UNIVERSIDADE DE CAXIAS DO SUL

ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GUSTAVO BOFF DA ROSA

DESENVOLVIMENTO DE UM PROTÓTIPO PARA COMBINAÇÃO DE TAXONOMIAS DE CATÁLOGOS ELETRÔNICOS EM UMA PLATAFORMA DE MARKETPLACE

TRABALHO DE CONCLUSÃO DE CURSO

CAXIAS DO SUL

2019

GUSTAVO BOFF DA ROSA

DESENVOLVIMENTO DE UM PROTÓTIPO PARA COMBINAÇÃO DE TAXONOMIAS DE CATÁLOGOS ELETRÔNICOS EM UMA PLATAFORMA DE MARKETPLACE

Trabalho de Conclusão de Curso do Título de Bacharel pela Universidade de Caxias do Sul. Área de concentração: Sistemas de Informação.
Orientador: Prof. Dr. Daniel Luis Notari

CAXIAS DO SUL

2019

GUSTAVO BOFF DA ROSA

DESENVOLVIMENTO DE UM PROTÓTIPO PARA COMBINAÇÃO DE TAXONOMIAS DE CATÁLOGOS ELETRÔNICOS EM UMA PLATAFORMA DE MARKETPLACE

Trabalho de Conclusão de Curso do Título de Bacharel pela Universidade de Caxias do Sul. Área de concentração: Sistemas de Informação.
Orientador: Prof. Dr. Daniel Luis Notari

Aprovado em: 28/11/2019

Banca Examinadora:

_____
Prof. Dr. Daniel Luis Notari
Universidade de Caxias do Sul

_____
Prof. Daniel Antônio Faccin
Universidade de Caxias do Sul

_____
Prof. Mr. Marcos Eduardo Casa
Universidade de Caxias do Sul

AGRADECIMENTOS

Agradeço à minha família, especialmente minha mãe, Inez Maria Boff, e meu pai, Roberto Neves da Rosa, por sempre batalharem em favor da minha educação e felicidade. Todo apoio, incentivo e confiança depositada em mim são os alicerces das minhas realizações.

Aos meus amigos, por todo o incentivo e compreensão com os momentos em que permaneci distante. Às minha amigas, Fabiana Branchini e Karine Seimetz, por nunca negarem palavras de apoio, conselhos e ajuda em momentos difíceis.

Por fim, sou grato ao meu orientador, Daniel Luis Notari, pelo comprometimento e dedicação durante minha vida acadêmica e também durante o desenvolvimento deste trabalho.

RESUMO

O problema abordado neste trabalho envolveu as dificuldades existentes nas integrações entre vendedores e plataformas de *marketplace*. Através de uma pesquisa bibliográfica e análise das documentações relativas às integrações em plataformas de marketplace, foi possível identificar uma exigência por parte das plataformas na integração de produtos via API (*Application Programming Interface*). Essa exigência diz respeito a posição dos produtos na taxonomia do catálogo eletrônico do *e-commerce* a qual corresponde o marketplace. Baseando-se nesta exigência, surgiu a necessidade de combinar as taxonomias dos catálogos eletrônicos de produtos dos vendedores com as taxonomias dos catálogos eletrônicos das plataformas de *marketplace*. Para tal, foi desenvolvido um protótipo para combinação de taxonomias de catálogos eletrônicos. A validação do protótipo se deu através de testes e análises realizadas juntamente a uma empresa, localizada no sul do Brasil responsável por uma plataforma de *markerplace*. Os objetivos esperados com o desenvolvimento do protótipo foram alcançados, além disso, as expectativas do usuário envolvido na validação foram atendidas.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
B2B	*Business to Business*
B2B2C	*Business to Business to Consumer*
B2C	*Business to Customer*
B2E	*Business to Employee*
B2G	*Business to Government*
C2B	*Consumer to Business*
C2C	*Consumer to Consumer*
C2G	*Consumer-to-Government*
CPV	*Common Procurement Vocabulary*
G2B	*Government-to-Business*
G2C	*Government-to-Consumer*
G2G	*Government-to-Government*
JSON	*JavaScript Object Notation*
OECD	*Organization for Economic Co-Operation and Development*
TI	Tecnologia da Informação
TEF	Transferência Eletrônica de Fundos
TIC	Tecnologias de Informação e Comunicação
UML	*Unified Modeling Language*
UNSPSC	*The United Nations Standard Products and Services Code*
URL	*Uniform Resource Locator*
VSM	*Vector Space Model*
DAO	*Data Access Object*
TO	*TransferObject*

SUMÁRIO

1. INTRODUÇÃO

A internet modificou a forma de interação entre empresas, fornecedores e clientes finais. Conforme Galinari *et al.* (2015), as transformações dos últimos anos, nas atividades do varejo, são resultado direto da influência da tecnologia nesse meio, tendo maior destaque às atividades relacionadas ao *e-commerce* devido à dinamicidade e influência sobre empresas e consumidores.

Com a evolução tecnológica possibilitando que diversos tipos de operações pudessem ser realizadas de forma on-line, diversos modelos de negócio surgiram e se desenvolveram. De acordo com Turban et al. (2015), Estes modelos não se referenciam apenas a compra e venda de bens e serviços, mas a realização de todos os tipos de negócios online, por exemplo, atendimentos a clientes, colaboração entre parceiros comercias, e-learning e transações eletrônicas. Esta definição mais ampla de negócios realizados de forma eletrônica pode ser classificada como e-business. O comércio eletrônico propriamente dito pode ser visto como um subconjunto do E-business.

Visando o aumento na rentabilidade, diversos *e-commerce* no Brasil estão optando por novos modelos de negócio, entre eles o *Marketplace*. Nesse modelo, produtos de pequenas lojas são vendidos por grandes varejistas (GUISSONI; OLIVEIRA; TEIXEIRA, 2016). Em troca de um percentual sobre as vendas, as plataformas de *Marketplace* oferecem vantagens aos seus parceiros - como é o caso da Lojas Colombo, um dos *Marketplace* que surgiram recentemente no Brasil. Algumas vantagens oferecidas pelas Lojas Colombo[1] são: redução de custo de marketing, time focado em vendas disponível, divulgação da marca do lojista, alcance de milhares de clientes, plataforma própria e segura, fácil gerenciamento de produtos, sem necessidade de investimento inicial.

Um dos papéis do *Marketplace* é possibilitar e facilitar ao máximo as interações tecnológicas entre a plataforma, fornecedores e consumidores finais. As interações entre os atores envolvidos geram uma série de integrações de recursos tecnológicos e informacionais. A responsabilidade de promover essas integrações é da empresa fornecedora da plataforma (Standing e Standing, 2015). Contudo, apesar do foco do *Marketplace* ser justamente a integração de informações entre

[1]https://www.colombo.com.br/marketplace#section3. Acesso em 29/05/2019.

sistemas de diferentes empresas, muitas organizações falharam ao enfrentar os desafios de integrações complexas envolvendo múltiplos relacionamentos em uma única plataforma (KOPPENHAGEN *et al.*, 2015).

Dentre os diversos processos de negócio existentes em operações de *Marketplace*, a integração de dados referente a produtos tem destaque por se tratar da primeira integração necessária para que um fornecedor inicie suas atividades[2]. A plataforma, por sua vez, tem a função de fornecer e viabilizar esse serviço de integração, intermediando camadas de comunicação e reduzindo o número de mapeamentos necessários. Contudo, para prover esse serviço, faz-se necessário lidar com problemas de heterogeneidade nos produtos e catálogos dos novos fornecedores (FENSEL *et al.*, 2001).

De acordo com Fensel *et al.* (2001), desenvolver uma abordagem para integração de informações tornou-se o principal pré-requisito para viabilizar a escalabilidade do negócio. Essas integrações estão diretamente relacionadas com o gerenciamento de conteúdo eficiente e devem lidar com diversos desafios, tais como extração de informações de fontes irregulares, classificação de informações para tornar os dados do produto passíveis de manutenção e acessíveis, reclassificação de dados de produtos e criação de mapeamentos entre diferentes estruturas de informação. Fensel *et al.* (2001) também afirma que a falta de padrões e heterogeneidade em dados referentes a produtos surgem em, pelo menos, três níveis: o conteúdo, a estrutura do catálogo de produtos e a estrutura de documentação.

1.1. PROBLEMA DE PESQUISA

Através da experiência do autor em processos de integração envolvendo uma plataforma de *marketplace* e múltiplos vendedores, foi possível identificar um dos pontos de maior dificuldade e demora no processo de integração de produtos, que é a combinação de taxonomias. Atualmente, a empresa detentora desta plataforma opta pela contratação de terceiros que disponibilizam ferramentas para auxiliar os vendedores neste procedimento de combinação.

Um dos fatores que dificulta o processo de combinação das taxonomias é que

[2]<https://www.colombo.com.br/marketplace#section4, https://www.b2wmarketplace.com.br/v3/como-funciona>. Acesso em 29/05/2019.

cada vendedor possui um modo de trabalho e uma organização interna distinta, resultando em diversos tipos e formatos de taxonomias de catálogos de produtos. Além disso, existem casos, em empresas de menor porte, que nem sequer possuem um estrutura definida, e é no momento de integrar seus produtos em um *marketplace* que essa necessidade surge.

Todas essas dificuldades observadas pelo autor serviram de motivação para abordar este assunto no presente trabalho.

1.2. OBJETIVOS

Tem-se, como objetivo geral modelar e desenvolver um protótipo de sistema para adequar o processo de categorização de produtos de uma plataforma de *Marketplace* com seus fornecedores.

Tem-se, como objetivos específicos:

a) Pesquisar sobre o cenário do *e-commerce* e modelos de negócio.
b) Pesquisar sobre o cenário de plataformas de *Marketplace* e os participantes envolvidos.
c) Pesquisar sobre sistemas de representação do conhecimento.
d) Pesquisar sobre o processo de combinação de taxonomias.
e) Analisar métodos para categorização de produtos em taxonomias.
f) Elaborar o projeto de solução.
g) Desenvolver um protótipo de um sistema para combinação de taxonomias.
h) Validar o protótipo.

1.3. ESTRUTURA DO TRABALHO

Este trabalho está dividido nos seguintes capítulos: Introdução, referencial teórico, integração de produtos em marketplaces via API, proposta de solução, desenvolvimento da proposta de solução e considerações finais. No primeiro capítulo é apresentada a introdução, motivação e objetivos do trabalho. No segundo e terceiro capítulos, são abordados os temas: *e-commerce*, *marketplace* eletrônico, integrações de produtos em *marketplaces* e combinação de taxonomias e ontologias em plataformas de *marketplace*. O quarto capítulo apresenta a proposta de solução,

juntamente com a modelagem do sistema. O quinto capítulo detalha o processo de desenvolvimento do protótipo de sistema, além dos testes realizados. Por fim, são apresentadas as considerações finais e trabalhos futuros.

2. REFERENCIAL TEÓRICO

No referencial teórico, serão abordados os principais conceitos e temas necessários para o entendimento do problema de pesquisa e, por conseguinte, da proposta de solução apresentada. Essa etapa do trabalho foi dividida em três principais capítulos: *E-commerce*, *Marketplace* eletrônico e Combinação de taxonomias e ontologias em plataformas de *marketplace*.

2.1. *E-COMMERCE*

Este capítulo tem o objetivo de situar o leitor em relação aos benefícios e importância do *e-commerce* no meio econômico. Além disso, os tipos de *e-commerce* e os tipos de transações que os definem são relacionados.

2.1.1. Evolução do *e-commerce* no meio econômico

As atividades envolvendo o comércio estão em constante evolução. Tais evoluções trazem consigo diversos benefícios a pessoas e organizações. Conforme Lie *et al.*(2017), no passado, as negociações entre cliente e vendedor ocorriam unicamente através do encontro físico entre esses dois agentes. Contudo, atualmente, essas transações envolvendo compra e venda de bens podem ser realizadas totalmente *on-line* através do *e-commerce*. De acordo com OECD (*Organization for Economic Co-Operation and Development*) 2011, as soluções de *e-commerce* eram limitadas a comunicações entre grandes empresas em setores específicos que abriram canais de comunicação dedicados. Com o passar dos anos, os benefícios do *e-commerce* foram disseminados a qualquer pessoa ou empresa com acesso à internet, possibilitando que transações sejam realizadas entre as empresas e clientes finais a qualquer momento, em qualquer lugar com acesso à rede.

O desenvolvimento da tecnologia mudou a forma de interação entre empresas e consumidores. Além disso, a tecnologia proporciona ferramentas para que processos organizacionais sejam constantemente aprimorados. Conforme afirma Galinari *et al.* (2015, p. 136),

> com o advento da internet e de outras tecnologias genéricas, inúmeras aplicações vêm alterando não apenas a forma de comercialização, mas

> também diversas práticas associadas à administração da firma varejista, à gestão de cadeias de fornecimento, ao marketing, às formas de pagamento e ao relacionamento com clientes.

As evoluções no *e-commerce* estão diretamente relacionadas com as evoluções tecnológicas de cada período. Dentre as tecnologias de maior impacto nesse processo evolutivo, destacam-se as Tecnologias de Informação e Comunicação (TIC). Levando em consideração o desenvolvimento do *e-commerce* até os dias atuais, pode-se dividir a evolução observada em três fases (Galinari *et al.*, 2015). Na primeira, o *e-commerce* se limitava a transações entre grandes organizações através de conexões privadas e por meio de Transferência Eletrônica de Fundos (TEF). Com a popularização da internet e o desenvolvimento tecnológico relacionado a pagamentos *on-line*, ao barateamento de aparelhos de informática e ao desenvolvimento de aplicações de *e-commerce* mais atrativas, iniciou-se a segunda fase. Esse momento do *e-commerce* foi marcado pelo florescimento do comércio entre empresas e clientes finais. Ademais, as transações entre empresas evoluíram, aumentando em número e complexidade - isso porque não somente empresas de grande porte, como também empresas menores, estariam em contato através da internet, criando, assim, cadeias de fornecimento complexas. A última fase ainda está em andamento e é caracterizada pela presença massiva de dispositivos móveis que, hoje, representam grande parte dos acessos em lojas virtuais e internet como um todo (Galinari *et al.*, 2015).

O aumento de transações entre empresas e clientes finais - B2C (*business-to-customer*) - impulsionaram o desenvolvimento de tecnologias que que visam facilitar e estimular transações no meio eletrônico. Segundo Turban *et al.* (2015), o início do *e-commerce* B2C foi marcado por transações pequenas, de itens que poderiam ser facilmente enviados aos clientes como, por exemplo, livros, softwares, músicas, entre outros. A segunda fase de evolução veio por volta do ano 2000, na qual itens maiores e de maior valor agregado começaram a ser comercializados, como eletrodomésticos, móveis e roupas. Hoje em dia, os consumidores têm a possibilidade de pesquisar, a partir de categorias, todos os tipos de itens, desde joias até materiais de construção. Além disso, as vendas de serviços, como cursos *on-line*, cursos de ensino superior, seguros e muitos outros, estão presentes em diversas lojas virtuais. O *e-commerce* vem ganhando cada vez mais espaço e, conforme afirma a OECD 2011, trata-se de um mecanismo fundamental para o suporte e crescimento da atividade econômica.

2.1.2. Benefícios e dificuldades do *e-commerce*

Conforme afirma a OECD 2011, a mudança estrutural no comércio que o *e-commerce* está liderando afeta os negócios em diversos sentidos, entre eles: reduzindo custos operacionais, ampliando escopo de mercado e facilitando a adesão de empresas nesse modelo de negócio e, consequentemente, aumentando a concorrência. Para as empresas que já utilizam *e-commerce,* existe a necessidade de desenvolver novas competências para se manter no mercado cada vez mais competitivo.

Dentre os benefícios da adesão do *e-commerce* por empresas, destaca-se a redução de custos operacionais. Além disso, o comércio *on-line* corrobora para que as organizações trabalhem com o varejo de forma multilocal, conforme afirmam Galinari *et al.* (2015, p. 140):

> A atividade proporciona também economias relativas ao transporte e estocagem de mercadorias. Novos modelos de negócios, como os das empresas que comercializam produtos digitais, como e-books, músicas, filmes, imagens, base de dados, softwares etc., operam com custo de transporte praticamente nulo. As que transacionam mercadorias físicas estão sujeitas a menores custos de estocagem, dado que seus produtos podem ser mantidos em poucos centros de distribuição, simplificando o complexo gerenciamento de estoques que se observa no varejo multilocal.

Contudo, os benefícios do *e-commerce* não se resumem apenas à redução de custos. Galinari *et al.* (2015) também destacam a possibilidade de acompanhamentos das vendas e dos perfis dos clientes em tempo real, possibilitando as análises de tendências do mercado. Outra vantagem destacada pelos autores é o aumento significativo de escopo de atuação como potencializador de vendas. Através de lojas *online*, a empresa pode ofertar seus produtos em todo território nacional, e até mesmo em outros países. Por fim, é destacada a ampliação de escopo temporal, uma vez que as lojas podem permanecer em operação durante as 24 horas do dia, todos os dias do ano.

Para a OECD 2011, as vantagens do *e-commerce* para os consumidores estão relacionadas à disponibilidade de uma ampla variedade de produtos, economia de tempo e de custo de deslocamento até uma loja física, entrega imediata de produtos digitais, redução das barreiras de tempo (*sites* operam 24 horas por dia), facilidade na comparação de preços, entre outras.

Por fim, a implementação de uma loja virtual se torna muitas vezes mais fácil e barata do que uma loja física, como explicam Galinari *et al.* (2015). Ademais, os autores explicitam o fato de que empresas que dispõem de menos capital para investimento podem entrar no *e-commerce* mesmo sem o próprio site. Para tal, podem optar pela abertura de uma loja em redes sociais ou através da adesão em um *e-marketplace*, que será detalhado mais a frente neste trabalho.

Apesar de todos os benefícios que o *e-commerce* pode proporcionar ao negócio, dispor de capital de investimento para construção ou adoção de um sistema de *e-commerce* pode ser um problema para muitas empresas, mesmo que, na maioria das vezes, seja um investimento menor do que se espera para a abertura de uma loja física. Segundo Lie *et al.* (2017), além da questão referente a capital de investimento inicial, outro fator impeditivo para utilização do *e-commerce* é a dificuldade e custo na realização do *marketing*. Ademais, a disponibilidade de infraestrutura de TI (Tecnologia da Informação) e sua implementação também são condições determinantes para o sucesso do *e-commerce* e exigem atenção por parte das organizações.

2.1.3. Tipos de *e-commerce*

Altualmente, o *e-commerce* pode ser dividido em diversas categorias. De acordo com Silveira e Silveira (2015), essa divisão ocorre em decorrência do aumento do *e-commerce* e da complexidade das transações realizadas - isso facilita não apenas o entendimento, como também a realização de estudos específicos.

Além de tipos de transações realizadas, os participantes envolvidos também podem influenciar na classificação do tipo de *e-commerce*. “Transações de *e-commerce* podem envolver diversas classes de agentes econômicos. Dependendo da natureza das partes envolvidas, essas transações recebem diferentes denominações [...].” (Galinari *et al.*, 2015, p. 137).

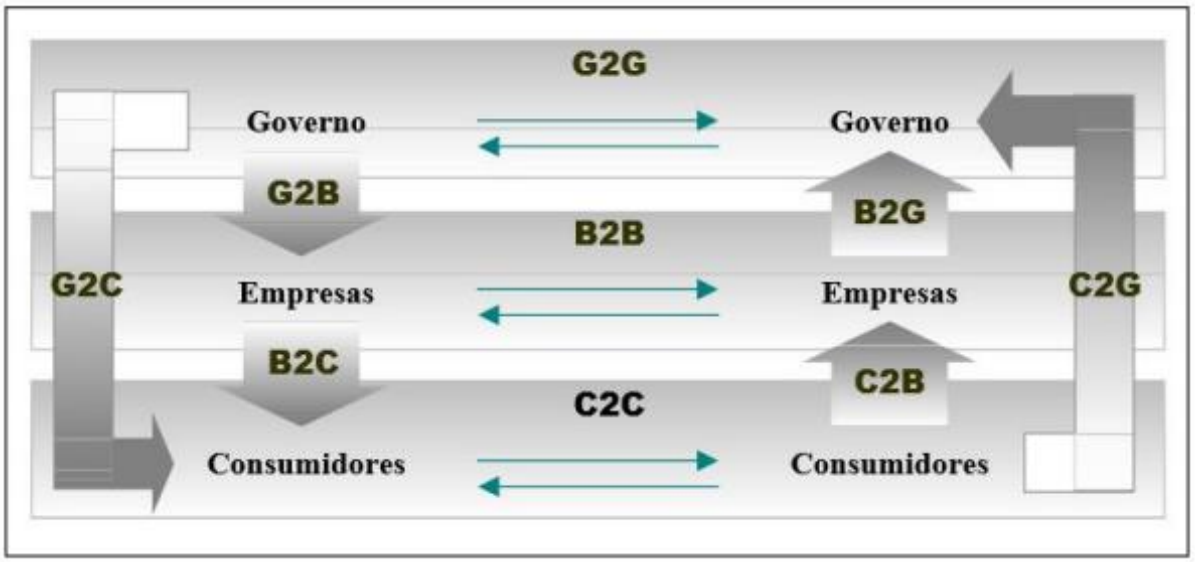
Conforme a tecnologia e o mercado econômico evoluem, novos tipos de *e-commerce* são criados. Na literatura, os autores destacam diversas classificações diferentes: Galinariet al. (2015), Silveira e Silveira (2015) e Mendonça (2016) destacam o *e-commerce business to business* (B2B), *business to customer* (B2C), *business to government* (B2G) e *consumer to consume*r (C2C). No *e-commerce* B2B, as negociações ocorrem entre organizações. Esse tipo de transação pode ser

realizada através de sistemas que operam em redes públicas ou redes privadas partilhadas pelas empresas. O *e-commerce* B2C é o mais conhecido pelos usuários em geral. É caracterizado pela venda direta entre fabricantes e distribuidores ao consumidor final. Geralmente, as transações são realizadas através de um sistema disponibilizado por alguma organização. Em transações B2G, empresas vendem diretamente para o governo. No formato C2C, as vendas ocorrem diretamente entre consumidores, geralmente através de uma plataforma que promove a intermediação da operação.

Silva e Filho (2017) e Silveira e Silveira (2015) também qualificam o *consumer to business* (C2B), no qual consumidores negociam com empresas. Do mesmo modo, Mendonça (2016) define o tipo *business to employee* (B2E), em que as empresas vendem seus produtos para seus próprios funcionários através de plataformas. Silveira e Silveira (2015) também conceituam algumas relações entre consumidores ou empresas com orgãos governamentais: *Consumer-to-Government* (C2G), no qual consumidores vendem para o governo; *Government-to-Business* (G2B), no qual o governo vende diretamente para organizações; *Government-to-Consumer* (G2C), no qual o governo vende para consumidores e *Government-to-Government* (G2G), envolvendo transações entre órgãos do governo.

Alguns modelos de negócio podem envolver diversos tipos de transação, como é o caso do *e-commerce business to business to consumer* (B2B2C). Silva e Filho (2017) explicam que, nesse tipo de *e-commerce*, o agente (B2) pode vender produtos ou realizar a prestação de serviços a um agente (B1). Por fim, há a transção com o consumidor final. O modelo de negócio de *marketplaces* eletrônicos se enquadra nesse tipo. Na Figura 1, é possível vizualizar as relações de negociação de alguns tipos de *e-commerce* citados.

Figura 1 – Inter-relações de negociações no comércio eletrônico e os agentes envolvidos.



Fonte: Silveira e Silveira (2015); Rocha et. al. (2002).

2.2. *MARKETPLACE* ELETRÔNICO

A evolução das tecnologias envolvendo *e-commerce* é desenvolvida constantemente. Esse desenvolvimento tecnológico resulta na criação de novos modelos de negócio, envolvendo transações cada vez mais complexas. Um exemplo desses novos modelos de negócio provenientes da evolução tecnológica é o *marketplace* eletrônico. Neste capítulo, o *marketplace* será conceituado e relacionado ao *design* de negócio de plataforma. Por fim, os principais participantes e componentes envolvidos serão relacionados.

2.2.1. Conceito de *marketplace* eletrônico

Lie *et al.* (2017) definem que um *marketplace* eletrônico (*e-marketplace*) é um portal de *e-commerce* que busca reunir empresas como vendedores (*sellers)* para ofertar e comercializar seus produtos. Nesse portal, o consumidor pode buscar os produtos desejados e realizar a compra, efetuando o pagamento. Igualmente, Turban *et al.* (2015) afirmam que os vendedores podem vender diretamente de seus *sites* ou através de um *e-marketplace*. Existem diversas empresas que optam pelas duas formas de trabalho, mantendo um *e-commerce* próprio e participando de

*marketplac*es. Um exemplo é a empresa Madesa Móveis[3], que possui um *e-commerce* próprio[4], mas também participa de *marketplaces* como Lojas Colombo[5], Americanas[6], Submarino[7] e Mercado Livre[8].

Sob o mesmo ponto de vista, Kolomvatsos, Anagnostopoulos e Hadjiefthymiades (2014) afirmam que as entidades distribuídas envolvidas têm por objetivo a cooperação, a fim de atingir objetivos comuns. Segundo os autores, os *marketplaces* eletrônicos são altamente dinâmicos, isto é, o número e o comportamento das entidades envolvidas variam muito ao longo do tempo. Além disso, existem outros agentes importantes para o mantimento das relações entre vendedores e *marketplace*, conhecidos como intermediários. Entidades intermediárias visam facilitar o processo de interação entre os demais agentes.

Atualmente, diferentes tipos de *marketplace* estão disponíveis no mercado, abrangendo as mais diversas necessidades de negócio. Segundo Standing e Standing (2015), a presença de *marketplaces* em diferentes tipos de negócio suportam diferentes tipos de transação como B2B e B2C. Contudo, Turban *et al.* (2015) afirmam que a distinção entre B2C e B2B quanto a *e-commerce* pode não ser clara, o que se reflete também em processos de *e-marketplace.* Lie *et al.* (2017) classificam *marketplaces* eletrônicos como B2B2C justamente pelo fato de compreender múltiplos modelos de negócio e tipos de transação.

