. Porto alegre: Sociedade Brasileira de Computação, 2002. Disponível em: <<https://www.irit.fr/~Marco.Winckler/2002-winckler-pimenta-ERI-2002-cap3.pdf>>. Acesso em: 23 jun. 2020.

ZENKER, Aline Maciel et al. **Arquitetura de Sistemas**. Porto Alegre: SAGAH, 2019. E-book.