De acordo com Standing e Standing (2015), o *marketplace* oferece a oportunidade de executar transações por meio de canais eletrônicos, geralmente através de uma plataforma *web*. Chang e Wong (2010) complementam classificando o *e-marketplace* como uma plataforma organizacional que permite que os participantes troquem informações sobre preços e ofertas de produtos. Conforme Standing e Standing (2015), o fornecedor da plataforma de *marketplace* é o responsável pela integração de recursos tecnológicos e informacionais dos seus clientes e deve facilitar a integração de recursos o máximo possível.

Segundo Kutera e Gryncewicz (2017), e-*marketplaces* de serviço requerem um conjunto de pontos de acesso à API que compreendam dois alicerces básicos: o

[3]<https://www.madesa.com/> Acesso em 05/06/2019.
[4]<https://www.madesa.com/> Acesso em 29/05/2019.
[5]<https://www.colombo.com.br/lojista/Madesa> Acesso em 29/05/2019.
[6]<https://www.americanas.com.br/marca/madesa> Acesso em 29/05/2019.
[7]<https://www.submarino.com.br/marca/madesa> Acesso em 29/05/2019.
[8]<https://loja.mercadolivre.com.br/madesa-moveis> Acesso em 29/05/2019.

primeiro envolvendo dados e lógica de negócios, e o segundo envolvendo segurança. Essa estrutura é válida também para *marketplaces* voltados ao varejo, como pode ser observado no markeplace das Lojas Colombo[9]. Conforme os autores, em ambientes homogêneos, as integrações de dados e de processos relativos a lógicas de negócio entre vendedores e plataforma de *marketplace* poderiam ser suportados pela plataforma. Contudo, em geral, isso não ocorre, e há um desafio em encontrar a solução comumente aceita para esse problema. É por isso que é possível permitir que alguns processos sejam executados somente em serviços de terceiros, que são totalmente especializados em tais processos.

Diversos benefícios podem ser identificados na adoção de *marketplaces,* tanto para vendedores, quanto para a empresa detentora da plataforma e clientes finais. De acordo com Jiang e Balasubramanian (2014), *marketplaces* eletrônicos ampliam a oferta de produtos, facilitam as comparações de preço e disponibilizam uma quantidade maior de informações referentes a produtos. Ademais, Turban *et al.* (2015) afirmam que a melhoria e a redução de custos relacionadas a processos de logística, precificação e disponibilização de produtos aumentam a eficiência de mercado do *e-commerce* em geral.

2.2.2. Participantes e componentes

Por ser tratar de uma plataforma que busca viabilizar e facilitar ao máximo as interações entre diferentes agentes ligados ao negócio, as partes envolvidas em processos de negócio do *marketplace* possuem grande importância e precisam ser cuidadosamente estabelecidas. De acordo com Turban *et al.* (2015), os principais componentes e participantes em um *e-marketplace* são clientes, vendedores (*sellers*), varejista eletrônico, produtos e serviços (físicos ou digitais), infraestrutura, *front-end, back-end*, intermediários e outros parceiros de negócios, além de serviços de suporte, como segurança e pagamentos.

Conforme explicam Turban *et al.* (2015), um varejista é um intermediário de vendas entre fornecedores e clientes. Embora muitos fabricantes vendam diretamente para os consumidores, eles costumam fazê-lo para complementar suas principais

[9]Conjunto de de pontos de acesso à API relacionados a dados e lógica de negócios – disponível em: <https://api.marketplace.colombo.com.br/swagger-ui.html>; e enpoint relativo à segurança e autenticação:<https://api.marketplace.colombo.com.br/swaggerui.html?urls.primaryName=marketplace-authentication>. Acesso em 29/05/2019.

vendas através de atacadistas e varejistas. No mundo físico, o varejo é feito em lojas (ou fábricas) que os clientes devem visitar fisicamente para fazer uma compra, embora, às vezes, seja possível fazer pedidos por telefone. O varejo conduzido pela Internet é chamado de varejo eletrônico (*e-tailing*) e os vendedores que fazem negócios *on-line* de varejo são chamados de varejistas *on-line* (*e-tailers*). No caso do *marketplace*, o varejista eletrônico geralmente é a empresa proprietária da plataforma de *marketplace*. Os clientes representam os mais de 2 bilhões de usuários da Internet em todo o mundo. Todos esses usuários são potenciais compradores de bens e serviços oferecidos na Internet. Esses consumidores estão procurando promoções, itens personalizados, entretenimento e muito mais. Empresas também são clientes em potencial e, hoje, representam mais de 85% do volume de *e-commerce*. Já os vendedores podem ser representados por lojas virtuais de propriedade de empresas, agências governamentais ou indivíduos. São esses vendedores que optam por participar de *marketplaces*. Por fim, um intermediário é tipicamente uma empresa terceira que opera entre os demais participantes. Intermediários de todos os tipos oferecem seus serviços na *Web*.

Referente aos componentes relacionados, os autores afirmam que os que possuem maior relevâncias para este estudo são os produtos e serviços, infraestrutura, *front-end* e *back-end*. Os produtos e serviços se referem às ofertas disponibilizadas aos clientes através do *marketplace*. A infraestrutura de *marketplace* inclui redes eletrônicas, bancos de dados, *hardware*, *software*, entre outros elementos. O *front-end* representa a forma de interação gráfica disponibilizada aos clientes. Os principais componentes do *front-end* podem incluir o portal do vendedor, catálogos eletrônicos, um carrinho de compras, um mecanismo de pesquisa, um mecanismo de leilão, um *gateway* de pagamento, etc. Por fim, o *back-end* do negócio compreende todas as atividades relacionadas a agregação e cumprimento de pedidos, gerenciamento de estoques, compras de fornecedores, contabilidade e finanças, seguros, processamento de pagamentos, embalagem e entrega.

As plataformas de *marketplace* são construídas pensando na forma mais eficiente de integrar vendedores, possibilitando que uma ampla quantidade de produtos seja disponibilizada para venda no *e-commerce* na empresa responsável. Contudo, conforme afirma Koppenhagen *et al.* (2015), apesar do foco do *marketplace* ser justamente a integração de informações entre sistemas de diferentes empresas,

muitas organizações falharam ao enfrentar os desafios de integrações complexas envolvendo múltiplos relacionamentos em uma única plataforma. Portanto, cada processo, tecnologia e agente intermediário devem ser cuidadosamente analisados e escolhidos.

2.2.3. Plataformas como *design* de negócio

No *design* de negócios de plataformas, os envolvidos nos processos do negócio interagem, gerando valor. Segundo Choudary (2015), as plataformas possuem duas funções principais: a primeira é fornecer uma estrutura aberta, participativa e *plug-and-play*, possibilitando a integração entre os participantes das atividades do negócio; a segunda é organizar e gerenciar os participantes e as interações sociais e econômicas que ocorrem através da plataforma. De modo geral, o objetivo da plataforma é possibilitar e gerenciar as interações entre os agentes envolvidos em processos de negócio de forma repetida e eficiente.

Atualmente, conforme explicam Choudary (2015), plataformas sociais como *Facebook, YouTube* e *Twitter* permitem que os usuários criem conteúdo e interajam entre si. *Marketplaces* como o *eBay* e o *Etsy* facilitam as interações remotas. Algumas plataformas, como o *Tinder* e o *Airbnb*, facilitam as interações entre pessoas. Outras, como *Uber* e *Munchery*, gerenciam o movimento de recursos do mundo real em tempo real. Todas essas plataformas executam as duas funções principais mencionadas.

A característica mais notável das plataformas é a sua estrutura projetada para fácil conexão. Contudo, conforme afirma Choudary (2015), essa característica de conduzir interações por meio de uma infraestrutura colaborativa é bastante desafiadora. Uma forma de simplificar as integrações necessárias é através da utilização de APIs. Cada vez mais as APIs estão possibilitando uma nova forma de desenvolvimento de negócios, evitando a necessidade de integrações complexas e, em alguns casos, acordos contratuais complexos.

2.3. COMBINAÇÃO DE TAXONOMIAS E ONTOLOGIAS EM PLATAFORMAS DE *MARKETPLACE*

Existem diversas integrações necessárias para que um vendedor possa realizar vendas através de um *marketplace*. Destas, a integração de produtos é a primeira que

deve ser realizada[10]. Para que essa integração tenha sucesso, o vendedor deve integrar seus produtos, respeitando a hierarquia e organização de categorias do *e-commerce* relacionado ao *marketplace*. Devido a tais fatores, faz-se necessária a combinação das estruturas de catálogo eletrônico existentes.

Neste capítulo, serão apresentados dois tipos de estruturas de organização do conhecimento utilizadas por sistemas de *e-commerce*. Além disso, será apresentada a relação entre taxonomias e integrações em plataformas de *marketplace*. Por fim, serão relacionados alguns métodos, mecanismos e metodologias adotadas por outros autores com a finalidade de combinar diferentes estruturas de informação.

2.3.1. Sistemas de organização e representação do conhecimento em ambientes digitais

Como resultado da utilização massiva da tecnologia nos processos organizacionais, percebe-se a necessidade de estruturar as informações armazenadas. De acordo com Steimer e Luz, “a partir do desenvolvimento e da inovação nas tecnologias de comunicação e informação, a gestão e organização da informação têm tido um papel cada vez mais central e decisivo seja no mercado, na academia e também na vida em sociedade como um todo.” (2015, p.3).

No campo de gestão do conhecimento, Luz (2010) e Steimer e Luz (2015) afirmam que dificilmente as empresas possuem uma base de dados única, portanto, não há um ponto de acesso unificado às informações. Isso pode gerar um impacto negativo às organizações em longo prazo.

Empresas que trabalham diretamente com o comércio de bens e serviços necessitam de uma organização simples e efetiva de suas informações, principalmente no que tange a catálogo de produtos ou serviços. Segundo Steimer e Luz (2015), essa organização pode ser realizada através da classificação estruturada baseada no conceito de Tipos de Produto e seu conjunto de atributos. Nesse sentido, segundo Vital e Café (2011), podem ser utilizadas as ontologias e taxonomias - dois sistemas de organização e representação do conhecimento. Cavalcante e Bräscher (2014) complementam, definindo a categorização como o processo de classificação de elementos a partir de atributos em comuns. De acordo com os autores, as

[10] <https://www.colombo.com.br/marketplace#section4>. Acesso em 15/06/2019.

taxonomias são um instrumento dedicado a esse fim.

Segundo Vital e Café (2011), as taxonomias agem no sentido de organizar a informação em relações hierárquicas entre os termos. Já as ontologias têm como objetivo estabelecer relações semânticas entre conceitos através de redes conceituais. Apesar da linha tênue entre a definição desses dois conceitos, sua aplicação na prática é muito diferente.

Cavalcante e Bräscher (2014, p.193), conceituam taxonomia como “ferramentas que se prestam à classificação a partir de uma estrutura hierarquizada”. Segundo os autores, apesar do conceito ser muito antigo, apenas a partir de 1990 as taxonomias passaram a ser utilizadas como elementos estruturantes da informação em meio digital, sobretudo na *Web*. Isso ocorreu na tentativa de sanar dificuldades dos motores de busca ao lidar com grandes bases de dados, dificuldade dos usuários para encontrar as informações desejadas, e a necessidade por parte das empresas para organizar e integrar suas informações. Portanto, conforme afirmam Steimer e Luz (2015, p. 8), “[...] em ambiente *web*, as taxonomias facilitam o acesso e navegação, contribuem para a *findability* e para recuperação inteligente”.

Apesar da utilização de ontologias para organização e representação do conhecimento em sistemas de *e-commerce* não ser tão comum quanto taxonomias, essa abordagem é possível.

Enquanto a taxonomia remete a estruturas hierárquicas com relação de atributos, as ontologias remetem à conceitualização dos termos utilizados. Segundo Almeida e Bax (2015), as ontologias possibilitam a compreensão comum e compartilhada de todo um domínio de conhecimento. Isso é possível através da criação de uma estrutura semântica das fontes de dados. Entretanto, o uso de ontologias em sistemas de *e-commerce* não é comum. De acordo com Aanen, Vandic e Frasincar (2015), descartadas algumas exceções, a informação na *web*, especialmente em ambientes de produtos, não é semanticamente anotada. Por essa razão, a taxonomia é mais aplicável que a ontologia nessa área.

2.3.2. Combinação de taxonomias em integrações de *marketplace*

A variedade de produtos ofertada através de sistemas de *e-commerce* é muito

ampla. As redes varejistas Magazine Luiza[11] e Lojas Americanas[12], por exemplo, possuem respectivamente mais de 44 e 60 mil itens em seus catálogos (considerando lojas físicas e *e-commerce*). Contudo, como Vital e Café (2011) e Steimer e Luz (2015) afirmam, apesar de existirem diversos modelos propostos para padronização de catálogos eletrônicos, cada *e-commerce* possui uma taxonomia única, diferenciando-se a partir de uma série de fatores, como critérios internos de priorização, sazonalidade, identidade da marca e de seu público alvo.

Levando em conta o constante aumento das vendas *online*, espera-se que haja também um crescimento significativo dos catálogos eletrônicos, ainda mais considerando existência de *marketplaces* eletrônicos; conforme Steimer e Luz (2015, p. 5), "[...] muitas empresas não estão se limitando apenas aos próprios produtos, mas ao invés disso se tornando verdadeiros *shoppings* virtuais, que são chamados também de *Marketplaces*".

Com a grande quantidade de produtos sendo integrados para o sistema de *e-commerce* da empresa detentora da plataforma de *marketplace*, surge a necessidade de organizar os produtos de modo que faça sentido para os clientes. Segundo Steimer e Luz (2015), independente do meio de acesso, seja *desktop* ou *mobile*, a usabilidade, a arquitetura e a organização de um *e-commerce* em varejo precisa ser coesa e alinhada. Portanto, é fundamental que a taxonomia do *e-commerce* correspondente ao *marketplace* seja respeitada no momento das integrações. Contudo, agregar informações de diferentes catálogos de produtos em um modelo único, como nas integrações de produtos em plataformas de *marketplace*, é uma atividade complexa. Conforme afirmam Almeida e Bax (2015), "[...] com um número crescente de fontes de dados disponíveis, torna-se cada vez mais difícil a seleção, aquisição e combinação de dados".

Devido à complexidade, plataformas ou vendedores podem optar pela contratação de empresas especializadas em integrações de *marketplace*. Atualmente, existem diversas dessas empresas no mercado[13], porém, as técnicas ou ferramentas específicas que são utilizadas nos processos de integração não são divulgadas publicamente. Já na literatura, poucos estudos foram realizados com foco na

[11] <https://www.magazineluiza.com.br/quem-somos/perfil-da-empresa/>. Acesso em 14/06/2019.
[12] <https://ri.lasa.com.br/a-empresa/perfil>. Acesso em 14/06/2019.
[13]https://plugg.to/marketplace-de-integracao/?gclid=EAIaIQobChMI8Mq62-np4gIViF3Ch0yEgXjEAAYAiAAEgJD1vD_BwE>; <https://anymarket.com.br/>; <https://skyhub.com.br/home/>; <https://www.hub2b.com.br/>. Acessos em 14/06/2019.

integração e combinação de catálogos eletrônicos em plataformas de *marketplace*, especialmente no Brasil. Entretanto, analisando a documentação das plataformas, como o *marketplace* da Lojas Colombo[14], é possível identificar quais dados são necessários nas integrações e, por conseguinte, quais técnicas, métodos ou ferramentas poderiam auxiliar esse processo. Através da documentação da Lojas Colombo, também é possível observar que a empresa se preocupa com a organização dos produtos em sua taxonomia, exigindo que o vendedor informe em qual categoria específica do *e-commerce* o produto enviado deve ser enquadrado. Isso pode ser observado em um dos campos obrigatórios para envio do produto, chamado "*groupId".*

2.3.3. Métodos e ferramentas para combinação de taxonomias

Para que um vendedor inicie sua operação em um *marketplace*, é necessário que seu catálogo de produtos seja integrado na plataforma[15]. Contudo, como é possível observar nas documentações das APIs, o vendedor deve respeitar a hierarquia existente no *e-commerce* da empresa detentora da plataforma. Isso é feito no momento da integração, quando o vendedor indica em qual categoria da taxonomia determinado produto será inserido. Portanto, antes de tudo, é necessário que as taxonomias ou ontologias dos vendedores e plataforma sejam mapeadas e combinadas em um modelo único.

Em um estudo realizado por Mehrbod et al. (2015), um modelo chamado *vector space model* (VSM), geralmente utilizado como base em técnicas para determinação de similaridade em mecanismo de buscas, foi aprimorado para tratar o problema de integração de catálogos eletrônicos. Segundo Salazar (2012), VSM é um modelo algébrico que busca criar uma representação que descreva a informação contida em documentos textuais de forma estruturada através de vetores, com a mínima perda de informação possível. Essa representação leva em consideração a frequência de termos considerados relevantes.

No mecanismo desenvolvido, cada um dos catálogos é interpretado individualmente para que, então, seja apontada sua similaridade para dado catálogo de origem. A interpretação sintática é dada através da taxonomia do catálogo, e a

[14]<https://api.marketplace.colombo.com.br/swagger-ui.html#/Product/saveUsingPOST>. Acesso em 15/06/2019.
[15] <https://www.colombo.com.br/marketplace#section4>. Acesso em 15/06/2019.

análise semântica, através da ontologia. O principal diferencial da abordagem proposta pelos autores é a utilização da interpretação semântica dos termos no processo.

O aprimoramento feito no modelo VSM levou em conta outras modificações já desenvolvidas em um estudo anterior realizado por Mehrbod, Zutshi e Grilo (2014). O diferencial do aprimoramento está relacionado à semântica que, desta vez, também seria levada em consideração. Primeiramente, a abordagem utilizada buscou realizar a expansão dos termos através de bancos de dados léxicos, como o *WordNet*[16], e também através da utilização de ontologias existentes. No processo de expansão utilizando ontologias, os vetores criados pelo VSM foram enriquecidos com conceitos semânticos provenientes dos catálogos utilizados. Por fim, relações semânticas foram adicionadas nos termos dos vetores, possibilitando a busca de catálogos eletrônicos semanticamente similares.

Recentemente, Mehrbod et al. (2018) publicaram outro estudo relativo ao mecanismo de combinação de catálogos eletrônicos já desenvolvido. Com o objetivo de encontrar oportunidades de negócio adequadas, os autores utilizaram o mecanismo para medir a similaridade entre propostas públicas de licitações - disponibilizadas em plataformas de *marketplace* de licitações - e catálogos eletrônicos de produtos ou serviços oferecidos por fornecedores.

O processo utilizado pelo mecanismo desenvolvido inicia-se com a seleção das entidades existentes no catálogo. Depois, uma ontologia considerada relevante é escolhida através de um sistema de classificação, tendo como base as entidades selecionadas. Se nenhuma ontologia adequada for encontrada no repositório, o algoritmo utilizará todas as ontologias disponíveis e tentará reconhecer todas as entidades do catálogo eletrônico. Por fim, o processo recebe uma entidade e uma ontologia como entradas e retorna recursivamente quaisquer outras entidades existententes nessa ontologia que estejam relacionadas com a entidade informada como entrada.

O estudo faz uso de duas ontologias geradas a partir de um sistema de classificação. Essa geração é feita através do método proposto por Stolz *et al*. (2014), que busca extrair ontologias a partir de sistemas de classificação de produtos. As extrações foram feitas em dois desses sistemas: *Common Procurement Vocabulary*

[16] <https://wordnet.princeton.edu/>. Acesso em 15/06/2019.

(CPV) e *the United Nations Standard Products and Services Code* (UNSPSC).

Os estudos de Mehrbod *et al*. (2015) e Mehrbod *et al*. (2018) não preveem a criação de um modelo único de catálogo, de taxonomia ou de ontologia. Além disso, os métodos utilizados visam a análise semântica dos dados. Porém, conforme já citado, a grande maioria dos sites de *e-commerce* não anotam semanticamente seus dados. Portanto, independente dos resultados, essas abordagens não podem ser utilizadas de forma prática no processo de integração de catálogos em plataformas de *marketplace*.

No que tange a combinação de taxonomias, a metodologia mais promissora encontrada durante a pesquisa realizada no presente trabalho foi proposta por Aanen, Vandic e Frasincar (2015). Nesse estudo, os autores desenvolveram dois algoritmos para realização de combinação de taxonomias em um modelo único. Além disso, avaliaram o desempenho de outros algoritmos de combinação já existentes: algoritmo de Park e Kim (2007) e AnchorPROMPT.

Anchor-PROMPT é um algoritmo que encontra termos semanticamente similares automaticamente. O algoritmo utiliza, como entradas, um conjunto de âncoras (*anchors*). Âncoras são pares de termos relacionados definidos manualmente pelo usuário ou de forma automática, através da paridade lexical. O Anchor-PROMPT analisa a frequência de ocorrência de termos em posições similares. Tais termos, provalmente, estarão representando conceitos semanticamente similares (NOY; MUSEN, 2001; AANEN; VANDIC; FRASINCAR, 2015).

O algoritmo de Park e Kim (2007) foi desenvolvido especificamente para combinação de taxonomias de produtos. O processo realizado pelo algoritmo tem, como foco inicial, a desambiguação do significado do nome de uma categoria a partir da taxonomia de origem. Esse algoritmo pode ser facilmente aplicado no que diz respeito a *e-commerce* e, de acordo com Aanen, Vandic e Frasincar (2015), experimentos já realizados mostram que esse algoritmo apresenta melhores resultados do que a maioria dos outros algoritmos existentes.

Os algoritmos desenvolvidos pelos autores utilizam como base o algoritmo de Park e Kim (2007). Além disso, cada um dos dois algoritmos utilizou um procedimento de desambiguação de sentidos de palavras diferente. O primeiro fez uso do modelo proposto por Park e Kim (2007), e o outro utilizou o processo proposto por Lesk (1986). Essas versões do algoritmo foram nomeadas pelos autores como Aanen-Park e Aanen-Lesk, e serão assim representados nas imagens.

Word Sense Disambiguation (WSD), ou Desambiguação de Sentidos de Palavras, é uma tecnologia desenvolvida com o objetivo de encontrar automaticamente o sentido de uma palavra em determinado contexto. O processo de desambiguação de sentidos é muito útil para recuperação de informações, extração de informações, pesquisa na *web* e indexação. Desde seu surgimento, tem sido foco de diversas pesquisas, principalmente na área de processamento de linguagem natural (AGIRRE; LACALLE; SOROA, 2014; LOPEZ-AREVALO et al., 2017).

Para avaliação do algoritmo proposto pelos autores, foram utilizados três *datasets.* O primeiro é a taxonomia de produtos da *amazon.com* que, atualmente, é um dos maiores *e-commerce* dos Estados Unidos e do mundo. A segunda taxonomia utilizada foi do *Overstock.com*, uma grande varejista dos Estados Unidos que disponibiliza mais de um milhão de produtos em sua loja virtual, contando com mais de mil categorias. Uma diferença da taxonomia da *Overstock* para as demais utilizadas é que as páginas de produtos do *e-commerce* são semanticamente anotadas de acordo com a ontologia *GoodRelations*[17]. O terceiro *dataset* é conhecido como *Open Directory Project* (ODP). Não se trata de uma loja *on-line*, mas sim de um projeto que visa a categorização completa da *web* em estruturas hierárquicas.

Com base nas três taxonomias de produtos utilizadas pelos autores, seis mapeamentos origem-destino puderam ser criados. Umas das principais dificuldades dos autores foi determinar quais mapeamentos de categoria origem para categoria destino estariam corretos, levando em conta a intepretação de um ser humano. Para isso, os autores criaram, manualmente, mapeamentos antes da execução dos algoritmos. Esses mapeamentos foram, então, comparados com os resultados obtidos.

Tanto o algoritmo de Park e Kim quanto o algoritmo proposto pelos autores, utilizam um parâmetro chamado limiar de similaridade. Esse limiar funciona como um limite para filtrar mapeamentos baseados em uma similaridade que são insuficientes. Isto é, quando a semelhança entre a categoria de origem e a categoria destino está abaixo do limiar de similaridade, a categoria não será mapeada pelo algoritmo. Para avaliação dos resultados, foram analisadas a precisão, acurácia, especificidade, *Recall* e *F1-measure* dos algoritmos. O Recall representa a taxa de classes mapeadas referente as classes que deveriam ter sido mapeadas pelo processo. Em outras

[17] <http://www.heppnetz.de/projects/goodrelations/>. Acesso em 16/06/2019.

palavras, o Recall é o número de resultados corretos dividido pelo número de resultados que deveriam ter sido obtidos. Já o *F1-measure* representa a medida de performance geral utilizada pelos autores. Os resultados foram separados por *dataset* e por limiar de similaridade.

A Tabela 1 provê uma visão geral das medidas mais importantes utilizadas para a avaliação:

Tabela 1 – Visão geral dos resultados médios por algoritmo

Algorithm	Precision	Recall	F1-measure	Avg. time	Senses found	WSD accuracy
PROMPT	19.82%	10.62%	0.1350	0.47 s	n/a	n/a
Park & Kim	37.89%	17.93%	0.2415	4.52 s	0.057	83.72%
Aanen–Park	35.84%	35.21%	0.3459	7.97 s	0.048	91.30%
Aanen–Lesk	35.75%	45.87%	0.3909	80.94 s	0.411	90.72%

Fonte: (AANEN; VANDIC; FRASINCAR, 2015, p.25).

2.4. CONSIDERAÇÕES FINAIS

A evolução constante da tecnologia possibilita o desenvolvimento, também, dos modelos de negócios existentes nas organizações. Uma dessas evoluções, o *marketplace* eletrônico, traz consigo diversos desafios para as partes envolvidas. Um desses desafios consiste na combinação de diferentes taxonomias de catálogos eletrônicos em um modelo único, processo necessário durante a integração de produtos entre vendedores e plataforma de *marketplace*.

Apesar de existirem ferramentas e algoritmos para esse fim, poucas se propõem a estabelecer um modelo único de taxonomia. Dentre as possibilidades verificadas, os algoritmos desenvolvidos por Aanen, Vandic e Frasincar (2015) poderiam ser uma possível alternativa, pois visam a criação de um modelo unificado de forma automática. Contudo, como mostra a Tabela 1, a precisão e o *Recall* não se mostram satisfatórios, uma vez que não chegam a 50%. Faz-se, então, necessário o desenvolvimento de outra abordagem, conforme será apresentada neste trabalho.

3. INTEGRAÇÃO DE PRODUTOS EM MAKETPLACES VIA API

A integração entre o catálogo de produtos dos vendedores e o catálogo já existente nas plataformas de *marketplace* é uma necessidade básica. Portanto, é fundamental que os responsáveis pela plataforma disponibilizem formas simples e eficientes de realizar esse processo.

Em conformidade com o que afirma Choudary (2015), as plataformas de *marketplace* existentes hoje no mercado disponibilizam APIs para que as integrações sejam realizadas. Alguns exemplos são as plataformas de *marketplace* da Lojas Colombo[18] e B2W[19], as quais podem ser observadas nas Figuras 2 e 3.

Figura 2 – Modelo Json Lojas Colombo


```
{
 "brand": "string",
 "description": "string",
 "groupId": 0,
 "guide": "string",
 "model": "string",
 "name": "string",
 "productAttributeValues": [
  {
   "attributeId": 0,
   "attributeValue": "string"
  }
 ],
 "productItemVariations": [
  {
   "color": "string",
   "deliveryTimeDays": 0,
   "ean": "string",
   "height": 0,
   "length": 0,
   "price": 0,
   "productItemPictures": [
    {
     "picture": "string"
    }
   ],
   "size": "string",
   "skuSellerId": "string",
   "stock": 0,
   "voltage": "string",
   "weight": 0,
   "width": 0
  }
 ],
 "youtubeLink": "string"
}
```


Fonte: Documentação API Marketplace Lojas Colombo – disponível em: <https://api.marketplace.colombo.com.br/swagger-ui.html>. Acesso em 31/05/2019.

[18]Documentação API Marketplace Lojas Colombo – disponível em: <https://api.marketplace.colombo.com.br/swagger-ui.html>. Acesso em 31/05/2019.
[19]Documentação API Marketplace B2W – disponível em <http://apisandbox.bonmarketplace.com.br/explorer/>.Acesso em 31/05/2019.

Figura 3 – Modelo Json B2W

Product {
id (string): (*size*:50) Identificador do produto,
name (string): (*size*:100) Nome do produto,
sku (array[Sku]): Lista contendo os SKUs do produto,
manufacturer (Manufacturer, *optional*): Informações do fabricante,
deliveryType (string): Tipo de entrega do produto. Este campo recebe os seguintes valores: SHIPMENT, WITHDRAW e ALL
- SHIPMENT - Entrega do produto pelo revendedor
- WITHDRAW - Retirada local do produto em lugar estabelecido pelo revendedor
- e ALL - Efetua as duas modalidades

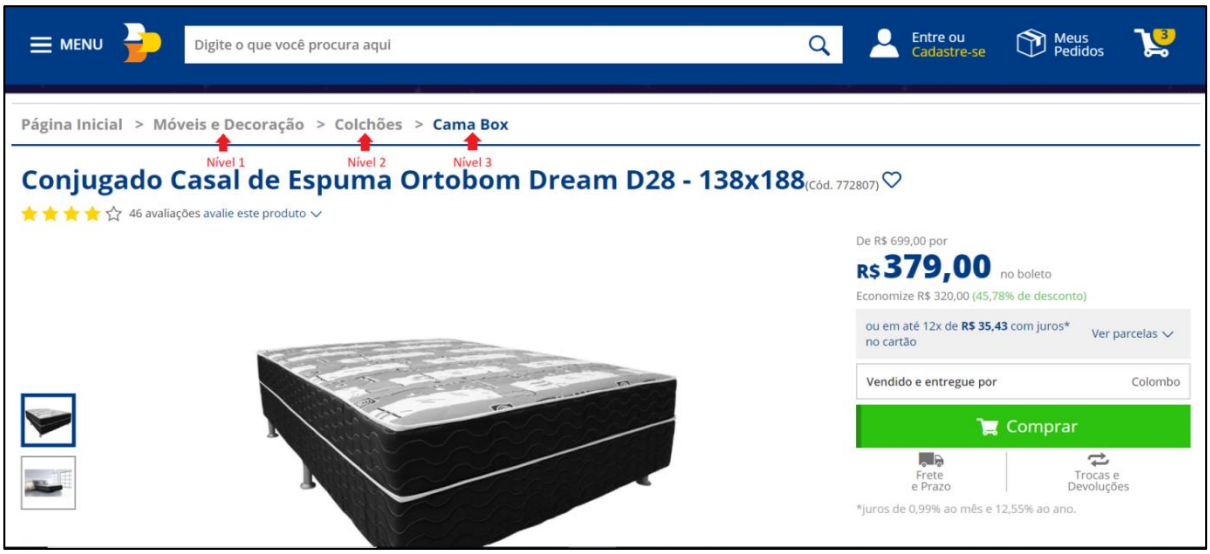
,
nbm (Nbm, *optional*): Nomenclatura Brasileira de Mercadoria
}
Sku {
id (string): (*size*:18) Id do SKU que é fornecido pelo parceiro,
name (string): (*size*:100) Nome/Descrição do SKU a ser importado, ex.: sku + marca + modelo + diferenciador se necessário,
description (string): (*size*:4000) Descrição detalhada do SKU. Podem ser enviadas as seguintes tags: , , ,
</br> ou Não é permitido enviar: Tags HTML exceto as mencionadas acima, caracteres especiais e vídeos. Erros ortográficos também serão reprovados pela auditoria.,
ean (array[string], *optional*): Lista de EANs do SKU. Pode possuir de 1 a N elementos. O campo EAN passa a ser opcional se o parceiro estiver liberado pela equipe comercial a não enviar o mesmo.,
height (number, *optional*): (*size*:15.4) Altura do SKU (metros),
width (number, *optional*): (*size*:15.4) Largura do SKU (metros),
length (number, *optional*): (*size*:15.4) Comprimento do SKU (metros),
weight (number): (*size*:15.4) Peso do SKU (kg),
stockQuantity (integer): Quantidade em estoque do SKU,
enable (boolean): Indicador do status do SKU. (True/False),
price (Price): Informações sobre preço. Obs: Será ignorado no caso que a lista 'brandPrices' seja preenchida,
brandPrices (array[BrandPrice], *optional*): Lista de preços por marca. O campo BRAND PRICE é opcional, mas se informado sobrepõe os dados do campo PRICE.,
urlImage (array[string]): (*size*:125) Lista de URLs onde podem ser encontradas as imagens do produto. No mímimo 1 imagem, no máximo 10 imagens. Resolução mínima 350px x 350px. URL deve ser válida no protocolo http,
marketStructure (MarketStructure, *optional*): Posição da estrutura mercadologica selecionada para o SKU. Se pelo menos um nível da estrutura é enviado, todos os níveis restantes devem ser enviados,
attributeValues(array[AttributeValue], *optional*): Atributos do SKU conforme Estrutura Mercadológica selecionada.,
crossDocking (integer, *optional*): (*size*:3) Prazo cross-docking do produto (em dias),
instructions (string, *optional*): (*size*:4000) Utilize esse campo para informar as instruções de uso do seu produto não físico. Esse campo é recomendado caso você esteja comercializando um produto virtual
}
Price {
listPrice (number): (*size*:15.2) "Preço de" do SKU,
sellPrice (number): (*size*:15.2) "Preço por" do SKU
}
BrandPrice {
store (string): (*size*: 15) Identificador da Loja (SHOPTIME, LOJASAMERICANAS, SUBMARINO, SOUBARATO, B2WEMPRESAS, AMERICANASAPP, AMERICANASQUIOSQUE, SUBMARINOAPP, SUBMARINOQUIOSQUE, SHOPTIMEAPP, SHOPTIMEQUIOSQUE),
listPrice (number): (*size*:15.2) "Preço de" do SKU,
sellPrice (number): (*size*:15.2) "Preço por" do SKU
}
MarketStructure {
categoryId (integer): (*size*:30) ID da Categoria selecionada para o SKU.,
subCategoryId (integer): (*size*:30) ID da Sub-categoria selecionada para o SKU.,
familyId (integer): (*size*:30) ID da Familia selecionada para o SKU.,
subFamilyId (integer): (*size*:30) ID da Sub-familia selecionada para o SKU.
}
AttributeValue {
name (string): (*size*:100) Nome do atributo conforme informado no recurso "attribute",
value (Object): (*size*:100) Valor do atributo
}
Manufacturer {
warrantyTime (integer, *optional*): (*size*:2) Tempo da garantia (em meses) do produto no fabricante,
model (string, *optional*): (*size*:30) Modelo do produto de acordo com o fabricante,
name (string, *optional*): (*size*:14) Nome do fabricante
}
Nbm {
origin (integer, *optional*): (*size*:2) Procedência: 0 Nacional, 1 Importação Direta ou 2 Estrangeira - Adquirido Mercado Int.,
number (integer, *optional*): (*size*:10) Número referente ao NBM (Nomenclatura Brasileira de Mercadoria)
}

Fonte: Documentação API Marketplace B2W – disponível em <http://apisandbox.bonmarketplace.com.br/explorer/>.Acesso em 31/05/2019.

Em ambas as APIs, é possível observar um conjunto de métodos de acesso a API relacionados ao cadastro ou integração de novos produtos[20]. No ponto de acesso específico para a inserção de um novo produto no catálogo do *marketplace*, são disponibilizados os modelos de Json que devem ser enviados pelo vendedor.

Analisando os modelos de arquivo disponibilizados pelos responsáveis nas plataformas de *marketplace*, nota-se a necessidade de informar a qual categoria do *marketplace* o produto se refere. Na API da Lojas Colombo, por exemplo, o campo que se refere a categoria é representado por "*groupid*". O código informado nesse campo deve ser correspondente ao nível mais específico da taxonomia do *e-commerce* da Lojas Colombo. Nos produtos disponíveis no *e-commmerce* da empresa, é possivel observar a existência de três níveis (Figura 4).

Figura 4 – Níveis da taxonomia Lojas Colombo

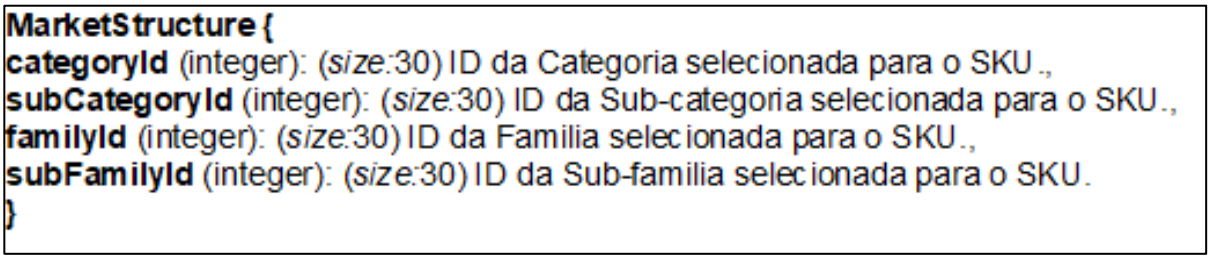


Fonte: Disponível em: <https://www.colombo.com.br/produto/Moveis-e-Decoracao/Conjugado-Casal-de-Espuma-Ortobom-Dream-D28-138x188>. Acesso em 01/06/2019.

Já na API da plataforma pertencente à empresa B2W, é possível identificar a existência de quatro níveis de hierarquia:

[20]Endpoint para cadastro de produtos, API Lojas Colombo – disponível em: <https://api.marketplace.colombo.com.br/swagger-ui.html#/Product/saveUsingPOST>. Acesso em 01/06/2019.
Endpoint para cadastro de produtos, API B2W – disponível em: <http://api-sandbox.bonmarketplace.com.br/explorer/#!/product/createProduct_post_0>. Acesso em 01/06/2019.

Figura 5 – Níveis de hierarquia da plataforma da empresa B2W



Fonte: Documentação API Marketplace B2W – disponível em <http://apisandbox.bonmarketplace.com.br/explorer/>.Acesso em 31/05/2019.

A categoria informada nesses campos deve ser análoga à estrutura hierárquica do *e-commerce* ao qual a plataforma de *marketplace* corresponde. Portanto, os vendedores ou a plataforma devem realizar um processo de combinação entre as categorias do vendedor e a taxonomia do *marketplace* eletrônico.

No estudo realizado por Aanen, Vandic e Frasincar (2015), foram analisados algoritmos de combinação de taxonomias que poderiam ser utilizados de forma integral ou parcial nas integrações de *marketplace*, tornando o processo automatizado. Contudo, como é possível observar na Tabela 1, mesmo com as melhorias propostas por Aanen, Vandic e Frasincar (2015), a precisão e o *Recall* desses métodos não chegam a 50%. Esse resultado não é satisfatório para utilização comercial e em grande escala.

4. PROPOSTA DE SOLUÇÃO

A proposta de solução deste estudo baseia-se no desenvolvimento de um protótipo que objetiva facilitar o processo de integração de produtos em uma plataforma de *marketplace* através da combinação das taxonomias envolvidas. Primeiramente, será detalhado o cenário de integração com base nas documentações abertas de duas plataformas de *marketplace* do mercado. Após, a modelagem da solução será apresentada, seguida das considerações finais.

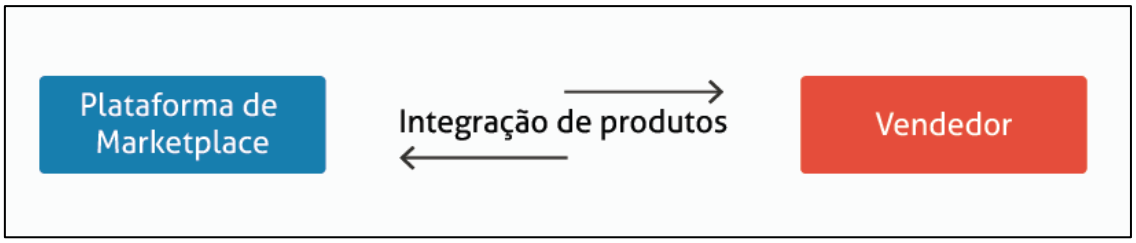
4.1. PROTÓTIPO PARA COMBINAÇÃO DE TAXONOMIAS DE DIFERENTES CATÁLOGOS ELETRÔNICOS EM UMA PLATAFORMA DE *MARKETPLACE*

A solução proposta neste trabalho tem como objetivo o desenvolvimento de um protótipo que possibilite a combinação de categorias e atributos que constituem a taxonomia de diferentes catálogos de vendedores com a taxonomia de uma plataforma de *marketplace*. Através do sistema, os vendedores poderão visualizar as categorias e atributos existentes na plataforma de *marketplace* e, então, associar com as categorias e atributos dos seus próprios catálogos.

A combinação manual de categorias e atributos precisará ser realizada uma única vez. Após a conclusão da combinação, uma API será disponibilizada para consulta da associação realizada. Dessa forma, independente de quem for o responsável pela integração, seja o vendedor, intermediários ou plataforma de *marketplace*, poderá consultar a combinação sempre que necessário durante o processo de integração de produtos. Isso garante uma assertividade maior nas integrações, maior agilidade e isenção de qualquer erro por parte da plataforma, uma vez que a responsabilidade da categorização recairá sobre o vendedor, que é o maior interessado nesse processo.

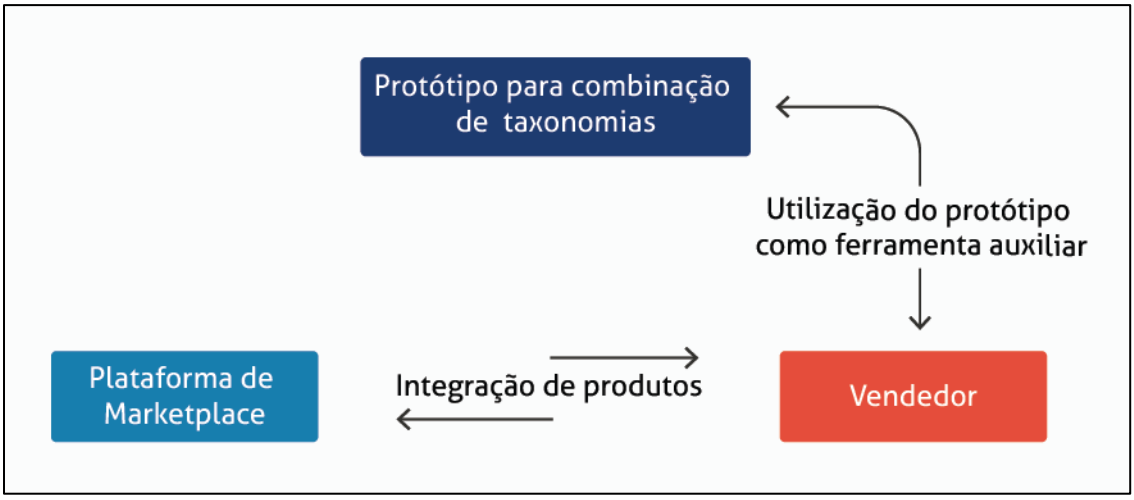
Nas Figuras 6 e 7 pode-se observar a integração do verdedor de forma direta com a plataforma, sendo que na Figura 7 o vendedor conta com o protótipo como ferramenta auxiliar. Já as Figuras 8 e 9 representam as integrações realizadas através de um agente intermediário. A Figura 9 exemplifica a utilização do protótipo na integração através do intermediário.

Figura 6 – Integração de produtos realizada pelo vendedor.



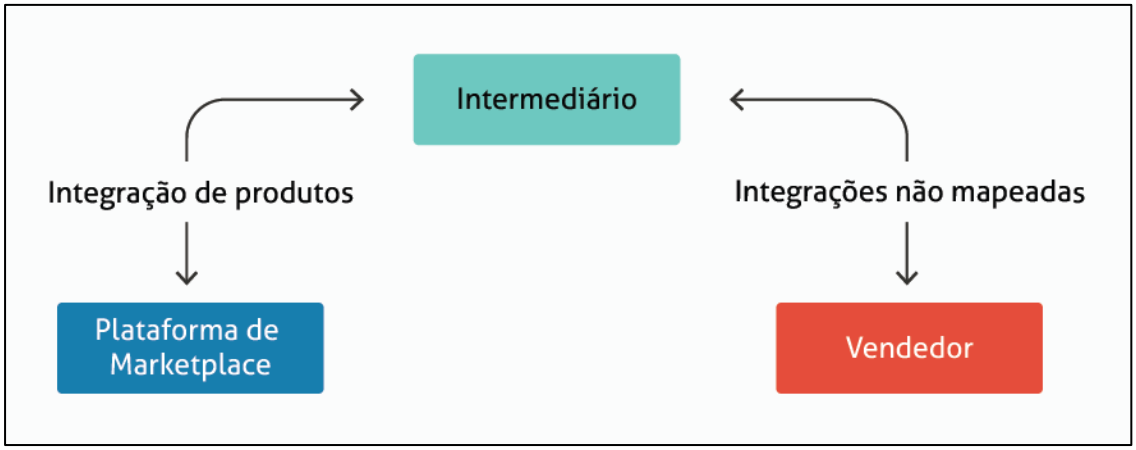
Fonte: elaborado pelo autor (2019).

Figura 7 – Integração de produtos realizada pelo vendedor com o auxílio do protótipo para combinação de taxonomias.



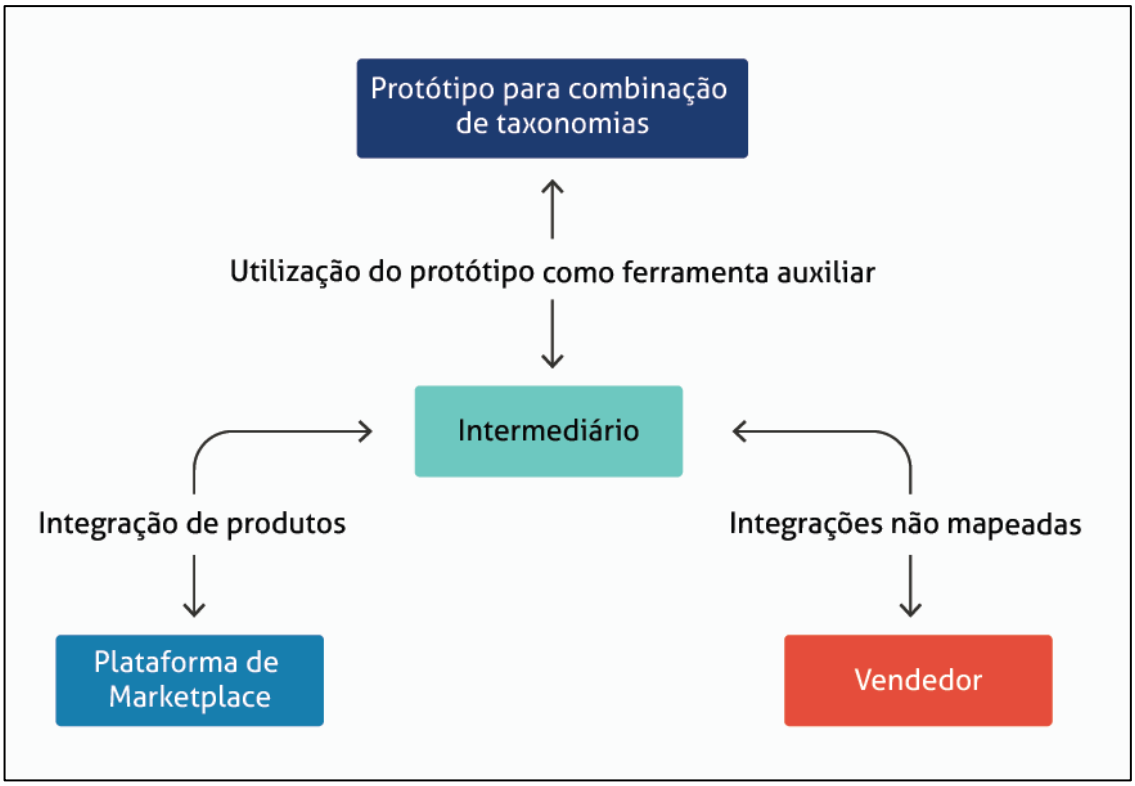
Fonte: elaborado pelo autor (2019).

Figura 8 – Integração de produtos realizada por intermediário.



Fonte: elaborado pelo autor (2019).

Figura 9 – Integração de produtos realizada por intermediário com o auxílio do protótipo para combinação de taxonomias.



Fonte: elaborado pelo autor (2019).

4.1.1. Requisitos do sistema

A fim de atender as necessidades dos clientes, o processo de engenharia de requisitos visa descobrir, analisar e documentar os serviços e restrições de um sistema. Sommerville (2011) define requisitos e o processo de engenharia de requisitos:

> Os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que oferece e as restrições a seu funcionamento. Esses requisitos refletem as necessidades dos clientes para um sistema que serve a uma finalidade determinada, como controlar um dispositivo, colocar um pedido ou encontrar informações. O processo de descobrir, analisar, documentar e verificar esses serviços e restrições é chamado engenharia de requisitos (RE, do inglês requirements engineering). (SOMMERVILLE, 2011, pag. 57)

Segundo Sommerville (2011), os requisitos de *software* podem ser classificados como funcionais e não funcionais. Os requisitos funcionais correspondem a serviços que o sistema deve fornecer e comportamentos esperados; além disso, podem especificar o que o sistema não deve fazer. Já os requisitos não

funcionais representam restrições ou funções que o sistema possui. De maneira oposta ao que ocorre nas características ou serviços, os requisitos não funcionais podem se aplicar ao sistema como um todo - exemplos desse tipo de requisito são: tempo limite para processamento, responsividade, compatibilidade, entre outros.

Os requisitos também são fundamentais para garantir que os serviços e características do sistema desenvolvido sejam entregues corretamente, conforme afirma Sommerville (2011):

> [...] os requisitos não são independentes e que muitas vezes geram ou restringem outros requisitos. Portanto, os requisitos de sistema não apenas especificam os serviços ou as características necessárias ao sistema, mas também a funcionalidade necessária para garantir que esses serviços/características sejam entregues corretamente. (pag. 59).

Portanto, seguem, abaixo relacionados, nos Quadros 1 e 2, os requisitos do protótipo para combinação de taxonomias:

Quadro 1 – Requisitos Funcionais

CÓDIGO	**DESCRIÇÃO**
RF001	Cadastro e atualização dos dados de plataformas de *marketplace*.
RF002	Cadastro e atualização dos dados de vendedores.
RF004	Consumo da API da plataforma de *marketplace* para persistência dos dados da hierarquia de categorias.
RF005	Consumo da API do vendedor para persistência dos dados da hierarquia de categorias.
RF006	Consumo da API da plataforma de *marketplace* para persistência dos dados da hierarquia de atributos relacionados às categorias existentes.
RF007	Consumo da API do vendedor para persistência dos dados da hierarquia de atributos relacionados às categorias existentes.
RF008	Cadastro e atualização dos dados de uma combinação de taxonomias.
RF009	Busca e visualização das combinações já realizadas.
RF010	Disponibilização de uma API REST para consulta das combinações já realizadas.
RF011	Navegação entre as funcionalidades do sistema de forma gráfica.

Fonte: elaborado pelo autor (2019).

Quadro 2 – Requisitos não funcionais

(continua)

CÓDIGO	**DESCRIÇÃO**
RNF001	Manutenibilidade - desenvolver o sistema de forma organizada, respeitando a arquitetura definida e promovendo, assim, a manutenibilidade.
RNF002	Disponibilidade – a API para consulta das combinações deve estar disponível de forma online, para que qualquer outro sistema possa consumi-la.

(conclusão)

RNF003	Portabilidade - o sistema deverá permitir o acesso de qualquer dispositivo que tenha acesso à internet.

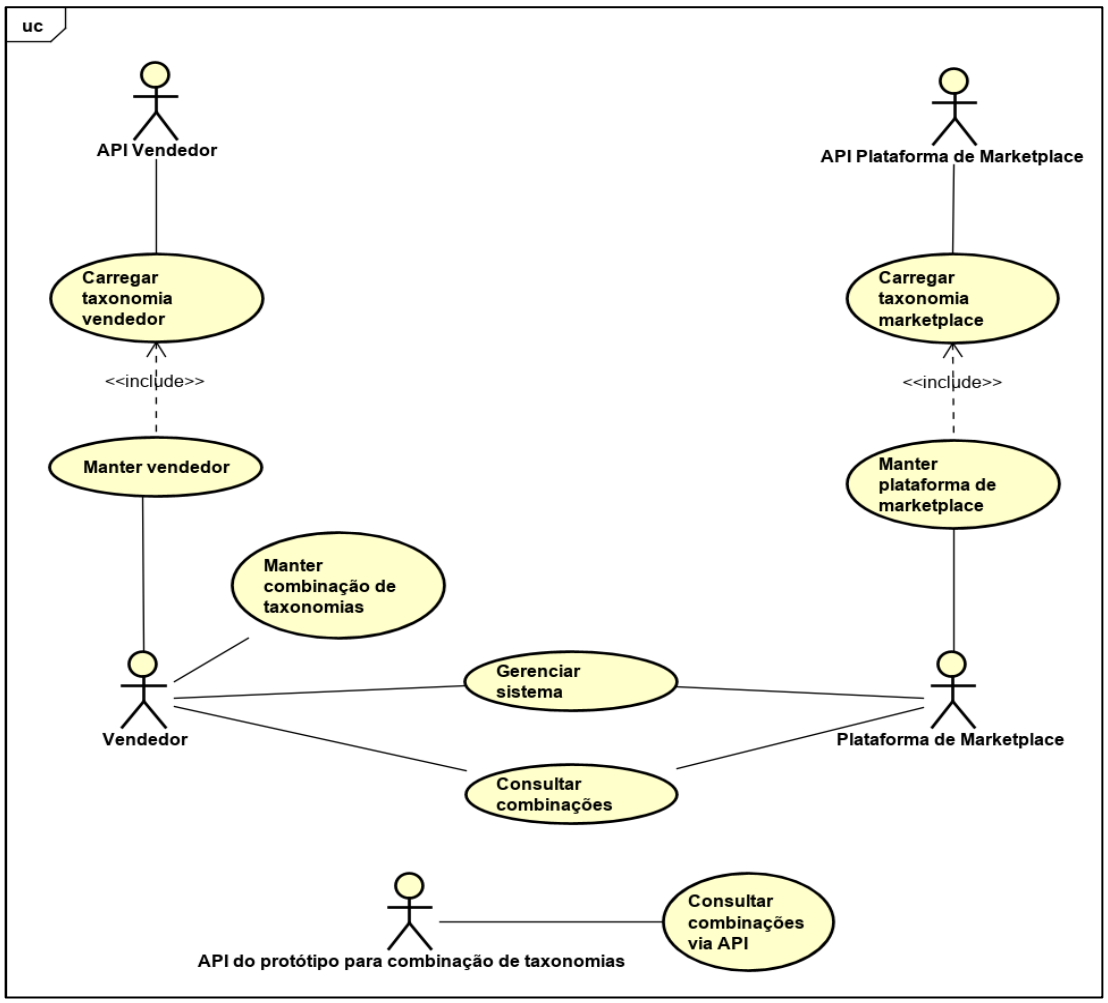
Fonte: Elaborado pelo autor (2019).

4.1.2. Casos de uso do sistema

Conforme afirma Larman (2007), o modelo de casos de uso representa um conjunto de cenários típicos de uso de um sistema. Geralmente, os casos de uso correspondem aos requisitos funcionais existentes e, conforme explica o autor, influenciam diversos aspectos do projeto, pois enfatizam os objetivos e perspectivas do usuário. Pressman e Maxim (2016) complementam, afirmando que, conforme os casos de uso são definidos, uma visão geral das funções e características do sistema começa a se materializar.

Segundo Pressman e Maxim (2016), casos de uso podem ser representados de diversas formas: texto narrativo, descrição geral das tarefas ou interações, descrição baseada em modelos ou uma representação esquemática. Sobre a representação esquemática, Larman (2007) afirma que casos de uso podem ser representados através de um diagrama de casos de uso UML (*Unified Modeling Language)* (Figura 10), relacionando os nomes dos casos de uso e os atores. Esse tipo de diagrama retrata, de forma simples e clara, o contexto do sistema e seu ambiente.

Figura 10 – Diagrama de casos de uso



Fonte: elaborado pelo autor (2019).

Para modelagem do protótipo desenvolvido neste estudo, os casos de uso serão representados através de um diagrama UML e dos quadros dispostos abaixo. Para corroborar com o entendimento, serão apresentadas também algumas representações da interface visual do sistema.

Além dos nomes dos casos de uso, é possível observar cinco atores no diagrama. De acordo com Larman (2007), um ator representa uma entidade com determinado comportamento no sistema, a qual pode ser uma pessoa, um outro sistema de computador ou uma organização. No diagrama criado, os atores Vendedor e Plataforma representam os usuários que irão interagir com o sistema. O Vendedor é um usuário da empresa que deseja integrar-se no *marketplace*, e o ator Plataforma de *Marketplace* é um usuário da empresa detentora da plataforma. Os demais atores representam sistemas computacionais - no caso, APIs.

Para que os usuários tenham acesso às funcionalidades do sistema, serão

criadas interfaces gráficas. No presente estudo, essas interfaces são representadas como imagens e encontram-se associadas aos casos de uso com os quais estão relacionadas.

Com o objetivo de possibiliar o acesso às demais funcionalidades do sistema, foi criado o caso de uso “Gerenciar sistema”, conforme o Quadro 1, no Apêndice A.

O Quadro 2, localizado também Apêndice A, representa o caso de uso “Manter vendedor” que objetiva o cadastro e atualização de vendedores no sistema. Um vendedor deve possuir uma taxonomia atrelada a ele e pode ou não possuir combinações. O caso de uso em questão possui um ator relacionado chamado “Vendedor”. O “Vendedor” representa a empresa ou o funcionário que irá interagir com o sistema no cadastro.

O objetivo do caso de uso “Carregar taxonomia vendedor” é consumir a API do vendedor para persistir no banco de dados a hierarquia de categorias e atributos do vendedor em questão. Ao salvar o cadastro do vendedor, o sistema irá realizar automaticamente o processo descrito no caso de uso, conforme o Quadro 3, Apêndice A. O ator envolvido neste caso de uso é o “API vendedor”.

Os casos de uso "Manter plataforma de *marketplace*" e "Carregar taxonomia *marketplace*" possuem os mesmos objetivos, funcionalidades e fluxos de funcionamento dos casos de uso “Manter vendedor” e “Carregar taxonomia vendedor”. As únicas diferenças estão nos atores envolvidos, requisitos relacionados e nos modelos de tela, conforme Quadros 4 e 5, existentes no Apêndice A. O caso de uso "Manter plataforma de *marketplace*" está ligado ao ator “Plataforma de *marketplace*”, o qual representa a empresa detentora da plataforma de *marketplace* ou um usuário que trabalha nesta empresa. Já o caso de uso “Carregar taxonomia *marketplace*”, assim como no caso de uso “Carregar taxonomia vendedor”, se relaciona a um ator que representa a API consumida no processo, denominado “API Plataforma de *marketplace*”.

O objetivo do caso de uso “Manter combinação de taxonomias” é o cadastro e atualização de combinações de taxonomias de um vendedor e um *marketplace* previamente cadastrados. O ator ligado a este caso de uso é o “Vendedor”. O Quadro 6, Apêndice A, relaciona os principais aspectos do caso de uso.

O objetivo do caso de uso “Consultar combinações” é possibilitar que o usuário busque, visualize e atualize as combinações existentes. Os atores ligados a esse caso

de uso são “Vendedor” e “Plataforma de *marketplace*”. Mais detalhes podem ser analisados no Quadro 7, Apêndice A.

Conforme o Quadro 8, Apêndice A, o último caso de uso, chamado “Consultar combinações via API”, tem como objetivo a disponibilização de uma API REST para consulta das combinações existentes. O ator envolvido nesse caso de uso é apenas o “API do protótipo para combinação de taxonomias”, correspondente à API propriamente dita. O ator “Vendedor” não está ligado ao caso, pois não é necessariamente o vendedor o responsável pela integração. Como já citado em outro capítulo deste estudo, as integrações podem ser responsabilidade de uma empresa terceirizada, vendedor ou outros agentes não mapeados. Por esse motivo, não existem mais atores envolvidos nesse caso de uso.

4.1.3. Arquitetura do sistema

Para a construção do protótipo de combinação de taxonomias, será utilizada a linguagem de programação Java, um *framework* de desenvolvimento e um banco de dados relacional. Além dessas tecnologias, foi optado pelo uso do padrão de arquitetura *Model View Controller* (MVC), juntamente com o padrão *Composite*. Por fim, como parte do sistema proposto, será desenvolvida uma API REST para consulta das combinações criadas.

No desenvolvimento do protótipo, será utilizada a linguagem de programação Java, juntamente com o *framework Spring Boot*. Será optado pelo uso de um *framework* devido aos diversos benefícios que essa tecnologia oferece. De acordo com Sousa et al. (2017), diferente das bibliotecas que disponibilizam apenas conjuntos de funcionalidades, os *frameworks* disponibilizam, também, uma arquitetura e um conjunto de abstrações em alto nível muito úteis no desenvolvimento de sistemas.

O *framework* escolhido, chamado *Sprint Boot*, é baseado no *framework Spring* MVC. Segundo Sousa et al. (2017), o *Spring* MVC é um *framework* para a linguagem de programação Java que implementa o padrão de arquitetura *Model View Controller* (MVC) ou Modelo Visão Controle.

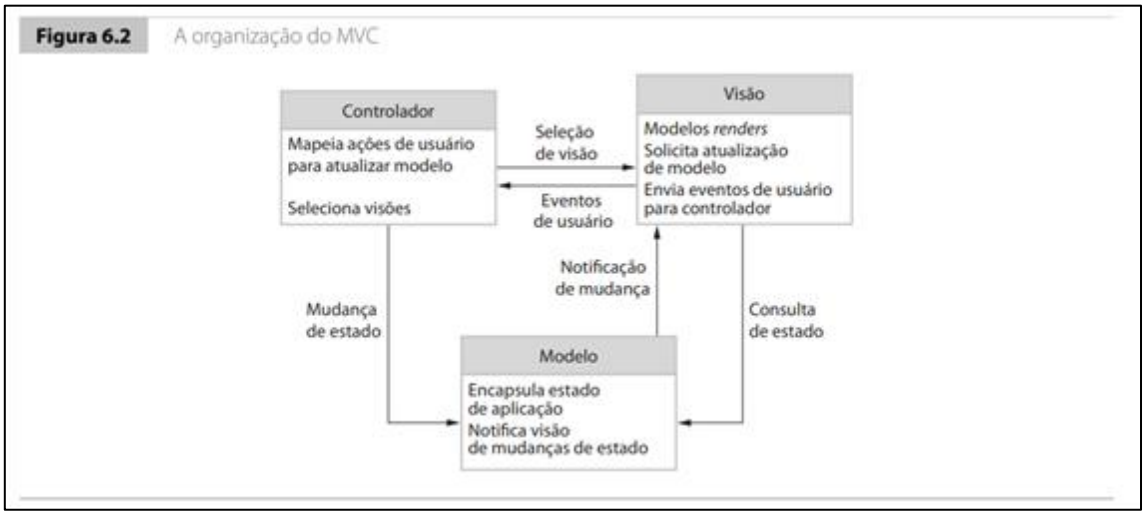
> O framework Spring MVC define uma série de componentes que podem ser estendidos pelas aplicações, esses componentes tem papel fundamental na arquitetura de cada aplicação pois representam conceitos pertencentes a diferentes camadas, como interface com o usuário, tratamento de

> requisições, regras de negócio e persistência de dados. (SOUSA et al., 2017, p. 2).

O *Spring Boot* torna o processo de desenvolvimento mais simples e ágil. Segundo Bortoli e Rufino (2016), o principal objetivo do *Spring Boot* é agilizar e reduzir custos do desenvolvimento de sistemas. Além de todos os benefícios provenientes do *Spring MVC*, esse *framework* ainda possibilita a criação de aplicações *web* sem a necessidade de um servidor específico para execução, uma vez que dispõe de um servidor embarcado. Isso facilita e reduz o tempo de configuração inicial, garantindo maior produtividade (Bortoli e Rufino (2016)**;** Soares (2017)).

Segundo Sommerville (2011), um padrão de arquitetura pode ser entendido como uma descrição abstrata de boas práticas que descreve a organização de um sistema. Esse é o caso do padrão MVC, implementado pelos *frameworks Spring MVC* e *Spring Boot*. Na Figura 11, a organização do padrão MVC pode ser observada.

Figura 11 – Organização do padrão MVC.



Fonte: Sommerville (2011, p.109).

Esse padrão é estruturado em três componentes que interagem entre si. De acordo com Sommerville (2011), o componente *Modelo* gerencia as operações associadas aos dados; o componente *Visão* define como os dados são apresentados aos usuários; por fim, o *Controle* gerencia as interações que ocorrem no sistema, repassando tais interações aos demais componentes.

Conforme afirmam Sommerville (2011) e Soares et al. (2017), o MVC proporciona diversos benefícios. Dentre estes, destacam-se a fácil implementação,

facilidade na reutilização de código, escalabilidade e manutenibilidade. Por esses motivos, esse padrão foi escolhido.

O padrão de projeto MVC implementa, em sua estrutura, um conjunto de outros padrões. Segundo Gamma et al. (2000, p. 22),

> a abordagem MVC usa outros padrões de projeto, tais como Factory Method (112), para especificar por falta (by default) a classe controladora para uma visão e Decorator (170), para acrescentar capacidade de rolagem (scrolling) a uma visão. Mas os principais relacionamentos na MVC são fornecidos pelos padrões Observer, Composite e Strategy.

Paralelamente ao MVC propriamente dito, no desenvolvimento do protótipo será utilizado o padrão *Composite* - Isso porque esse padrão se enquadra perfeitamente nas necessidades arquiteturais do sistema a ser desenvolvido, uma vez que pode ser utilizado para representação de estruturas hierárquicas, como taxonomias. Conforme explica Gamma *et al*. (2000, p. 24), o padrão *Composite* “compõe objetos em estrutura de árvore para representar hierarquias do tipo partes-todo. O *Composite* permite que os clientes tratem objetos individuais e composições de objetos de maneira uniforme.”

A API criada para disponibilizar a consulta de combinação será desenvolvida no modelo REST (*Representational State Transfer*). Segundo Lima (2016, p. 1), "API (*Application Programming Interface*) é uma forma simples de prover serviços em um formato comum e de fácil entendimento." Ainda segundo o autor, uma API pode ser utilizada para as mais diversas funções, como autenticação com uma rede social, integração com plataforma de pagamento e emissão de relatórios gerenciais. Isso é possível graças às séries de métodos e funções disponibilizadas pelas APIs, as quais permitem a utilização das funcionalidades dos sistemas sem a necessidade de conhecer sua implementação. Dessa forma, quaisquer interessados na utilização de uma API deverão apenas se preocupar em consumir os serviços disponibilizados. No caso do protótipo para combinação de taxonomias, APIs REST de plataformas de *marketplace* e vendedores serão consumidas para coletar os dados referentes às estruturas taxonômicas existentes. Além disso, será disponibilizada uma API REST para que os interessados nas integrações entre vendedores e *marketplace* possam consultar as combinações de taxonomias realizadas.

APIs REST têm ganhado cada vez mais destaque entre desenvolvedores. Conforme afirmam Montanheiro, Carvalho e Rodrigues (2017, p. 186), “pensando na

velocidade da transmissão é necessário que se façam otimizações em serviços para que consumam poucos dados, graças a isso, o desenvolvimento de APIs REST está em ênfase na atualidade e seu uso vem crescendo a cada ano.”

Além disso, os autores destacam o fato de que a maioria dos navegadores modernos possuem suporte nativo ao *JavaScript Object Notation* (JSON), o que facilita ainda mais a utilização dessa tecnologia, já que o REST utiliza esse formato para a troca de dados.

Segundo Fielding e Taylor (2002), REST é um conjunto de restrições arquiteturais que visa minimizar a latência e o número de comunicações de rede ao mesmo tempo em que procura maximizar a independência e escalabilidade da implementação de componentes. Isso é feito através das restrições colocadas na semântica dos conectores. Além disso, REST permite o armazenamento em cache, a reutilização de interações, a substituição dinâmica de componentes e o processamento de ações por intermediários. Isso torna o modelo REST ideal para construção de aplicações distribuídas e escaláveis.

4.1.4. Modelo Conceitual

A visão lógica do sistema desenvolvido será apresentada através de um digrama de classes UML, exposto na Figura 21. Segundo Sommerville (2011), diagramas de classes são utilizados para modelagem de sistemas orientados a objetos, com o objetivo mostrar as classes do sistema e suas associações.

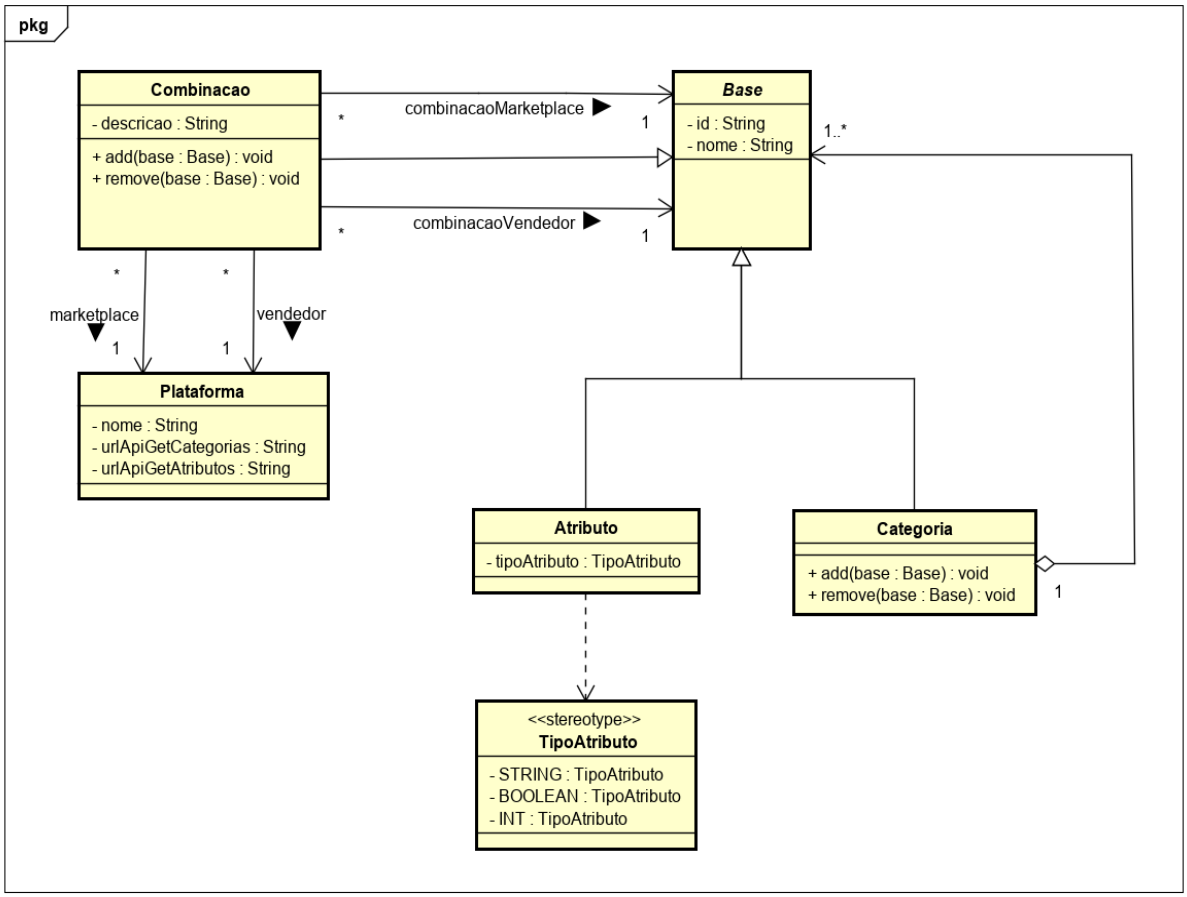
Através dos casos de uso e requisitos levantados, tem-se uma visão geral do comportamento do sistema. É possível, também, identificar as relações entre os atores e processos e, consequentemente, quais objetos que precisaram ser representados no sistema. De acordo com Sommerville (2011), esses objetos, que são identificados como essenciais, dão origem às classes representadas nos diagramas.

O diagrama criado para representar o protótipo de combinação de categorias se baseia no padrão *Composite*. No diagrama, é possível identificar a classe abstrata Base, que é o principal componente desse padrão. Essa classe é base para as outras duas classes que representam uma taxonomia: Categoria e Atributo. Além disso, ela também é a base da classe que representa as combinações. Por fim, a classe

Plataforma é utilizada para representar tanto as plataformas de *marketplace*, quanto vendedores dentro do sistema.

Conforme explica Sommerville (2011), no padrão *Composite* a composição criada pode ser utilizada de forma recursiva. Para entendimento, com fim meramente didático, pode-se imaginar um motor de um carro. Esse motor é composto de peças; essas peças podem ou não ser compostas por outras peças, que são compostas por outras peças e assim por diante. É possível fazer uma analogia desse exemplo com o protótipo para combinação de taxonomias de catálogos eletrônicos desenvolvido neste estudo. A taxonomia em si representa o motor. Esta taxonomia é composta por categorias (nível 1 ou linha), que são compostas por outras categorias (nível 2 ou família), que podem ou não ser compostas por outras categorias (nível 3 ou grupo). Nas combinações, que são representadas pela classe Combinacao, o comportamento é o mesmo. Além disso, esse modelo foi utilizado para representação dos atributos. Contudo, os atributos não possuem subníveis. Portanto, como é possível verificar na Figura 12, a classe Atributo não possui os métodos *add* e *remove.*

Figura 12 – Diagrama de classes do sistema.



Fonte: Elaborado pelo autor (2019).

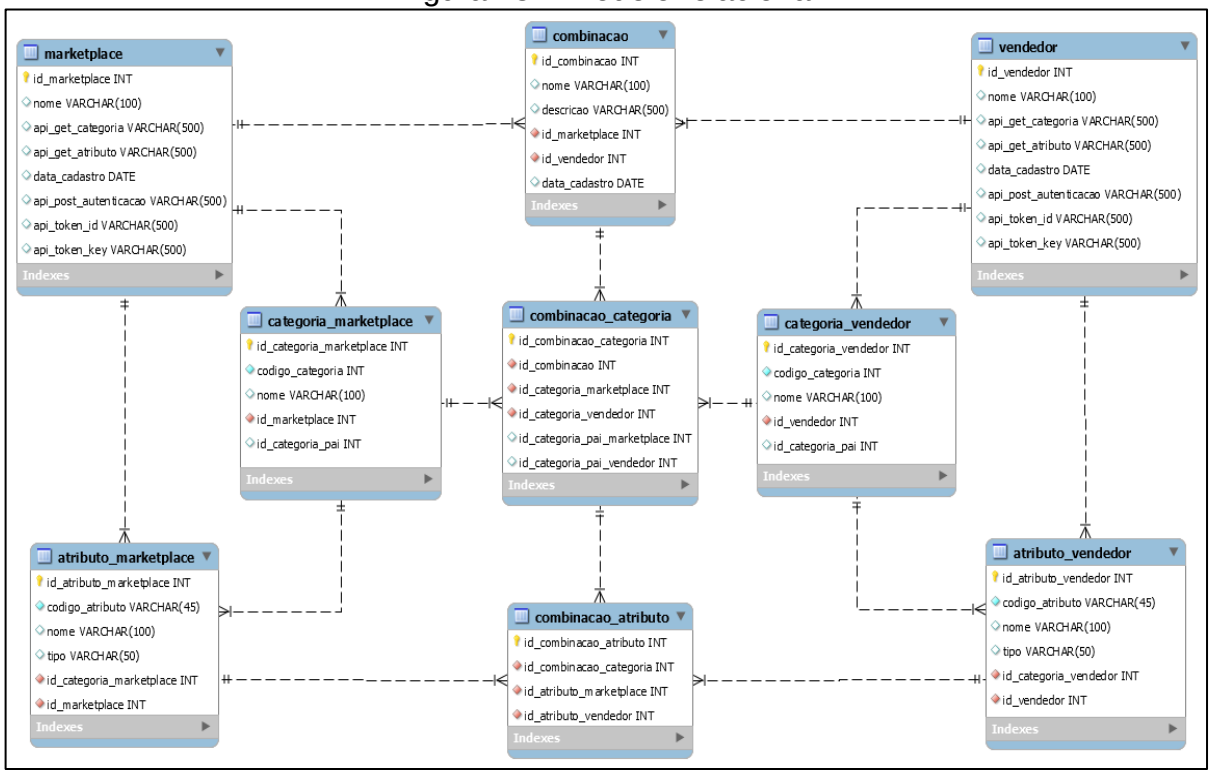
4.1.5. Modelo Relacional

No desenvolvimento do protótipo, será utilizado um banco relacional chamado *Postgresql*, e, para representar a estrutura necessária, foi criado um modelo relacional que pode ser analisado na Figura 13. Segundo Elmasri e Navathe (2005), um modelo relacional representa um conjunto de relações. Tais relações são pensadas como tabelas de valores em que cada linha da tabela representa uma coleção de dados relacionados. Esses dados representam um fato que corresponde a uma entidade ou relacionamento. Já o nome da tabela e os nomes das colunas são usados para interpretação dos signicados dos valores de cada linha.

De acordo com Elmasri e Navathe (2005), na representação formal de um modelo relacional a linha é chamada de tupla, as colunas como atributos e a tabela em si é chamada de relação. Além disso, os tipos de dados que podem ser contidos em cada uma das colunas também podem ser representados nesse modelo.

Em modelos relacionais é possível identificar, também, o tipo de relação entre duas tabelas. Segundo Elmasri e Navathe (2005), esse relacionamento binário é chamado de razão de cardinalidade. Ainda segundo o autor, a cardinalidade especifica o número máximo de instâncias de relacionamento em que uma entidade pode participar.

Figura 13 – Modelo relacional



Fonte: Elaborado pelo autor (2019)

No modelo criado, a plataforma de *marketplace* e o vendedor são representados pelas tabelas “marketplace” e “vendedor”. A conclusão do cadastro, tanto de um *marketplace* quanto de um vendedor, resulta no consumo das respectivas APIs para obtenção das taxonomias. Os dados referentes as categorias da taxonomia serão persistidos nas tabelas “categoria_marketplace” e “categoria_vendedor’. Já os atributos retornados, serão persistidos nas tabelas “atributo_marketplace” e “atributo_vendedor”.

Após o processo de cadastro do *marketplace* e vendedor, as taxonomias estarão disponíveis para realização da combinação. Quando uma combinação é cadastrada, os dados básicos são persistidos na tabela “combinacao”, as relações entre categorias na tabela “combinacao_categoria” e as relações de atributos na tabela “combinacao_atributo”.

Conforme é possível verificar no modelo relacional, a estrutura criada possui uma cardinalidade definida. Entende-se que uma plataforma de *marketplace* ou um vendedor deverão ter uma ou mais categorias atreladas que, por sua vez, terão um ou mais atributos relacionados. Tanto as categorias quanto os atributos deverão ter apenas uma plataforma ou vendedor relacionados. Uma plataforma e um vendedor

poderão ter uma ou mais combinações atreladas. Já as combinações deverão possuir uma ou mais combinações de categorias que terão uma ou mais combinação de atributos.

5. DESENVOLVIMENTO DA PROPOSTA DE SOLUÇÃO

Em conformidade com os requisitos e casos de uso definidos na proposta de solução, foi desenvolvido o protótipo de sistema. O processo de desenvolvimento iniciou-se pela preparação do ambiente de desenvolvimento e instalação de todas as dependências necessárias. Posteriormente, foi implementada a tela de gerenciamento do sistema, relacionada ao caso de uso Gerenciar Sistema. Após esta fase inicial, os cadastros de *marketplace* e vendedor foram desenvolvidos. Somente após a conclusão destes cadastros que o módulo do sistema responsável pela combinação de categorias e atributos foi desenvolvido. Por fim, foi desenvolvida a API REST para consulta das combinações criadas no sistema. Conforme definido na proposta, o protótipo foi desenvolvido com a linguagem de programação Java, *framework Spring Boot* e utilizando a arquitetura MVC.

Uma empresa responsável por uma plataforma de *marketplace* própria, localizada no estado do Rio Grande do Sul, foi contatada durante o desenvolvimento do projeto. Esta empresa, mesmo que de forma sigilosa, forneceu dados relativos à sua taxonomia e disponibilizou sua API de *marketplace* para eventuais testes. Após a análise das informações coletadas, algumas mudanças pontuais foram realizadas na arquitetura do protótipo, para atender a necessidade real desta empresa, que, no presente trabalho, será referenciada como “empresa XY”.

5.1. ARQUITETURA IMPLEMENTADA

A arquitetura MVC e o padrão de projetos *composite* foram a base arquitetural do protótipo desenvolvido. Entretanto, a implementação do padrão *composite* sofreu uma alteração após a análise da estrutura taxonômica do catálogo de produtos da empresa XY.

Inicialmente a implementação do padrão *composite* estava prevista para todo o processo de combinação de taxonomias, como pode ser visto na Figura 12. Contudo, a estrutura do catálogo da empresa XY é muito restrita, definida obrigatoriamente em três níveis de categoria. Estes níveis foram denominados “linha”, “família” e “grupo”, sendo “grupo” o nível mais específico da hierarquia. Levando em consideração este modelo de taxonomia, foram criadas as classes “Combinacao” e “CombinacaoTO” representando a estrutura da empresa XY. Esta modificação

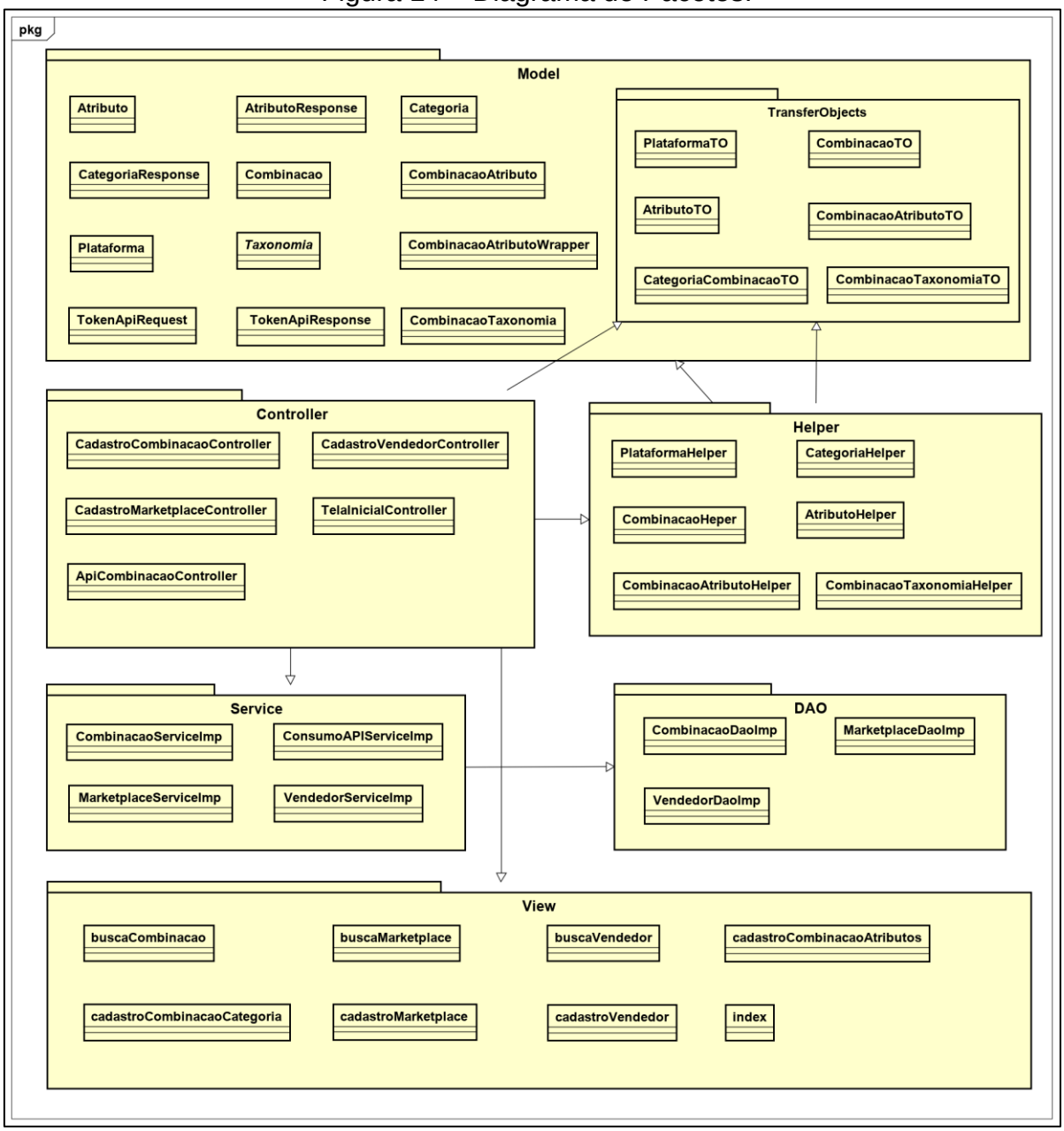
contribuiu para a legibilidade do código, deixando-o menos complexo e contribuindo com a manutenibilidade do sistema. Apesar da mudança ocorrida, o padrão *composite* ainda foi bastante utilizado, principalmente no processo de consumo de APIs e persistência dos dados relativos a categorias e atributos dos vendedores e marketplaces.

No decorrer do desenvolvimento do protótipo, para garantir uma melhor organização e reaproveitamento de código, foi optado pela utilização do padrão *Data Access Object* (DAO). Conforme Engholm Junior (2013), o padrão DAO é utilizado para o encapsulamento da lógica de acesso ao banco de dados, promovendo a reutilização do código uma vez que não há o acoplamento à camada de negócios. Ainda segundo o autor, juntamente com este padrão pode ser utilizado o padrão *TransferObject* (TO) para trocas de dados entre componentes do sistema. No protótipo desenvolvido este padrão também foi utilizado, entretanto, não em sua totalidade.

Na implementação foram definidos dois tipos de classes na camada *Model*. As classes de negócio e as classes TO. As classes de negócio são utilizadas internamente pelo sistema e transacionam dados entre as camadas *Controller*, *Service* e DAO. Já as classes TO são utilizadas para transacionar dados apenas entre a camada de *View* e a camada *Controller*. Foi optado pelo uso do padrão TO devido à complexidade que o sistema adquiriu durante seu desenvolvimento. Este padrão foi implementado após grande parte do desenvolvimento já ter sido realizado. Por este motivo, algumas classes ainda precisam de adequação, pois transacionam apenas objetos TO entre as camadas.

Para conversão dos objetos TO em objetos de negócio foram criadas classes com métodos estáticos localizadas em uma camada chamada *Helper*. A Figura 14 representa as camadas do sistema, bem como as classes e arquivos HTML referentes às interfaces gráficas.

Figura 14 – Diagrama de Pacotes.

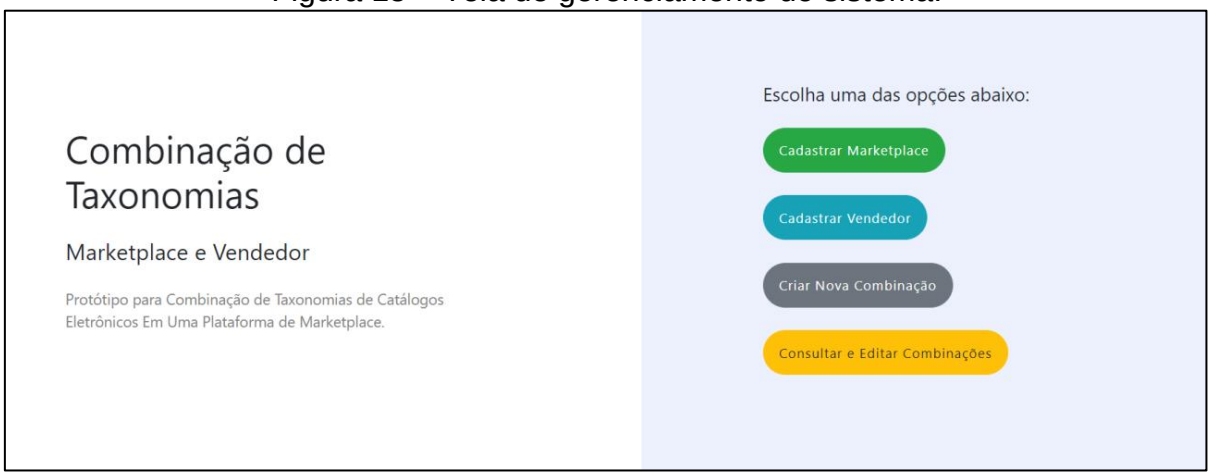


Fonte: Elaborado pelo autor (2019)

5.2. GERENCIAR SISTEMA

Com o objetivo de possibilitar a navegação do usuário entre as funcionalidades do sistema, foi desenvolvida a tela de navegação inicial. Esta foi a primeira tela desenvolvida no protótipo e a partir do seu desenvolvimento estabeleceu-se a estrutura de todos os componentes gráficos do sistema. Um modelo desta tela pode ser analisado na Figura 15.

Figura 15 – Tela de gerenciamento do sistema.



Fonte: Elaborado pelo autor (2019)

Além de HTML e CSS, foram utilizadas diversas outras tecnologias de desenvolvimento de interfaces gráficas web, são elas: *Thymeleaf*, B*ootstrap* e *Jquery*.

Thymeleaf[21] é um motor de *templates server-side* utilizado em conjunto com a linguagem de programação Java. Os modelos HTML escritos com *Thymeleaf* ainda parecem e funcionam como HTML, permitindo que *o de design* dos artefatos possam ser modificados normalmente. O *Boostrap*[22] por sua vez, se trata de uma biblioteca de componentes para design de interfaces gráficas desenvolvida com HTML, CSS e *JavaScript*. Esta biblioteca abstrai o uso de código CSS e Javascript, além disso, os componentes existentes melhoram a experiência do usuário através de *templates* amigáveis e responsivos. A Figura 16 mostra um trecho de código utilizando *Thymeleaf* juntamente com *Boostrap*.

[21] Documentação *Thymeleaf* – disponível em: <https://www.thymeleaf.org/>. Acesso em 12/11/2019.
[22] Documentação Bootstrap – disponível em: <https://getbootstrap.com/> . Acesso em 12/11/2019.

Figura 16 – Exemplo *Thymeleaf* e *Boostrap.*

```
<div class="row">
  <div class="col-12 " th:if="${resultadoBuscaVendedores != null}">
    <table class="table">
      <div class="row mt-3 mx-4">
        <thead>
          <tr>
            <th scope="col">ID VENDEDOR</th>
            <th scope="col">NOME</th>
            <th scope="col">API GET CATEGORIAS</th>
            <th scope="col">API GET ATRIBUTOS</th>
            <th scope="col"></th>
          </tr>
        </thead>
      </div>
      <div class="row mt-3 mx-4">
        <tbody>
          <th:block th:each="vendedor : ${resultadoBuscaVendedores}">
            <tr>
              <th scope="row" th:text="${vendedor.id}"></th>
              <td th:text="${vendedor.nome}"></td>
              <td th:text="${#strings.substring(vendedor.urlAPIGetCategorias,0,30)}+' ...'"></td>
              <td th:text="${#strings.substring(vendedor.urlAPIGetAtributos,0,30)}+' ...'"></td>
              <td>
                  <a th:href="@{'/editarVendedor/' + ${vendedor.id}}">Editar</a>
              </td>
```

Fonte: Elaborado pelo autor (2019)

Jquery[23] é uma biblioteca *JavaScript* muito versátil e simples, que facilita a utilização de animações, manipulação de eventos e manipulação de documentos HTML em tempo real. No protótipo, o Jquery foi utilizado em todas as interfaces, principalmente na tela de combinação de categorias e atributos em conjunto com o *Boostrap*, como mostra a Figura 17.

Figura 17 – Exemplo de código com utilização de *Jquery*

```
function populaLinhaMarketplace(marketplace) {

    $.ajax({
        url: "/marketplacelinha",
        data: {idMarketplace: $(marketplace).val()},
        dataType: 'JSON',
        beforeSend: function () {

            $('#linhaMarketplaceSelect')
            .find('option')
            .remove()
            .end();
        }
    }).done(function (data) {

        if (data) {
            $("#linhaMarketplaceSelect").append("<option value=0>Selecione uma opção</option>");
            data.forEach(function (obj) {
                $("#linhaMarketplaceSelect").append("<option value=" + obj.idCategoria + ">" + obj.nome + "</option>");
            });

            $('#linhaMarketplaceSelect').selectpicker('refresh');

        } else {
            alert("Erro ao buscar categoria!");
        }
```

Fonte: Elaborado pelo autor (2019)

[23] Documentação JQuery – disponível em: <https://jquery.com/> . Acesso em 12/11/2019.

5.3. CADASTRO DE *MARKETPLACE* E VENDEDOR

O cadastro de *marketplace* e vendedor são pré-requisitos para que as combinações de categorias e atributos sejam criadas. Apesar do cadastro de vendedores e do cadastro de marketplaces possuírem estruturas separadas tanto na aplicação, como no banco de dados, o desenvolvimento a nível de código e os processos utilizados são os mesmos. Até mesmo graficamente estes dois módulos do sistema são muito similares. Por estes motivos, os exemplos relacionados a seguir levarão em consideração apenas o cadastro do *marketplaces*.

O processo de cadastro de um *marketplace* é iniciado pelo acesso à tela de cadastro. Na tela inicial, quando o usuário clica no botão “Cadastrar Marketplace”, a aplicação redireciona-o para a URL “/marketplace”. Esta URL está mapeada em uma classe chamada “CadastroMarketplaceController”, localizada na camada *controller*. Nesta classe, o método “getMarketplace” adiciona um objeto do tipo “Plataforma” como atributo do objeto “*model*” proveniente da classe nativa do Spring de mesmo nome. Por fim, o método retorna a *view* de busca de *marketplace*. A Figura 18 destaca o trecho do código descrito acima.

Figura 18 – CadastroMarketplaceController

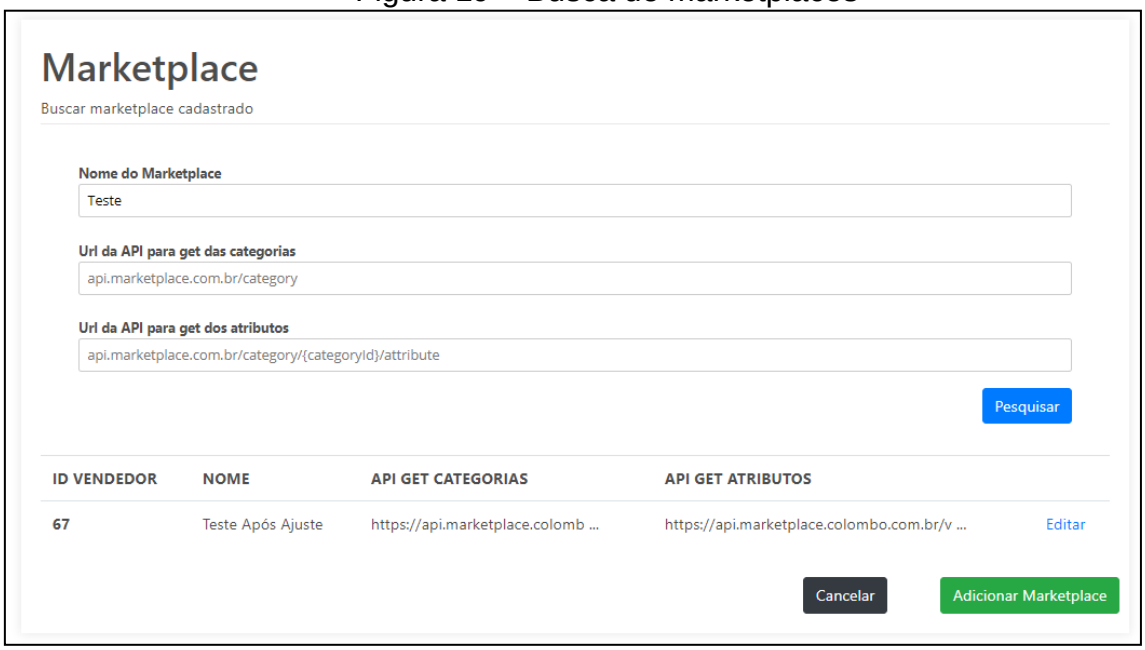
```
@RequestMapping(value = "/marketplace", method = RequestMethod.GET)
public String getMarketplace(Model model, Plataforma marketplace) {
    if (!marketplace.isEmpty()) {
        model.addAllAttributes(getAtributosBusca(marketplace));
    }
    model.addAttribute("marketplace", marketplace);

    return "buscaMarketplace";
}
```

Fonte: Elaborado pelo autor (2019)

Na tela de busca de *marketplace* retornada, o usuário terá a opção de buscar um marketplace existente para edição ou então poderá optar pelo cadastro de um novo marketplace através do botão “Adicionar Marketplace”. Ao pesquisar, o usuário é direcionado para o método “getMarketplace” já citado anteriormente, contudo, o objeto do tipo “Plataforma” estará instanciado com os dados preenchidos nos campos disponíveis na tela. Caso isso ocorra, a aplicação irá buscar os marketplaces existentes com base nos dados informados e então irá exibi-los na tela, conforme a Figura 19.

Figura 19 – Busca de marketplaces



Fonte: Elaborado pelo autor (2019)

Caso o usuário opte pela edição de um *marketplace* existente, deverá clicar sob o *link* “Editar”. Este *link* direciona o usuário para a URL “/editarMarketplace/{id}”, atribuindo o campo id do *marketplace* selecionado como parâmetro. Então, o método do *controller* responsável por mapear essa URL irá buscar o marketplace em questão através do id, adicioná-lo como atributo do objeto model e retornará a *view* cadastroMarketplace, conforme a Figura 20. Como a *view* de edição e cadastro de *marketplace* são a mesma, o processo que ocorre ao selecionar o botão “Adicionar Marketplace” é muito similar, a única diferença é que nenhum *marketplace* será buscado e adicionado ao *model*. A tela de cadastro e alteração de *marketplace* pode ser analisada na Figura 21.

Figura 20 – Método getEditarMarketplace

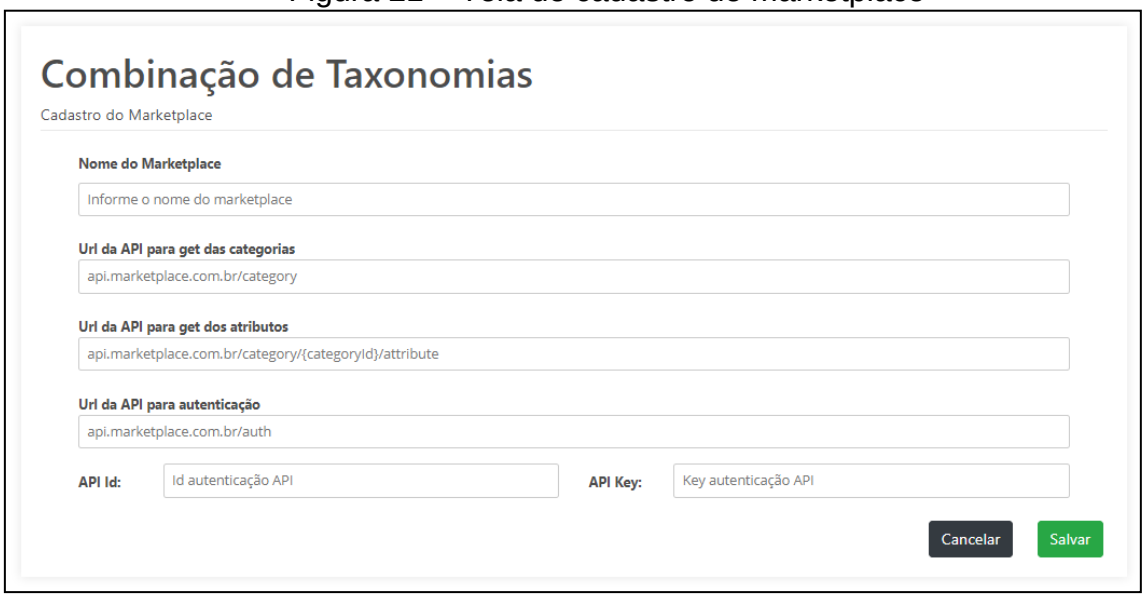
```
@RequestMapping(value = "/editarMarketplace/{id}", method = RequestMethod.GET)
public String getEditarMarketplace(@PathVariable Long id, Model model) {

    Plataforma marketplace = marketplaceService.buscaMarketplacePorId(id);
    model.addAttribute("marketplace", marketplace);

    return "cadastroMarketplace";
}
```

Fonte: Elaborado pelo autor (2019)

Figura 21 – Tela de cadastro de marketplace



Fonte: Elaborado pelo autor (2019)

O processo de cadastro é o mesmo para *marketplaces* e vendedores. Inicia-se pelo preenchimento dos campos do formulário e é concluído no momento em que o usuário clica no botão “Salvar”. Ao clicar neste botão, tanto na edição, quanto no cadastro de um novo *marketplace*, o usuário é direcionado para a URL “/salvacadastromarketplace” mapeada no *controller* “CadastroMarketplaceController”.

O método responsável por esse mapeamento faz a distinção entre uma edição e um novo cadastro, utilizando dois processos distintos para salvar os dados. Este método pode ser visto na Figura 22.

Figura 22 – Método postCadastroMarketplace

```
@RequestMapping(value = "/salvacadastromarketplace", method = RequestMethod.POST)
public String postCadastroMarketplace(Plataforma marketplace){
    if (marketplace.getId() != null) {
        marketplaceService.alterarMarketplace(marketplace);
    }else {
        marketplaceService.inserirMarketplace(marketplace);
    }
    return "index";
}
```

Fonte: Elaborado pelo autor (2019)

Outro detalhe que pode ser analisado na Figura 31 é a utilização da interface “MarketplaceService” para acesso às classes da camada de service. Isso ocorre porque o *Spring Boot,* framework utilizado para desenvolvimento, exige que a aplicação seja separada em camadas bem definidas. Para atender tal necessidade,

cada classe da camada de serviço e DAO possui sua respectiva interface. Estas interfaces são utilizadas como meio de acesso às classes de cada camada.

O método “inserirMarketplace” pertencente a camada de serviço possui três etapas: Persistência dos dados cadastrais do marketplace, consulta à API cadastrada para persistência dos dados relativos à estrutura de categorias e, por fim, consulta recursiva à API cadastrada para persistência dos dados relativos à estrutura de atributos. Estas etapas são pré-requisitos para que o cadastro de uma plataforma de marketplace ou vendedor seja concluído. Somente após este processo que as combinações de categorias e atributos poderão ser realizadas.

Para realizar o consumo das APIs cadastradas foram utilizados recursos específicos do *Spring Boot*. O principal deles é a classe nativa do Spring chamada *RestTemplate*. Esta classe possui uma série de métodos de fácil implementação para tratar requisições com APIs REST. No trecho de código da Figura 23 são exibidos os três métodos da classe “ConsumoApiServiceImp”. Esta classe é responsável por todas as consultas em APIs que o sistema realiza durante o cadastro de vendedores e *marketplaces*. O primeiro método, “getAutenticacao”, é responsável por gerar o *token* de autenticação na API do seller ou marketplace. O segundo, “getCategorias”, realiza uma requisição GET na API utilizando o *token* gerado como parâmetro. Para consulta dos atributos por categoria o processo é similar, a diferença é que o código da categoria também é passado como parâmetro. Em todos os métodos a classe *RestTemplate* é utilizada.

Figura 23 – Classe ConsumoApiServiceImp

```
@Service("consumoApiService")
public class ConsumoApiServiceImp implements ConsumoApiService{

    private RestTemplate restTemplate = new RestTemplate();

    @Override
    public void getAutenticacao(Plataforma marketplace) {
        TokenApiRequest tokenApiRequest = new TokenApiRequest(marketplace.getApiTokenId(),marketplace.getApiTokenKey());

        ResponseEntity<TokenApiResponse> response = restTemplate.postForEntity(marketplace.getUrlApiPostAutenticacao(),
                                                                                tokenApiRequest, TokenApiResponse.class);

        TokenApiResponse token = response.getBody();

        TokenApi.TOKEN = "Bearer "+ token.getAccess_token();

    }

    @Override
    public List<CategoriaResponse> getCategorias(Plataforma plataforma){

        getAutenticacao(plataforma);

        MultiValueMap<String, String> headers = new LinkedMultiValueMap<>();
        headers.add("Authorization", TokenApi.TOKEN);

        RequestEntity<Object> request = new RequestEntity<>(
                headers, HttpMethod.GET, URI.create(plataforma.getUrlAPIGetCategorias()));


        ResponseEntity<CategoriaResponse[]> response = restTemplate.exchange(request, CategoriaResponse[].class);


        return Arrays.asList(response.getBody());

    }

    @Override
    public List<AtributoResponse> getAtributos(Plataforma plataforma, Integer idCategoria){

        MultiValueMap<String, String> headers = new LinkedMultiValueMap<>();
        headers.add("Authorization", TokenApi.TOKEN);

        String url = UrlApi.preparaUrlGetAtributos(plataforma.getUrlAPIGetAtributos(), idCategoria.toString());
        RequestEntity<Object> request = new RequestEntity<>(
                                        headers, HttpMethod.GET, URI.create(url));

        ResponseEntity<AtributoResponse[]> response = restTemplate.exchange(request, AtributoResponse[].class);


        return Arrays.asList(response.getBody());

    }

}
```

Fonte: Elaborado pelo autor (2019)

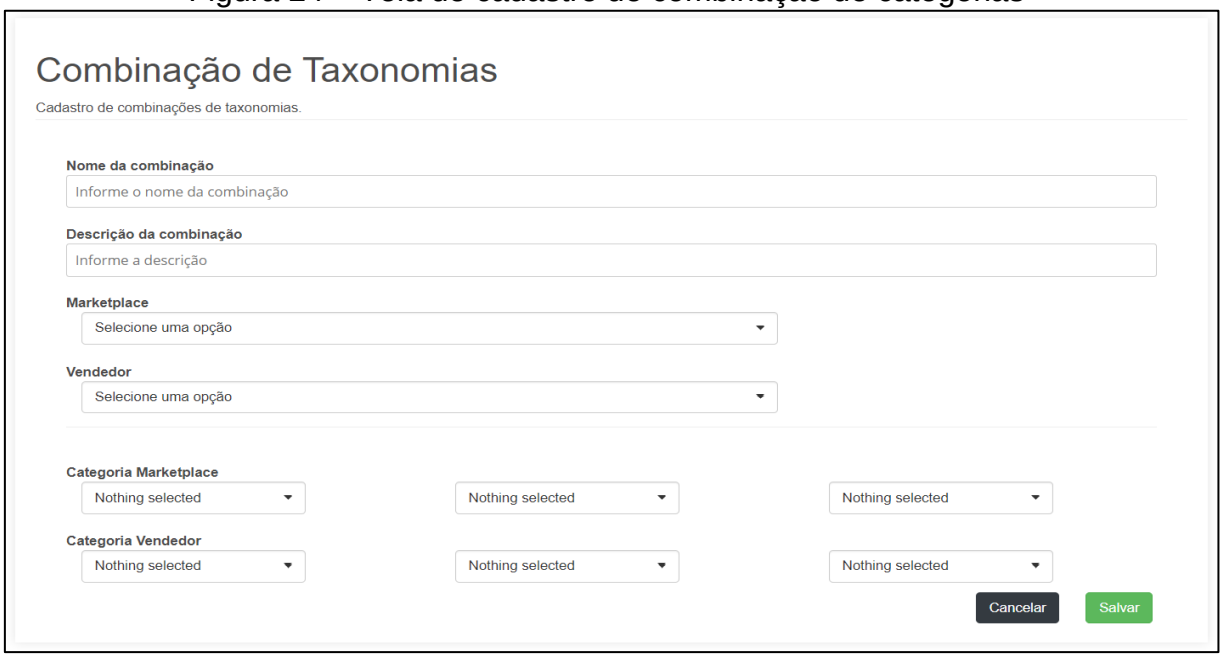
5.4. COMBINAÇÃO DE TAXONOMIAS

A combinação de categorias e atributos que compõe a taxonomia dos *marketplaces* e vendedores é o principal objetivo do protótipo. Este processo só pode ser iniciado após o cadastro de pelo menos um *marketplace* e um vendedor.

O primeiro passo para criação de uma nova combinação é o acesso à tela de cadastro. Para tal, o usuário deve clicar no botão “Criar Nova Combinação”, presente na tela inicial do sistema. Este botão redireciona o usuário para a URL “/cadastrocombinacaocategoria”, que está mapeada no “CadastroCombinacaoController”. Este *controller* busca e atribui à view todos os marketplaces e vendedores cadastrados no sistema, além de um objeto do tipo

“CombinacaoTO”. A Figura 24 mostra a tela de cadastro de combinação de categorias.

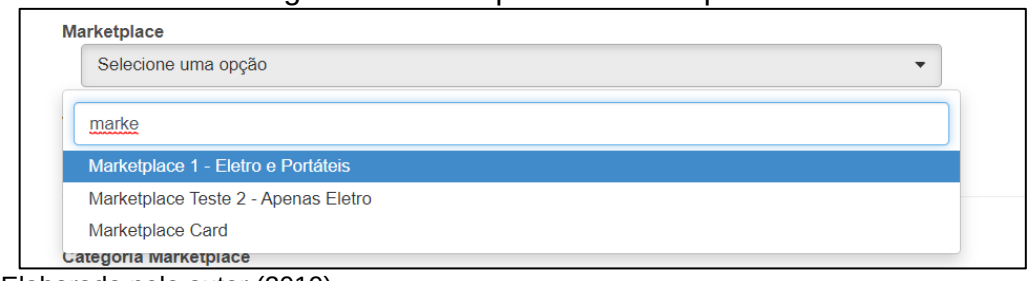
Figura 24 – Tela de cadastro de combinação de categorias



Fonte: Elaborado pelo autor (2019)

Para melhorar a experiência do usuário foram utilizados diversos recursos provenientes do Bootstrap e Jquery. Nas telas de combinação de categorias e atributos em especial, o componente *selectpicker* foi amplamente aplicado. Este componente possibilita que dados contidos em tags HTML do tipo *select* sejam pesquisados em tempo real, facilitando e agilizando o processo de combinação. A correta implementação deste componente exigiu um tempo considerável de desenvolvimento, pois, as versões do Bootstrap e Jquery inicialmente utilizadas não eram compatíveis com esta tecnologia. A Figura 25 demonstra a utilização deste componente no código e também como ele é renderizado ao usuário.

Figura 25 – Componente selectpicker



Fonte: Elaborado pelo autor (2019)

Grande parte das funcionalidades da tela de cadastro de combinação de

categorias não são executadas ao carregar a tela inicialmente. As funcionalidades são executadas conforme a interação do usuário na tela através de código JavaScript e JQuery. Ao selecionar um *marketplace* o sistema busca, em tempo de execução, o primeiro nível de categoria cadastrada levando em consideração o código do *marketplace* selecionado. Desta forma, apenas as categorias correspondentes ao *marketplace* selecionado serão exibidas, melhorando a experiência do usuário e facilitando o processo de cadastro. O mesmo também ocorre quando um vendedor é selecionado.

Na seleção de categorias também houve a preocupação em facilitar o cadastro. Sempre que uma categoria é escolhida, funções JavaScript são executadas buscando as categorias filhas tendo como base a categoria pai escolhida. A implementação destas funcionalidades não é complexa, contudo, problemas foram encontrados ao desenvolver a edição de combinações de categorias que também utiliza esta tela. Isso porque, para auxiliar ao máximo o usuário, foi definido que a combinação de categorias existente deveria ser exibida ao usuário ao acessar a tela para edição. Desta forma, o mesmo poderia realizar apenas as alterações necessárias levando em consideração o que já havia sido cadastrado anteriormente. Para que este comportamento ocorresse, todas as funções JavaScript executadas de forma assíncrona conforme interação do usuário deveriam ser executadas ao renderizar a página. Entretanto, foi constatado durante os testes que a aplicação Java estava sendo executada muito rapidamente, antes mesmo das funções JavaScript serem carregadas. A alternativa encontrada foi a utilização da função *setTimeout,* provida pelo JQuery, para atrasar a execução das funções em alguns milésimos de segundo. Na Figura 26 pode ser analisado o trecho de código JavaScript mencionado.

Figura 26 – JavaScript executado ao editar combinação

```
$(document).ready(function () {
    $("#bntCancelaCadastro").click(function(){
        location.href="/";
    });

    if ($('#idCombinacaoCategoria').val() > 0) {

        var linhaMarketplace = ($("#idLinhaMarketplaceSelecionada").val());
        var familiaMarketplace = ($("#idFamiliaMarketplaceSelecionada").val());
        var grupoMarketplace = ($("#idGrupoMarketplaceSelecionada").val());

        var linhaVendedor = ($("#idLinhaVendedorSelecionado").val());
        var familiaVendedor = ($("#idFamiliaVendedorSelecionado").val());
        var grupoVendedor = ($("#idGrupoVendedorSelecionado").val());

        populaLinhaMarketplace($("#idMarketplaceSelect"));
        setTimeout( function(){
            $('#linhaMarketplaceSelect').val(linhaMarketplace);
            $('#linhaMarketplaceSelect').selectpicker('refresh');
        }  , 2000 );

        populaFamiliaMarketplace(linhaMarketplace);
        setTimeout( function(){
            $('#familiaMarketplaceSelect').val(familiaMarketplace);
            $('#familiaMarketplaceSelect').selectpicker('refresh');
        }  , 2050 );

        populaGrupoMarketplace(familiaMarketplace);
        setTimeout( function(){
            $('#grupoMarketplaceSelect').val(grupoMarketplace);
            $('#grupoMarketplaceSelect').selectpicker('refresh');
        }  , 2100 );


        populaLinhaVendedor($("#idVendedorSelect"));
        setTimeout( function(){
            $('#linhaVendedorSelect').val(linhaVendedor);
            $('#linhaVendedorSelect').selectpicker('refresh');
        }  , 2000 );

        populaFamiliaVendedor(linhaVendedor);
        setTimeout( function(){
            $('#familiaVendedorSelect').val(familiaVendedor);
            $('#familiaVendedorSelect').selectpicker('refresh');
        }  , 2050 );

        populaGrupoVendedor(familiaVendedor);
        setTimeout( function(){
            $('#grupoVendedorSelect').val(grupoVendedor);
            $('#grupoVendedorSelect').selectpicker('refresh');

        }  , 2100 );

    }
});
```

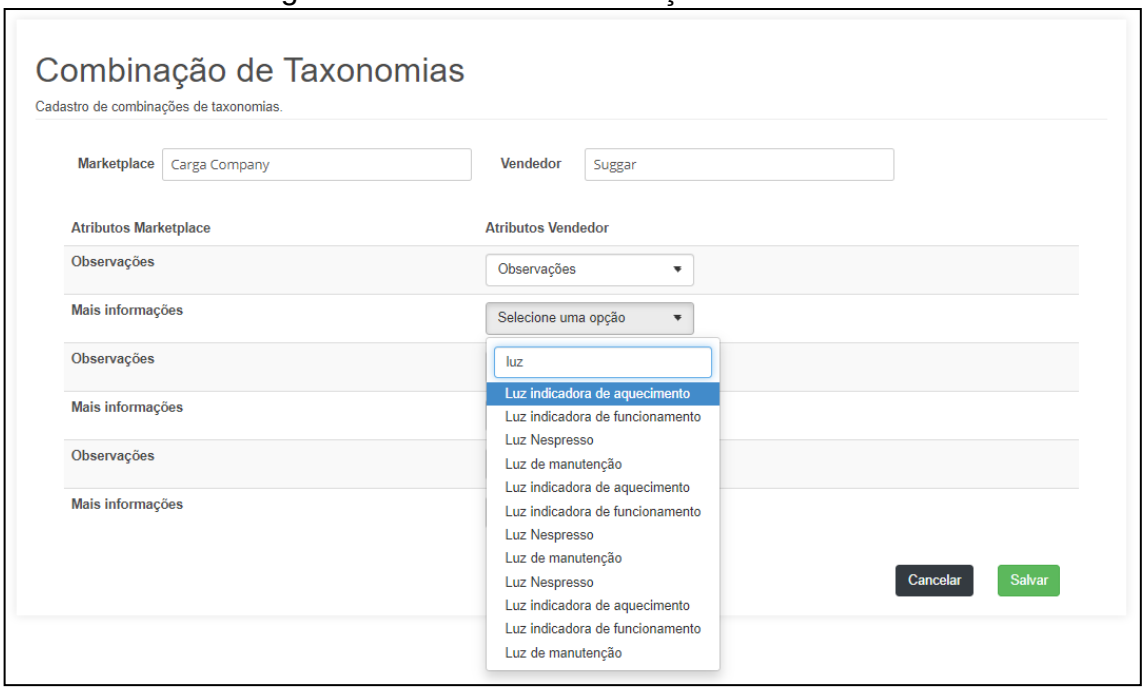
Fonte: Elaborado pelo autor (2019)

Quanto o usuário finaliza a combinação de categorias e clica no botão “Salvar”,

as informações são persistidas no bando de dados e o usuário é direcionado à tela de combinação de atributos. Com base no *marketplace*, vendedor e categorias selecionadas na combinação de categorias a aplicação irá buscar todos os atributos existentes, separando-os em atributos de *marketplace* e vendedor. Após relacionar os atributos e clicar em salvar, os dados são persistidos no banco de dados. Um modelo da tela de combinação de atributos pode ser analisado a Figura 27.

Esta etapa do desenvolvimento do protótipo foi a que mais demandou tempo em todo o projeto. O desenvolvimento desta tela foi complexo devido à necessidade de criar recursivamente os *inputs* HTML dos atributos do *marketplace* em uma tabela e, a cada iteração recursiva, criar um componente *select* com todos os atributos do vendedor. Conforme a documentação disponibilizada pelo próprio Thymeleaf[24], para adicionar *inputs* HTML ao código deve-se utilizar a o comando “th:field”. Contudo, para a situação descrita acima, este comando não funciona corretamente, retornando um erro ao carregar página. Para contornar este problema foi utilizado o comando “th:attr”, normalmente utilizado para definir um atributo *value* em inpunts HTML, para definição do atributo *name*. Mais detalhes da solução adotada podem ser visualizados na Figura 28.

Figura 27 – Tela de combinação de atributos



Fonte: Elaborado pelo autor (2019)

[24] Documentação Thymeleaf – disponível em: <https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html> . Acesso em 12/11/2019.

Figura 28 – View cadastroCombinacaoAtributos.html

```
h:block th:each="atributoMarketplace, stat : ${atributosMarketplace}">
<input th:value="${atributoMarketplace.idAtributo}" th:attr="name='combinacaoAtributos[' + ${stat.index} + '].idAtributoMarketplace'"
<tr>
  <th scope="row" th:text="${atributosMarketplace[__${stat.index}__].nome}"></th>

  <td>
      <select class="selectpicker" data-live-search="true" th:field="*{combinacaoAtributos[__${stat.index}__].idAtributoVendedor}">
          <option th:value="0">Selecione uma opção</option>
          <option th:each="atributoVendedor : ${atributosVendedor}" th:value="${atributoVendedor.idAtributo}" th:text="${atributoVend
      </select>
  </td>
</tr>
th:block>
```

Fonte: Elaborado pelo autor (2019)

5.5. CONSULTA DE COMBINAÇÕES

Foram criadas duas formas de visualização das combinações criadas. A consulta visual e a consulta via API. A consulta visual pode ser realizada pesquisando e selecionando a combinação existente na tela de consulta de combinações. Já a consulta via API deve ser realizada consumindo a API desenvolvida no protótipo.

O processo para consulta visual de combinações inicia-se com o acesso à tela de consulta de combinações. Posteriormente, o usuário deve buscar a combinação desejada através dos filtros da tela de busca. Com base nas informações preenchidas, a aplicação realizará uma consulta no banco de dados e retornará as combinações encontradas, conforme a figura 29.

Figura 29 – Tela para consulta de combinações

Fonte: Elaborado pelo autor (2019)

Para construção da API de consulta de combinações foram utilizados recursos do *Spring Boot* e da própria linguagem de programação Java. A anotação “@RestController” provida pelo framework Spring Boot, foi o recurso mais útil desta etapa do desenvolvimento. Esta anotação foi introduzida no Spring 4.0[25] para simplificar a criação de serviços RESTful. Ela combina recursos das anotações “@Controller” e “@ResponseBody”, eliminando a necessidade de anotar todos os métodos de tratamento de solicitações da classe *controller* com a anotação “@ResponseBody”.

Para o usuário ter acesso a API bastar fazer uma requisição para a URL “/getCombinacoesPorMarketplace” informando o parâmetro “idMarketplace” com o id do *marketplace* ao qual deseja consultar as combinações existentes. No *controller* que mapeia esta URL a aplicação realiza uma busca em todas as combinações atreladas ao id do *marketplace* informado. Assim como em todas as outras funcionalidades do protótipo, a classe da camada *controller* acessa a camada *service*, que acessa a camada de DAO. A Figura 30 retrata o *controller* mencionado, seguido pela Figura 31 com o trecho de código da camada *service* com o método chamado pelo *controller.* Por fim, a Figura 32 detalha o código da camada de DAO utilizada. As consultas no banco de dados realizadas nesta etapa são as mais complexas da aplicação, ligando quase todas as tabelas existentes. Isso faz com que o JSON retornado pela API disponha de todas as informações necessárias.

Figura 30 – Controller “ApiCombinacaoController"

```
@RestController
public class ApiCombinacaoController {
    @Autowired
    private CombinacaoService combinacaoService;

    private CombinacaoHelper combinacaoHelper;

    private CombinacaoTaxonomiaHelper combinacaoTaxonomiaHelper;

    @RequestMapping("/getCombinacoesPorMarketplace")
    public List<CombinacaoTaxonomiaTO> getCombinacoesTaxonomiaPorMarketplace(Long idMarketplace){

        List<CombinacaoTaxonomia> combinacaoTaxonomia = combinacaoService.BuscaCombinacoesTaxonomiaPorMarketplace(idMarketplace);
        List<CombinacaoTaxonomiaTO> combinacaoTaxonomiaTO = combinacaoTaxonomiaHelper.toListCombinacaoTO(combinacaoTaxonomia);

        return combinacaoTaxonomiaTO;
    }
}
```

Fonte: Elaborado pelo autor (2019)

[25] As anotações Spring @Controller e @RestController – disponível em: <https://www.baeldung.com/spring-controller-vs-restcontroller> . Acesso em 12/11/2019.

Figura 31 – Método “BuscaCombinacoesTaxonomiaPorMarketplace”

```
@Override
public List<CombinacaoTaxonomia> BuscaCombinacoesTaxonomiaPorMarketplace(Long idMarketplace) {
    List<CombinacaoTaxonomia> combinacoes = combinacaoDao.buscaCombinacaoTaxonomiaPorIdMarketplace(idMarketplace);
    for (CombinacaoTaxonomia combinacao : combinacoes) {
        List<CombinacaoAtributo> atributos = combinacaoDao.buscaCombinacaoTaxonomiaAtributos(combinacao.getIdCombinacao());
        combinacao.setAtributos(atributos);
    }

    return combinacoes;
}
```

Fonte: Elaborado pelo autor (2019)

Figura 32 – Métodos da camada de DAO utilizados

```
@Override
public List<CombinacaoTaxonomia> buscaCombinacaoTaxonomiaPorIdMarketplace(Long idMarketplace) {
    StringBuilder sql = new StringBuilder();

    sql.append(" SELECT linha.id_combinacao as idCombinacao, ");
    sql.append("        c.nome as nome, ");
    sql.append("        c.descricao as descricao, ");
    sql.append("        c.id_marketplace as idMarketplace, ");
    sql.append("        c.id_vendedor as idVendedor, ");
    sql.append("        linha.id_categoria_marketplace as idLinhaMarketplace, ");
    sql.append("        linhacm.nome as nomeLinhaMarketplace,    ");
    sql.append("        familia.id_categoria_marketplace as idFamiliaMarketplace, ");
    sql.append("        familiacm.nome as nomeFamiliaMarketplace,          ");
    sql.append("        grupo.id_categoria_marketplace as idGrupoMarketplace, ");
    sql.append("        grupocm.nome as nomeGrupoMarktetplace,       ");
    sql.append("        linha.id_categoria_vendedor as idLinhaVendedor, ");
    sql.append("        linhacv.nome as nomeLinhaVendedor,          ");
    sql.append("        familia.id_categoria_vendedor as idFamiliaVendedor, ");
    sql.append("        familiacv.nome as nomeFamiliaVendedor,     ");
    sql.append("        grupo.id_categoria_vendedor as idGrupoVendedor, ");
    sql.append("        grupocv.nome as nomeGrupoVendedor ");
    sql.append("   FROM combinacao c ");
    sql.append("   INNER JOIN combinacao_categoria linha on (linha.id_combinacao = c.id_combinacao) ");
    sql.append("   INNER JOIN combinacao_categoria familia on (familia.id_categoria_pai_marketplace = linha.id_categoria_marketplace and ");
    sql.append("                        familia.id_categoria_pai_vendedor = linha.id_categoria_vendedor and  ");
    sql.append("                          familia.id_combinacao = linha.id_combinacao) ");
    sql.append("   INNER JOIN combinacao_categoria grupo on (grupo.id_categoria_pai_marketplace = familia.id_categoria_marketplace and ");
    sql.append("                       grupo.id_categoria_pai_vendedor = familia.id_categoria_vendedor and  ");
    sql.append("                         grupo.id_combinacao = familia.id_combinacao) ");
    sql.append("   INNER JOIN categoria_marketplace linhacm on (linhacm.id_categoria_marketplace = linha.id_categoria_marketplace) ");
    sql.append("   INNER JOIN categoria_marketplace familiacm on (familiacm.id_categoria_marketplace = familia.id_categoria_marketplace) ");
    sql.append("   INNER JOIN categoria_marketplace grupocm on (grupocm.id_categoria_marketplace = grupo.id_categoria_marketplace) ");
    sql.append("   INNER JOIN categoria_vendedor linhacv on (linhacv.id_categoria_vendedor = linha.id_categoria_vendedor) ");
    sql.append("   INNER JOIN categoria_vendedor familiacv on (familiacv.id_categoria_vendedor = familia.id_categoria_vendedor) ");
    sql.append("   INNER JOIN categoria_vendedor grupocv on (grupocv.id_categoria_vendedor = grupo.id_categoria_vendedor) ");
    sql.append("   where c.id_marketplace = :idMarketplace ");

    SqlParameterSource param = new MapSqlParameterSource()
            .addValue("idMarketplace", idMarketplace);

    return template.query(sql.toString(), param, new BeanPropertyRowMapper(CombinacaoTaxonomia.class));
}

@Override
public List<CombinacaoAtributo> buscaCombinacaoTaxonomiaAtributos(Long idCombinacao) {
    StringBuilder sql = new StringBuilder();

    sql.append(" select  ca.id_combinacao as idCombinacao ");
    sql.append(" ,       am.id_marketplace as idMarketplace ");
    sql.append(" ,       av.id_vendedor as idVendedor ");
    sql.append(" ,       am.codigo_atributo as idAtributoMarketplace ");
    sql.append(" ,       am.nome as nomeAtributoMarketplace ");
    sql.append(" ,       am.tipo as tipoAtributoMarketplace ");
    sql.append(" ,       av.codigo_atributo as idAtributoVendedor ");
    sql.append(" ,       av.nome as nomeAtributoVendedor ");
    sql.append(" ,       av.tipo as tipoAtributoVendedor ");
    sql.append("   from combinacao_atributo ca ");
    sql.append("   inner join atributo_marketplace am on (am.id_atributo_marketplace = ca.id_atributo_marketplace) ");
    sql.append("   inner join atributo_vendedor av on (av.id_atributo_vendedor = ca.id_atributo_vendedor) ");
    sql.append("  where ca.id_combinacao = :idCombinacao ");

    SqlParameterSource param = new MapSqlParameterSource()
            .addValue("idCombinacao", Long.valueOf(idCombinacao));

    return template.query(sql.toString(), param, new BeanPropertyRowMapper(CombinacaoAtributo.class));
}
```

Fonte: Elaborado pelo autor (2019)

5.6. TESTES E VALIDAÇÕES

Logo no início do desenvolvimento do protótipo foi acordada a utilização dos dados da plataforma do *marketplace* da empresa XY, bem como sua API. Desde então, um analista da empresa se dispôs a realizar quaisquer testes necessários.

Conforme o desenvolvimento das funcionalidades era concluído o analista era contatado para realização de testes unitários e individuais. Segundo Braga (2016), apesar de básicos, os testes unitários são fundamentais. Este tipo de teste possibilita que todos os módulos do sistema sejam analisados individualmente, verificando cada funcionalidade.

Os eventuais problemas encontrados eram corrigidos e o sistema era testado novamente antes do desenvolvimento das próximas etapas. Após a conclusão do desenvolvimento foi realizado um teste integrado do protótipo, baseando-se nos casos de uso.

Dentre as diversas abordagens para realização de testes de software em geral, foi optado pela utilização da técnica caixa-preta para os testes finais do protótipo desenvolvido. Conforme explica Braga (2016), esta técnica pode ser utilizada para encurtar o tempo de realização dos testes em sistemas altamente complexos, uma vez que não leva em consideração o comportamento interno do sistema. Neste tipo de teste os dados de entrada são fornecidos ao sistema, que realiza determinado processo. Então, as saídas decorrentes destas entradas são comparadas com os resultados esperados já documentados. Para auxiliar o analista da empresa XY na realização dos testes foram criados os casos de teste listados no Apêndice B. O modelo de caso de teste utilizado foi o proposto por Braga (2016).

O analista da empresa XY realizou todos os testes do protótipo, assumindo também o papel de vendedor. O usuário necessitou de auxílio apenas para obter o código do *marketplace* cadastrado, para que então a consulta da combinação fosse realizada via API. Foi observado neste momento a necessidade de uma melhoria futura, para que este código fique visível no sistema.

Apenas um erro ocorreu durante os testes finais, relacionado ao caso de teste 2, “Manter vendedor e carregar taxonomia vendedor”. Ao salvar o cadastro, um erro 502 foi retornado ao usuário. O problema, entretanto, não estava no protótipo desenvolvido e sim na API cadastrada. A API utilizada, também de responsabilidade da empresa XY, estava fora de operação no momento do teste. Após reestabelecimento do serviço o cadastro foi salvo com sucesso.

Outra questão levantada durante os testes finais foi a duplicação de dados relativos a atributos. Ao testar a caso de teste 4, “Manter combinação de taxonomias”, o usuário alertou sobre uma possível duplicação dos dados de atributos para o

marketplace e vendedor cadastrados previamente. Contudo, analisando os dados persistidos, foi constato que não se tratava de uma duplicação, mas sim de inconsistências cadastrais nas taxonomias utilizadas pelo usuário nos testes. No cadastro de vendedores e *marketplaces*, após consultar as categorias disponíveis através da API cadastrada, são consultados e persistidos todos os atributos disponíveis para cada um dos níveis de categoria. As APIs utilizadas para teste, entretanto, retornam alguns atributos repetidos em níveis diferentes de categoria. Apesar destes atributos possuírem os mesmo nomes, os códigos são diferentes. Portando, o sistema salva todos os dados. Esta situação foi explicada ao usuário que verificou e confirmou as inconsistências cadastrais. Contudo, por se tratar de dados utilizados em ambiente de produção, os cadastros não foram alterados e os testes prosseguiram.

Para testar a consulta de combinações via API, caso de teste 5, foi utilizada uma plataforma chamada Postman[26]. Nesta plataforma foi criada uma requisição GET para a URL “http://localhost:8080/getCombinacoesPorMarketplace” informando o código do *marketplace* como parâmetro. O resultado dessa requisição foi então comparado com o modelo de JSON descrito no caso de uso correspondente. Além disso, os dados retornados foram comparados com o cadastro realizado pelo usuário.

O protótipo atendeu as expectativas do usuário. Alguns pontos de melhoria foram levantados por ambas às partes e serão descritos no capítulo seguinte.

[26] Plataforma *Postman* – disponível em: <https://www.getpostman.com/>. Acesso em 12/11/2019.

6. CONSIDERAÇÕES FINAIS

As vendas *online* possuem um papel fundamental no mercado econômico atual, e o desenvolvimento tecnológico nesse meio possibilita o desenvolvimento de novos modelos de negócio. Esse é o caso do *marketplace* eletrônico. Alguns dos maiores *e-commerces* do Brasil e do mundo já operam através do modelo de *marketplace*. Esse modelo funciona através de plataformas que buscam facilitar ao máximo as interações entre vendedores, que desejam vender seus produtos através do *e-commerce* relacionado ao *marketplace*, e a empresa detentora da plataforma.

Como pode ser observado nos *sites* das empresas que oferecem os serviços de *marketplace*, um dos primeiros passos para que um vendedor inicie sua operação na plataforma é a integração de produtos. A partir de uma análise bibliográfica, e também através das documentações fornecidas por essas plataformas, buscou-se identificar possíveis dificuldades encontradas nessas integrações. Apesar de poucos estudos terem foco nesse tipo de integração, principalmente no Brasil, foi possível identificar um ponto crítico para o sucesso desse processo - a combinação de taxonomias dos catálogos eletrônicos.

Cada vendedor envolvido na integração pode possuir uma taxonomia distinta no seu catálogo eletrônico, assim como o *e-commerce* ligado à plataforma de maketplace. Isso ocorre porque cada organização costuma ordenar seu catálogo de maneira diferente. Contudo, conforme pode ser observado nas documentações das plataformas, há uma exigência para que o vendedor informe em qual hierarquia da taxonomia do *e-commerce* relativo ao *marketplace* o seu produto deverá ser inserido. Desse modo, faz-se necessária a combinação das diferentes taxonomias.

Atuando como uma ferramenta auxiliar nas integrações de produtos entre vendedores e plataforma de *marketplace*, o presente estudo propôs o desenvolvimento de um protótipo para combinação de taxonomias. Através do protótipo, o vendedor tem a seu dispor a taxonomia do *marketplace* junto a sua própria, podendo, assim, criar uma combinação. Essa combinação precisa ser realizada uma única vez, pois uma API foi disponibilizada para que as relações criadas possam ser consultadas. Dessa forma, independente de quem for o responsável pelas integrações, pode fazer uso dessa ferramenta, incorporando-a ao seu processo de integração.

Para o desenvolvimento do protótipo, foi realizado um estudo sobre projetos

similares e tecnologias que poderiam ser utilizadas. Optou-se, então, pelo desenvolvimento utilizando a linguagem de programação Java, juntamente com o *framework Spring Boot*. Como estrutura arquitetural, serão utilizados o *MVC* e o padrão *Composite.*

No início do desenvolvimento, uma empresa responsável por uma plataforma de *marketplace*, referenciada com “empresa XY”, foi contatada. Esta empresa demonstrou interesse na solução proposta, uma vez que este é de fato um processo existente em suas integrações. À vista disso, a API de *marketplace* desta empresa foi disponibilizada para utilização, obviamente, com acessos controlados. Além disso, um analista ficou a disposição para a eventual realização de testes.

Ao final do desenvolvimento foi realizado um teste integrado de todo o sistema, tendo como base os casos de teste elaborados. Durante estes testes, o analista responsável identificou apenas um possível erro de impacto no sistema. Mediante análise, o mesmo foi identificado como um problema na estrutura taxonômica mantida pela própria empresa provedora da plataforma de *marketplace*. Confirmou-se então a dificuldade em manter a padronização dos dados referentes a estrutura cadastral de catálogos eletrônicos, conforme apontado por outros autores no referencial teórico deste estudo.

De forma geral, o protótipo desenvolvido atendeu aos objetivos propostos neste estudo e também às expectativas do usuário envolvido nos testes. Futuramente, este portótipo pode ser incorporado a diferentes processos de integração de dados envolvendo múltiplos *marketplaces* e vendedores, garantindo a integridade dos dados e agilidade nos processos.

6.1. TRABALHOS FUTUROS

Durante o desenvolvimento e testes do protótipo foram identificados alguns pontos de melhorias que podem contribuir com a qualidade e usabilidade do mesmo. No que diz respeito à codificação, visando atender as melhores práticas de desenvolvimento de software, deverá ser implementado o padrão TO para toda a aplicação. Além disso, deverá ser feira uma revisão do código garantindo que as hierarquias de acesso entre as camadas sejam respeitadas.

Apesar dos esforços para garantir uma melhor usabilidade do sistema, como por exemplo, o uso do componente *selectpicker*, o cadastro de uma nova combinação

ainda é um processo bastante manual. Uma possibilidade de melhoria futura seria a implementação de funcionalidades que realizem sugestões de combinação de catálogos através de inteligência artificial ou outras tecnologias.

Por fim, existe a necessidade de que mais opções para consulta de combinações via API sejam disponibilizadas aos vendedores. Para avaliar a real necessidade dessa melhoria, é necessário que os processos de integração de um ou mais vendedores sejam analisados tecnicamente.

7. REFERÊNCIAS

AANEN, Steven S.; VANDIC, Damir; FRASINCAR, Flavius. Automated product taxonomy mapping in an e-commerce environment. **Expert Systems With Applications**, [s.l.], v. 42, n. 3, p.1298-1313, fev. 2015. Disponível em: <https://www.researchgate.net/publication/266857831_Automated_product_taxonomy_mapping_in_an_e-commerce_environment>. Acesso em: 12 abr. 2019.

AGIRRE, Eneko; LACALLE, Oier López de; SOROA, Aitor. Random Walks for Knowledge – Based Word Sense Disambiguation. **Computational Linguistics**, [s.l.], v. 40, n. 1, p.57-84, mar. 2014. Disponível em: <https://www.mitpressjournals.org/doi/full/10.1162/COLI_a_00164>. Acesso em: 12 maio 2019.

BENATALLAH, Boualem et al. Towards semantic-driven, flexible and scalable framework for peering and querying e-catalog communities. **Information Systems**: Data: Creation, Management and Utilization, [s.l.], v. 31, n. 4-5, p.266-294, jun. 2006. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0306437905000347>. Acesso em: 27 maio 2019.

BERNERS-LEE, Tim; HENDLER, James; LASSILA, Ora. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. **Scientific American**, [s.l.], v. 1, n. 1, p.1-3, maio 2001. Disponível em: <https://www.researchgate.net/publication/225070375_The_Semantic_Web_A_New_Form_of_Web_Content_That_is_Meaningful_to_Computers_Will_Unleash_a_Revolution_of_New_Possibilities>. Acesso em: 29 maio 2019.

BORTOLI, Nathan da Silva de; RUFINO, Ricardo Ribeiro. **Conceito para o desenvolvimento web utilizando Spring Boot, Bootstrap e Angular JS**. SEINPAR, Paranavaí, PR, Brasil, 2016. Disponível em: <http://web.unipar.br/~seinpar/2016/publicacao/NATHAN%20S.%20DE%20BORTOLI.pdf>. Acesso em 15 jun 2019.

CHANG, Hsin Hsin; WONG, Kit Hong. Adoption of e-procurement and participation of e-marketplace on firm performance: Trust as a moderator. **Information & Management**, [s.l.], v. 47, n. 5-6, p.262-270, ago. 2010. Disponível em: <https://www.sciencedirect.com/science/article/abs/pii/S0378720610000492>. Acesso em: 05 maio 2019.

CHOUDARY, Sangeet Paul Choudary. **Platform Scale:** How an emerging business model helps startups build large empires with minimum investment. [s. L.]: Platform Thinking Labs, 2015. 336 p.

DA SILVA, Gislainy Laise; MENDES FILHO, Luiz. Perfil e frequência de uso das Online Travel Agency (OTA) por consumidores na rede hoteleira de Natal/RN. **Revista Acadêmica Observatório de Inovação do Turismo**, [S.l.], p. 22 - 44, jun. 2018. ISSN 1980-6965. Disponível em: <http://publicacoes.unigranrio.edu.br/index.php/raoit/article/view/4725>. Acesso em: 12 jun. 2019.

ELMASRI, Ramez; NAVATHE, Shamkant B..**Sistemas de Banco de Dados.** 4. ed. São Paulo: Pearson, 2005. 744 p.

e-marketplaces: A Fuzzy Logic based decision support system. **Information Sciences**, [s.l.], v. 278, p.267-284, set. 2014.

FENSEL, Dieter et al. **Product Data Integration in B2B E-Commerce**. IEEE Intelligent Systems, [S. I.], julho/agosto 2001. Disponível em:< https://ieeexplore. ieee.org/abstract/document/941358. Acesso em: 19 mar. 2019.

FIELDING, Roy T; TAYLOR, Richard N. **Principled Design of the ModernWeb Architecture**. University of California, Irvine, CA, 2000. Tradução própria. Disponível em:< https://www.ics.uci.edu/~fielding/pubs/webarch_icse2000.pdf>. Acesso em 15 jun 2019.

GALINARI, Rangel et al. E-commerce, tecnologias móveis e mídias sociais no Brasil. **BNDES Setorial**, Rio de Janeiro, v. 41, n. 1, p.135-180, mar. 2015. Semestral. Disponível em:<http://web.bndes.gov.br/bib/jspui/handle/1408/4281>. Acesso em: 24 maio 2019.

GAMMA, Erich et al. **Padrões de projeto:** soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000. 368 p.

GUISSONI, L. A.; OLIVEIRA, T. V. de.; TEIXEIRA, T. Um novo momento para o e-commerce. **GV-executivo**, v. 15, n. 1, janeiro-junho, 2016.

JIANG, Pingjun; BALASUBRAMANIAN, Siva K.. An empirical comparison of market efficiency: Electronic marketplaces vs. traditional retail formats. **Electronic Commerce Research And Applications**, [s.l.], v. 13, n. 2, p.98-109, mar. 2014. Elsevier BV. Disponível em: <https://www.sciencedirect.com/science/article/pii/S156742231300077X?via%3Dihub>. Acesso em: 05 maio 2019.

KOLOMVATSOS, Kostas; ANAGNOSTOPOULOS, Christos; HADJIEFTHYMIADES, Stathes. Sellers in e-marketplaces: A Fuzzy Logic based decision support system. Information Sciences, [s.l.], v. 278, p.267-284, set. 2014. Elsevier BV. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0020025514003430?via%3Dihu> . Acesso em: 10 maio 2019.

KOPPENHAGEN, Norbert et al. Designing a supply network artifact for data, process, and people integration. **Information Systems And E-business Management**, [s.l.], v. 14, n. 3, p.613-636, 19 nov. 2015. Springer Nature. Disponível em: <https://link.springer.com/article/10.1007/s10257-015-0296-1>. Acesso em: 10 maio 2019.

KUTERA, Robert; GRYNCEWICZ, Wieslawa. Web Oriented Architectural Styles for Integrating Service e-Marketplace Systems. **Proceedings Of The Seventh International Symposium On Business Modeling And Software Design**, [s.l.], v. 1, n. 1, p.72-80, jul. 2017. Disponível em: <https://www.researchgate.net/publication/318361001_Web_Oriented_Architectural_Styles_for_Integrating_Service_e-Marketplace_Systems>. Acesso em: 12 abr. 2019.

LARMAN, Craig. **Utilizando UML e Padrões:** Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Desenvolvimento Iterativo. 3. ed. Porto Alegre: Bookman,

2007. 696 p.

LESK, Michael. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. **Proceedings Of The 5th Annual International Conference On Systems Documentation - Sigdoc '86**, [s.l.], p.24-26, 1986. Disponível em: <https://dl.acm.org/citation.cfm?id=318728>. Acesso em: 25 maio 2019.

LIE, Yulius et al. A Comparison of Customer Preference Towards Two Different Types of E-Tailing Channel. In: International Conference on Information Management and Technology, 2., 2017, Yogyakarta. **Proceedings... .** Yogyakarta: Binus, 2017. p. 171 - 176. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8273532>. Acesso em: 20 maio 2019.

LIMA, Janssen dos Reis. **Consumindo a API do Zabbix com Python.** Rio de Janeiro: Brasport, 2016. 189 p.

LOPEZ-AREVALO, Ivan et al. Improving selection of synsets from WordNet for domain-specific word sense disambiguation. **Computer Speech & Language**, [s.l.], v. 41, n. 1, p.128-145, jan. 2017. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0885230816301899>. Acesso em: 01 maio 2019.

LUZ, Charlley. **Arquivologia 2.0:** a informação digital humana. Excertos de um arquivista 2.0 no mundo digital. Florianópolis: Bookess, 2010. 116 p. Disponível em: <https://www.researchgate.net/publication/302153587_Arquivologia_20_a_informacao_humana_digital>. Acesso em: 13 jun. 2019.

MEHRBOD, Ahmad et al. Matching heterogeneous e-catalogues in B2B marketplaces using vector space model. **International Journal Of Computer Integrated Manufacturing**, [s.l.], v. 30, n. 1, p.134-146, 4 nov. 2015. Disponível em: <https://www.tandfonline.com/doi/full/10.1080/0951192X.2015.1107915>. Acesso em: 01 maio 2019.

MEHRBOD, Ahmad; ZUTSHI, Aneesh; GRILO, António.
A Vector Space Model Approach for Searching and Matching Product E-Catalogues. In: Proceedings of the eighth international conference on management science and engineering management, 8., 2014, Lisboa. **Proceedings... .** [s.l.]: Springer, 2014. p. 833 - 842. Disponível em: <https://novaresearch.unl.pt/en/publications/a-vector-space-model-approach-for-searching-and-matching-product->. Acesso em: 5 maio 2019.

MENDONÇA, Herbert Garcia de. E-Commerce. **Revista Inovação, Projetos e Tecnologias**, [s.l.], v. 4, n. 2, p.240-251, 1 dez. 2016. Disponível em: <http://www6.uninove.br/ojs/journaliji/index.php/iptec/article/view/68>. Acesso em: 12 jun. 2019.

MONTANHEIRO, Lucas Souza; CARVALHO, Ana Maria Martins; RODRIGUES, Jackson Alves. Utilização de JSON Web Token na Autenticação de Usuários em APIs REST. In: **XIII Encontro anual de computação**, 8., 2017, Goiânia. [s. L.]: Enacomp, 2017. p. 186 - 193. Disponível em: <https://www.researchgate.net/profile/

Lucas_Montanheiro/publication/319205511_Utilizacao_de_JSON_Web_Token_na_Autenticacao_de_Usuarios_em_APIs_REST/links/599b14c4a6fdcc500349b4b6/Utilizacao-de-JSON-Web-Token-na-Autenticacao-de-Usuarios-em-APIs-REST.pdf>. Acesso em: 16 jun. 2019.

NOY, Natalya F.; MUSEN, Mark A.. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. **Ois@ijcai**, [s.l.], maio 2001. Disponível em: <https://www.researchgate.net/publication/242623376_Anchor-PROMPT_Using_Non Local_Context_for_Semantic_Matching>. Acesso em: 01 maio 2019.

OECD – Organization for Economic Co-Operation and Development. **OECD guide to measuring the information society 2011,** OECD Publishing, 2011. Disponível em: <https://www.oecd-ilibrary.org/science-and-technology/oecd-guide-to-measuring-the-information-society-2011_9789264113541-en>. Acesso em: 24 maio 2019.

PARK, Sangun; KIM, Wooju. Ontology Mapping Between Heterogeneous Product Taxonomies in an Electronic Commerce Environment. **International Journal Of Electronic Commerce**, [s.l.], v. 12, n. 2, p.69-87, dez. 2007. Disponível em: <https://www.researchgate.net/publication/240312285_Ontology_Mapping_Between _Heterogeneous_Product_Taxonomies_in_an_Electronic_Commerce_Environment> .Acesso em: 24 maio 2019.

PARKER, Geoffrey G.; VAN ALSTYNE, Marshall W.; CHOUDARY, Sangeet Paul. **Platform Revolution:** How Networked Markets Are Transforming the Economyand How to Make Them Work for You. Nova Iorque: W. W. Norton & Company, 2016. 336 p.

PRESSMAN, Roger S.; MAXIM, Bruce R.. **Engenharia de Software:** Uma Abordagem Profissional. 8. ed. Porto Alegre: Amgh, 2016. 968 p.

Recomendação para Lojas Virtuais. **Rct - Revista de Ciência e Tecnologia**, [s. L.], v. 1, n. 1, p.1-22, jan. 2015. Disponível em: <https://revista.ufrr.br/rct/article/view/2524>. Acesso em: 12 jun. 2019.

ROCHA, Rudimar Antunes da et al. A internet e a reinvenção do mundo dos negócios. In: XXII Encontro nacional de engenharia de produção, 22., 2002, Curitiba. **Proceedings.** Curitiba: Abepro, 2002. v. 1, p. 1 - 8. Disponível em: <https://www.researchgate.net/publication/237619883_A_INTERNET_E_A_REINVENCAO_DO_MUNDO_DOS_NEGOCIOS>. Acesso em: 12 jun. 2019.

SALAZAR, Frizzi Alejandra San Roman. **Um estudo sobre o papel de medidas de similaridade em visualização de coleções de documentos.** 2012. 101 f. Dissertação (Mestrado) - Curso de Ciências de Computação e Matemática Computacional, Universidade de São Paulo, São Carlos, 2012. Cap. 1. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-24012013-155903/pt-br.php>. Acesso em: 6 maio 2019.

SOARES, Edvan et al. Sistema Web para Mapeamento de Dados de Crimes Letais no Estado de Pernambuco. **Gestão.org**, [s.l.], v. 14, n. 2, p.288-295, 1 maio 2017. Disponível em: <https://periodicos.ufpe.br/revistas/gestaoorg/article/view/22554>.

Acesso em: 24 maio 2019.SOMMERVILLE, Ian. **Engenharia de Software.** 9. ed. São Paulo: Pearson Prentice Hall, 2011.

SOUSA, Eduardo P. de et al. Arquitetura de Aplicações Spring MVC: Uma Análise Baseada no Acoplamento Lógico. In: **XVII Workshop de computação aplicada**, 17., 2017, São José dos Campos. [s. L.]: Inep, 2017. p. 1 - 8. Disponível em: <http://vem2017.ufu.br/artigos/Sousa_et_al_2017.pdf>. Acesso em: 16 jun. 2019.

STANDING, Susan et al. Service value exchange in B2B electronic marketplaces. **Journal Of Business & Industrial Marketing**, [s.l.], v. 30, n. 6, p.723-/732, 6 jul. 2015.Disponível em: <https://www.emeraldinsight.com/doi/abs/10.1108/JBIM-05-2014-0112>. Acesso em: 01 maio 2019.

STANDING, Susan; STANDING, Craig. Service value exchange in B2B electronic marketplaces. **Journal Of Business & Industrial Marketing**, [s.l.], v. 30, n. 6, p.723-732, 6 jul. 2015. Emerald. Disponível em: <https://www.emeraldinsight.com/doi/abs/10.1108/JBIM-05-2014-0112>. Acesso em: 10 maio 2019.

STOLZ, Alex et al. PCS2OWL: A Generic Approach for Deriving Wb Ontologies from Product Classification Systems. **Lecture Notes In Computer Science**, [s.l.], v. 8465, n. 1, p.644-658, maio 2014. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-319-07443-6_43>. Acesso em: 12 maio 2019.

TURBAN, Efraim et al. Overview of Electronic Commerce. **Springer Texts In Business And Economics**, [s.l.], n. 1, p.3-49, 2015. Disponível em: <https://books.google.com.br/books?hl=ptBR&lr=&id=pGxyBgAAQBAJ&oi=fnd&pg=PR5&dq=Electronic+Commerce+A+Managerial+and+Social+Networks+Perspective&ots=lQFNRM19Rr&sig=16FVXWESaQxszjAZdppvZrT_Skk#v=onepage&q=Electronic%20Commerce%20A%20Managerial%20and%20Social%20Networks%20Perspective&f=false>. Acesso em: 25 maio 2019.

TURBAN, Efraim et al. Retailing in Electronic Commerce: Products and Services. **Springer Texts In Business And Economics**, [s.l.], n. 1, p.103-159, 2015. Disponível em: <https://books.google.com.br/books?hl=ptBR&Ir=&id=pGxyBgAAQBAJ&oi=fnd&pg=PR5&dq=Electronic+Commerce+A+Managerial+and+Social+Networks+Perspective&ots=lQFNRM19Rr&sig=16FVXWESaQxszjAZdppvZrT_Skk#v=onepage&q=Electronic%20Commerce%20A%20Managerial%20and%20Social%20Networks%20Perspective&f=false>. Acesso em: 25 maio 2019.

ZHENG, Qin et al. Computer System Integration and E-commerce. **Introduction To E-commerce**, [s.l.], p.336-372, 2009.

ZIEGLER, Patrick; DITTRICH, Klaus R.. Three Decades of Data Intecration — all Problems Solved? **Building The Information Society**, [s.l.], p.3-12, ago. 2004. Disponível em: <https://link.springer.com/content/pdf/10.1007%2Fb98986.pdf>. Acesso em: 01 maio 2019.

ALMEIDA, Mauricio B.; BAX, Marcello P.. Taxonomia para projetos de integração de fontes de dados baseados em ontologias. In: V ENCONTRO ENANCIB, 5., 2015, [s. L.]. **Proceedings**. [s.l.]: Ancib, 2015. v. 1, p. 1 - 20. Disponível em:

<http://enancib.ibict.br/index.php/enancib/venancib/paper/view/1912/1053>. Acesso em: 4 jun. 2019.

STEIMER, Isadora dos Santos Garrido; LUZ, Charlley dos Santos. Taxonomia para Comércio Eletrônico: diferentes perspectivas em front e back end. **Ciência da Informação em Revista**, Maceió, v. 2, n. 3, p.3-14, nov. 2015. Disponível em: <http://seer.ufal.br/index.php/cir/article/view/2186>. Acesso em: 01 maio 2019.

CAVALCANTE, Raphael da Silva; BRÄSCHER, Marisa. Taxonomias navegacionais em sítios de comércio eletrônico: critérios para avaliação. **Transinformação**, [s.l.], v. 26, n. 2, p.191-201, ago. 2014. Disponível em: <http://www.scielo.br/scielo.php?pid=S0103-37862014000200191&script=sci_abstract&tlng=pt>. Acesso em: 01 maio 2019.

SILVEIRA, Sidnei Renato; SILVEIRA, Sidnei Renato. Framework Genérico de Recomendação para Lojas Virtuais. **Rct - Revista de Ciência e Tecnologia**, [s. L.], v. 1, n. 1, p.1-22, jan. 2015. Disponível em: <https://revista.ufrr.br/rct/article/view/2524>. Acesso em: 12 jun. 2019.

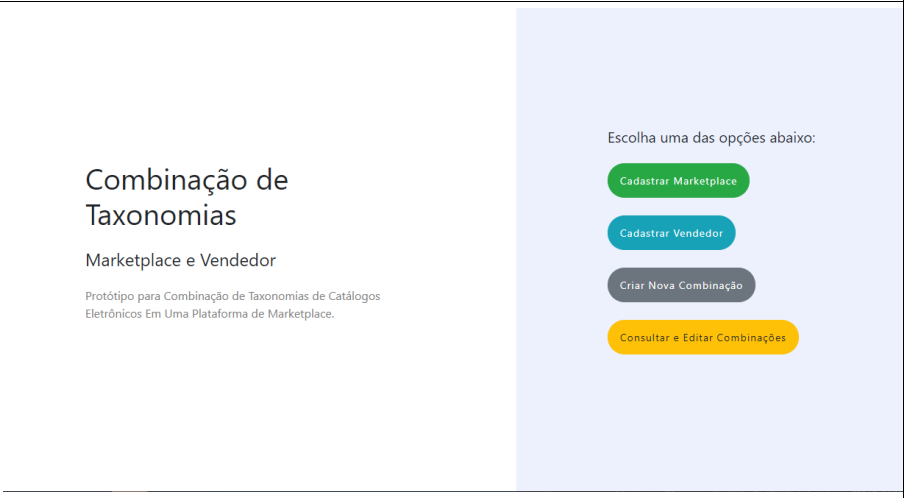
VITAL, Luciane Paula; CAFÉ, Ligia Maria Arruda. ONTOLOGIAS E TAXONOMIAS: DIFERENÇAS. **Perspectiva em Ciência da Informação**, Belo Horizonte, v. 2, n. 16, p.115-130, abr. 2011. Disponível em: <http://portaldeperiodicos.eci.ufmg.br/index.php/pci/article/view/200>. Acesso em: 01 maio 2019.

ENGHOLM JUNIOR, Hélio. **Análise e Design Orientados a Objetos.** São Paulo: Novatec, 2013. 376 p.

BRAGA, Pedro Henrique Cacique (Org.). **Teste de software.** São Paulo: Pearson, 2016. 139 p.

APÊNDICE A - Descrição dos casos de uso

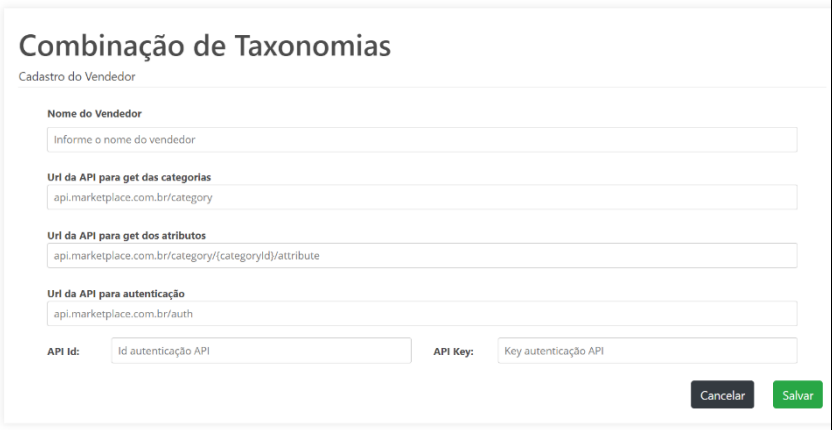
Quadro 1 – Gerenciar sistema

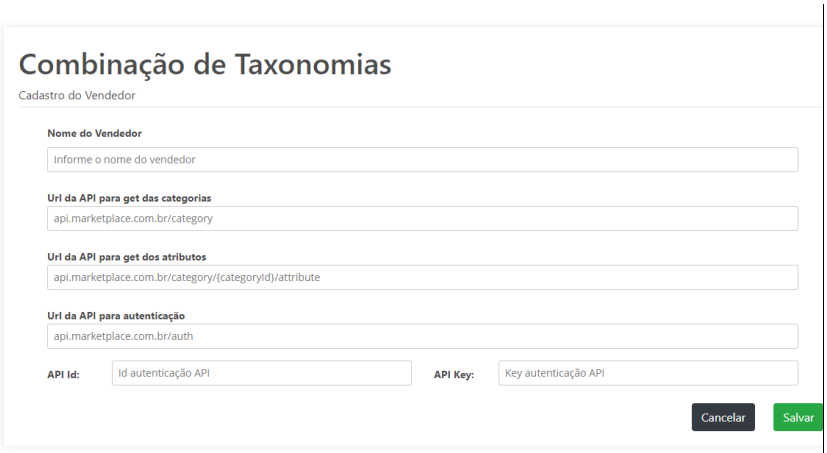
Caso de Uso:	Gerenciar sistema.
Objetivo:	Possibilitar o acesso às demais funcionalidades do sistema.
Descrição:	Tela de acesso inicial que possibilita o acesso às funcionalidades do sistema.
Ator:	Vendedor e plataforma de *marketplace*.
Requisitos:	RF011.
Pré-condição:	Não há.
Cenário Principal:	Acessar o sistema, navegar entre as opções existentes.
Cenários Alternativos:	Não há.
Protótipos:	

Fonte: elaborado pelo autor (2019).

Quadro 2 – Manter vendedor

Caso de Uso:	Manter vendedor.
Objetivo:	Cadastrar ou atualizar os dados dos vendedores no sistema.
Descrição:	Inicialmente será apresentada uma tela em que o usuário poderá pesquisar cadastros existentes. Será possível editar um cadastro específico após a busca.

	Ao clicar no botão “adicionar vendedor”, o usuário será direcionado a tela de cadastro. Este cadastro será composto pelos campos “nome”, “URL da API para get das categorias”, “URL da API para get dos atributos”, “URL da API para autenticação”, “API Id” e “API Key”. Além dos campos correspondentes às URL’s (Uniform Resource Locator) dos pontos de acesso à API do vendedor que fornecem a hierarquia de categorias e atributos que constituem a taxonomia, será necessário informar também a URL e demais dados para autenticação.
Ator:	Vendedor e API vendedor.
Requisitos:	RF002.
Pré-condição:	Não há.
Cenário Principal:	Acessar a tela de cadastro de vendedor, selecionar a opção “Adicionar Vendedor”, informar os dados necessários e salvar.
Cenários Alternativos:	Acessar a tela de cadastro de vendedor, buscar um vendedor existente, alterar as informações na tela de cadastro e salvar.
Protótipos:	

Fonte: elaborado pelo autor (2019).

Quadro 3 – Carregar taxonomia vendedor

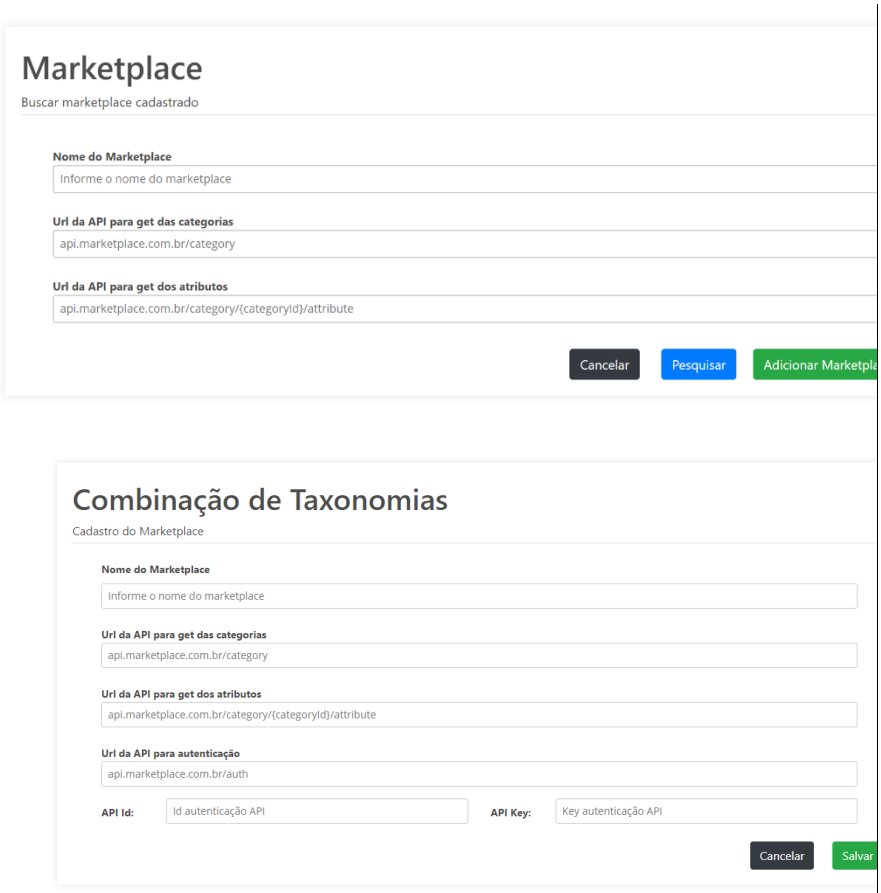
Caso de Uso:	Carregar taxonomia vendedor.
Objetivo:	Consumir a API do vendedor para que os dados da hierarquia de categorias e atributos sejam persistidos no banco de dados.
Descrição:	Com base nas informações salvas no cadastro, o sistema irá consumir a API do vendedor, persistindo no banco de dados a hierarquia de categorias retornada. Então, o ponto de acesso da API para consulta dos atributos por categoria será consultado recursivamente. A cada interação com a API, será informada uma categoria como parâmetro para que os atributos sejam retornados e persistidos em banco.
Ator:	Vendedor e API vendedor.
Requisitos:	RF005 e RF007.
Pré-condição:	Cadastro de vendedor realizado.
Cenário Principal:	Consumir a API do vendedor para que a hierarquia de categorias seja persistida no banco de dados. Consumir recursivamente a API do vendedor para que os atributos de cada uma das categorias sejam persistidos.

Cenários Alternativos:	Após a edição dos dados do vendedor, excluir a hierarquia de categorias e atributos para existente, consumir a API do vendedor para persistência dos dados da taxonomia.
Protótipos:	Esse caso de uso não possui modelos de telas.

Fonte: elaborado pelo autor (2019).

Quadro 4 – Manter plataforma de *marketplace*

Caso de Uso:	Manter plataforma de *marketplace.*
Objetivo:	Cadastrar ou atualizar os dados de *marketplaces* no sistema.
Descrição:	Inicialmente será apresentada uma tela em que o usuário poderá pesquisar cadastros existentes. Será possível editar um cadastro específico após a busca. Ao clicar no botão “adicionar *marketplace*”, o usuário será direcionado a tela de cadastro. Este cadastro será composto pelos campos “nome”, “URL da API para get das categorias”, “URL da API para get dos atributos”, “URL da API para autenticação”, “API Id” e “API Key”. Além dos campos correspondentes às URL’s (Uniform Resource Locator) dos pontos de acesso à API do *marketplace* que fornecem a hierarquia de categorias e atributos que constituem a taxonomia, será necessário informar também a URL e demais dados para autenticação.
Ator:	Plataforma de marketpalce e API da plataforma de *marketplace*.
Requisitos:	RF001.
Pré-condição:	Não há.
Cenário Principal:	Acessar a tela de cadastro de plataforma de *marketplace*, selecionar a opção “Adicionar *Marketplace*”, informar os dados necessários e salvar.
Cenários Alternativos:	Acessar a tela de cadastro de vendedor, buscar um *marketplace* existente, alterar as informações de cadastro e salvar.

Protótipos:	

Fonte: elaborado pelo autor (2019).

Quadro 5 – Carregar taxonomia *marketplace*

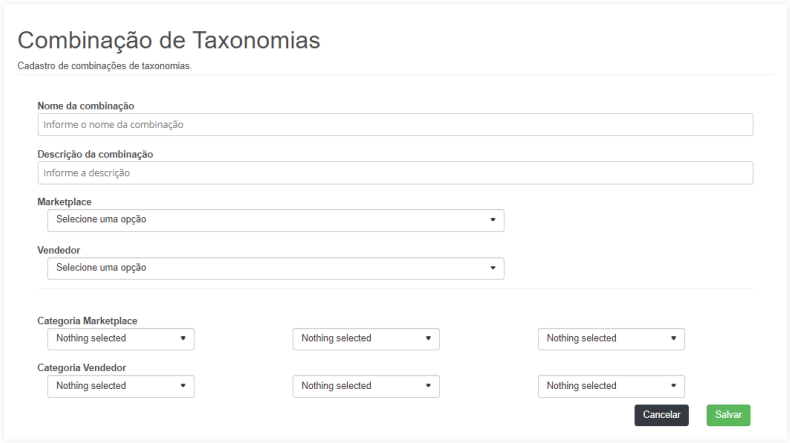
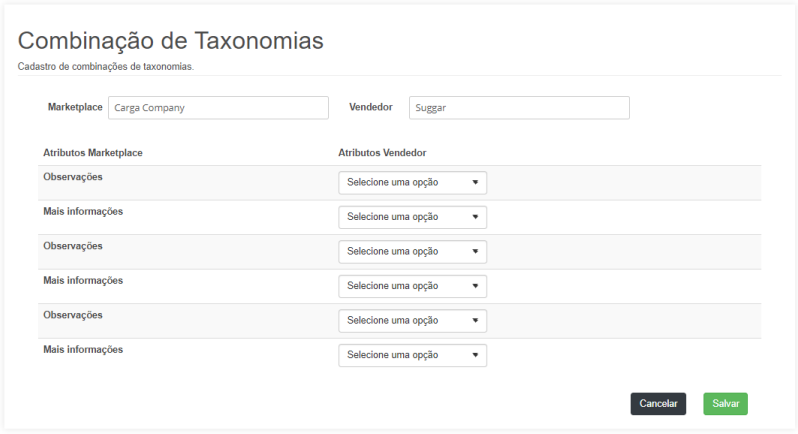
Caso de Uso:	Carregar taxonomia *marketplace..*
Objetivo:	Consumir a API do *marketplace* para que os dados da hierarquia de categorias e atributos sejam persistidos no banco de dados.
Descrição:	Com base nas informações salvas no cadastro, o sistema irá consumir a API do *marketplace*, persistindo no banco de dados a hierarquia de categorias retornada. Então, o ponto de acesso da API para consulta dos atributos por categoria será consultado recursivamente. A cada interação com a API, será informada uma categoria como parâmetro para que os atributos sejam retornados e persistidos em banco.
Ator:	Plataforma *Marketplace* e API Plataforma de *Marketplace.*

Requisitos:	RF004 e RF006
Pré-condição:	Cadastro de plataforma de *marketplace* realizado.
Cenário Principal:	Consumir a API do *marketplace* para que a hierarquia de categorias seja persistida no banco de dados. Consumir recursivamente a API do *marketplace* para que os atributos de cada uma das categorias sejam persistidos.
Cenários Alternativos:	Após a edição dos dados do vendedor, excluir a hierarquia de categorias e atributos para existente, consumir a API do *marketplace* para persistência dos dados da taxonomia.
Protótipos:	Esse caso de uso não possui modelos de telas relacionados.

Fonte: elaborado pelo autor (2019).

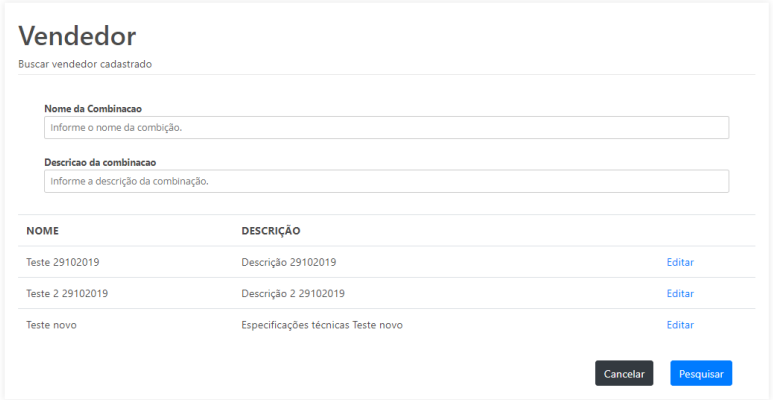
Quadro 6 – Manter combinação de taxonomias

Caso de Uso:	Manter combinação de taxonomias.
Objetivo:	Cadastro e atualização de combinações de taxonomias de um vendedor e de um *marketplace* previamente cadastrados.
Descrição:	Esse cadastro possuirá os campos “nome da combinação”, “descrição da combinação”, “*marketplace*” e “vendedor”. O usuário não irá interagir com o campo código, que deverá ter seu valor gerado automaticamente. Após selecionar um vendedor e uma plataforma de *marketplace*, as opções com as categorias para combinação ficarão acessíveis. Esses campos irão exibir os dados que foram persistidos após o consumo das APIs no momento do cadastro do vendedor e do *marketplace*, sempre respeitando as hierarquias definidas pelas empresas. Após o usuário finalizar a criação dos relacionamentos e clicar em salvar, o mesmo será direcionado para a tela de combinação de atributos. Para finalizar esse processo o usuário deverá realizar a combinação de atributos e clicar no botão salvar.

Ator:	Vendedor.
Requisitos:	RF008.
Pré-condição:	Cadastro de plataforma de *marketplace* e vendedor realizados.
Cenário Principal:	Acessar a tela de cadastro de combinação, informar todos os dados necessários, fazer a relação de categorias, salvar a relação de categorias, fazer a relação de atributos, salvar a combinação.
Cenários Alternativos:	Acessar o cadastro de uma combinação existente, atualizar os dados, salvar relação de categorias, fazer a relação de atributos, salvar a combinação.
Protótipos:	 

Fonte: elaborado pelo autor (2019).

Quadro 7 – Consultar combinações

Caso de Uso:	Consultar combinações.
Objetivo:	Possibilitar que o usuário busque, visualize e altere as combinações existentes.
Descrição:	Possibilitar que o usuário busque, visualize e altere as combinações existentes. Através desse caso de uso, o usuário terá acesso à tela de edição de combinações (caso de uso “Manter combinação de taxonomias”). Os campos disponíveis para a busca são nome e descrição da combinação.
Ator:	Vendedor e Plataforma de *Marketplace*.
Requisitos:	RF009.
Pré-condição:	Combinação de taxonomias realizadas.
Cenário Principal:	Acessar tela de consulta de combinações, preencher os campos, clicar no botão buscar, clicar no botão para edição da combinação desejada.
Cenários Alternativos:	Acessar tela de consulta de combinações, clicar no botão buscar sem preencher nenhum parâmetro, desta forma todas as combinações deverão ser retornadas. Clicar no botão para edição da combinação desejada.
Protótipos:	

Fonte: elaborado pelo autor (2019).

Quadro 8 – Consultar combinações via API

Caso de Uso:	Consultar combinações via API.
Objetivo:	Disponibilizar uma API REST para consulta das combinações existentes.
Descrição:	Será desenvolvida uma API para consulta das combinações realizadas. Deverá ser disponibilizada uma URL referente ao ponto de acesso à API, a qual retornará todas as combinações realizadas para determinado *marketplace*. Esse retorno será através de um arquivo do tipo JSON. Como parâmetros na URL, deverá ser informado o código do *marketplace*.
Ator:	API do protótipo de combinação de taxonomias.
Requisitos:	RF010
Pré-condição:	Combinação de taxonomias realizada.
Cenário Principal:	O vendedor consome a API desenvolvida informando um código de *marketplace*. Todos os dados referentes a combinações são retornados.
Cenários Alternativos:	Não existe fluxo alternativo para esse processo.
Protótipos:	O modelo de JSON:

```
[
   {
      "idCombinacao":"252339",
      "nome":"Teste 2 29102019",
      "descricao":"Descrição 2 29102019",
      "idMarketplace":"66",
      "idVendedor":"35",
      "idLinhaMarketplace":"6922",
      "nomeLinhaMarketplace":"Cards",
      "idFamiliaMarketplace":"6923",
      "nomeFamiliaMarketplace":"Entretenimento",
      "idGrupoMarketplace":"6924",
      "nomeGrupoMarktetplace":"Vídeo",
      "idLinhaVendedor":"5426",
      "nomeLinhaVendedor":"Eletroportáteis",
      "idFamiliaVendedor":"5516",
      "nomeFamiliaVendedor":"Aquecedor",
      "idGrupoVendedor":"5517",
      "nomeGrupoVendedor":"Aquecedor",
      "atributos":[
         {
            "idCombinacao":"252339",
            "idAtributoMarketplace":"3309",
            "nomeAtributoMarketplace":"Observações",
            "tipoAtributoMarketplace":"STRING",
            "idAtributoVendedor":"1215",
            "nomeAtributoVendedor":"Luz indicadora de aquecimento",
            "tipoAtributoVendedor":"BOOLEAN"
         },
         {
            "idCombinacao":"252339",
            "idAtributoMarketplace":"2318",
            "nomeAtributoMarketplace":"Mais informações",
            "tipoAtributoMarketplace":"STRING",
            "idAtributoVendedor":"795",
            "nomeAtributoVendedor":"Profundidade do nicho",
            "tipoAtributoVendedor":"STRING"
         },
         {
            "idCombinacao":"252339",
            "idAtributoMarketplace":"3309",
            "nomeAtributoMarketplace":"Observações",
            "tipoAtributoMarketplace":"STRING",
            "idAtributoVendedor":"1081",
            "nomeAtributoVendedor":"Profundidade",
            "tipoAtributoVendedor":"FLOAT"
         }
      ]
   }
]
```


Fonte: elaborado pelo autor (2019).

APÊNDICE B - Casos de teste

Quadro 1 – Caso de teste 1

Caso de teste	
ID:	1
Nome:	Gerenciar sistema
Ambiente:	Navegador Google Chrome.
Ator:	Vendedor e plataforma de *marketplace*.
Pré-condições:	Não há.
Procedimento:	1. Acessar o sistema através da URL. 2. Acessar a opção “Cadastrar Marketplace”. 3. Clicar no botão voltar. 4. Acessar a opção “Cadastrar Vendedor”. 5. Clicar no botão cancelar. 6. Acessar a opção “Criar Nova Combinação”. 7. Clicar no botão cancelar. 8. Acessar a opção “Consultar e Editar Combinações”.
Pós-condições:	Percorrer por todas as telas sem que nenhum erro seja retornado.

Fonte: elaborado pelo autor (2019).

Quadro 2 – Caso de teste 2

Caso de teste	
ID:	2
Nome:	Manter vendedor e carregar taxonomia vendedor.
Ambiente:	Navegador Google Chrome.
Ator:	Vendedor e API vendedor.
Pré-condições:	Não há.
Procedimento:	1. Acessar a opção “Cadastrar Vendedor”. 2. Clicar no botão “Adicionar Vendedor”. 3. Preencher os campos. 4. Clicar em “Salvar”. 5. Aguardar até que o cadastro seja salvo.

	6. Após o redirecionamento ao menu inicial, clicar em “Adicionar Vendedor”. 7. Informar o nome do vendedor cadastrado no campo “Nome do Vendedor” 8. Clicar em buscas. 9. Acessar o vendedor cadastrado previamente. 10. Editar alguma informação. 11. Clicar em salvar.
Pós-condições:	Cadastro de vendedor realizado com sucesso, busca realizada com sucesso e edição de cadastro realizada com sucesso.

Fonte: elaborado pelo autor (2019).

Quadro 3 – Caso de teste 3

Caso de teste	
ID:	3
Nome:	Manter plataforma de *marketplace* e carregar taxonomia *marketplace*.
Ambiente:	Navegador Google Chrome.
Ator:	Plataforma de *marketplace* e API plataforma de *marketplace*.
Pré-condições:	Não há.
Procedimento:	1. Acessar a opção “Cadastrar Marketplace”. 2. Clicar no botão “Adicionar Marketplace”. 3. Preencher os campos. 4. Clicar em “Salvar”. 5. Aguardar até que o cadastro seja salvo. 6. Após o redirecionamento ao menu inicial, clicar em “Adicionar Marketplace”. 7. Informar o nome do *marketplace* cadastrado no campo “Nome do Marketplace” 8. Clicar em buscar. 9. Acessar o *marketplace* cadastrado previamente.

	10. Editar alguma informação. 11. Clicar em salvar.
Pós-condições:	Cadastro de *marketplace* realizado com sucesso, busca realizada com sucesso e edição de cadastro realizada com sucesso.

Fonte: elaborado pelo autor (2019).

Quadro 4 – Caso de teste 4

Caso de teste	
ID:	4
Nome:	Manter combinação de taxonomias.
Ambiente:	Navegador Google Chrome.
Ator:	Vendedor.
Pré-condições:	Não há.
Procedimento:	1. Acessar a opção “Criar Nova Combinação”. 2. Preencher os campos, selecionando também as categorias da combinação. 3. Clicar em “Salvar”. 4. Fazer a combinação de atributos. 5. Clicar em salvar. 6. Após o redirecionamento ao menu inicial, clicar em “Editar e Consultar Combinações”. 7. Informar o nome ou descrição da combinação cadastrada. 8. Clicar em buscar. 9. Acessar a combinação cadastrada previamente. 10. Editar alguma informação da combinação de categorias. 11. Clicar em salvar. 12. Fazer a combinação de atributos. 13. Clicar em salvar.
Pós-condições:	Cadastro de combinação de categorias e atributos realizado com sucesso, busca realizada com sucesso e edição de cadastro realizada com sucesso.

Fonte: elaborado pelo autor (2019).

Quadro 5 – Caso de teste 5

Caso de teste	
ID:	5
Nome:	Consultar combinação de taxonomias via API.
Ambiente:	Plataforma Postman.
Ator:	API do protótipo para combinação de taxonomias.
Pré-condições:	Não há.
Procedimento:	1. Criar uma requisição get na plataforma Postman para a url “http://localhost:8080/getCombinacoesPorMarketplace”, informando o parâmetro idMarketplace com o id de *marketplace* cadastrado previamente. 2. Fazer a requisição à API.
Pós-condições:	JSON de retorno conforme o modelo do caso de uso “Consultar combinações via API”.

Fonte: elaborado pelo autor (2019).