SUPER PREÇO - COMPARADOR DE PREÇOS DE MERCADOS *ONLINE*

Eliel Oliveira Rodrigues, Fernanda Paterno

Orientador**: Roberto Luiz Debarba**

Curso Técnico em Informática - Ano/Semestre: 2023/3

Centro de Educação Profissional de Timbó (CEDUP) – Timbó, SC – Brasil

1 INTRODUÇÃO

Diariamente, consumidores percorrem os corredores de supermercados, mercearias e grandes atacados em busca de adquirir os itens essenciais para suas necessidades diárias. O mercado de consumo nacional voltado para o varejo sofreu fortes mudanças desde 2020, quando o comércio eletrônico ganhou força frente às fases de isolamento devido à pandemia e atingiu níveis surpreendentes de vendas *online*. Desde então, os hábitos dos consumidores se moldaram diante da disponibilidade do setor, e com isso, o mercado *online* se expandiu e novos formatos de vendas começaram a surgir. (ABRAS, 2022).

O e-commerce no Brasil registrou um crescimento de 2% em 2022, segundo relatório da *Nielsen|Ebit*, destacando os meses de janeiro, fevereiro e maio como os mais favoráveis. Os setores de mercado digital que mais se destacaram foram aqueles com menor *ticket* médio, como Alimentos e Bebidas, que teve um crescimento de 82,8%, seguido por Perfumaria e Cosméticos (22,5%), Saúde (16,9%), Bebês e Cia (12,3%) e Esporte e Lazer (8,4%). (*Nielsen|Ebit*, 2022)

Com o constante crescimento do e-commerce, os consumidores *online* têm à disposição uma ampla variedade de opções ao escolherem seus produtos durante o processo de compra. Pesquisar preços antes de fazer qualquer compra é um hábito que todo consumidor deveria ter. Afinal, o preço de um produto pode variar de uma loja para outra, e encontrar a melhor oferta sempre vai ser benéfico ao orçamento (SERASA, 2023).

Nesse cenário, emergem ferramentas que facilitam a comparação de preços entre diferentes produtos. Este tipo de recurso facilita a busca do consumidor por lojas que oferecem preços menores e estimula a competitividade de preços (BAKOS, 2001). Na elaboração de um site comparador de preços, a categorização e a associação de produtos surgem como desafios significativos durante o processo de coleta de dados. (CARVALHO, 2022).

> Em uma abordagem convencional, seria possível fazer essa comparação pelo *Global Trade Item Number (GTIN)*, um identificador global único muito utilizado em estabelecimentos físicos. A disponibilidade deste número em lojas *online* não é comum, tornando difícil utilizá-lo como referencial. Uma outra alternativa para identificar produtos, seria a utilização da *Stock Keeping Unit (SKU)*. Este valor é uma identificação única para cada item, dentro do estoque de um determinado estabelecimento. Um dos problemas com essa abordagem, é que cada supermercado implementa seu próprio valor, de maneira que, um mesmo produto dificilmente terá o mesmo *SKU* em sistemas diferentes. (CARVALHO, 2022).

Diante do exposto, este estudo propõe o desenvolvimento de um comparador de preços, utilizando a inteligência artificial por meio da *API GPT* da *OpenAI*, a mesma utilizada no *ChatGPT*. O principal objetivo é criar uma plataforma que simplifique a comparação de produtos de mercados para os consumidores. Para alcançar isso, será empregado o método de *web scraping*, uma técnica para coletar dados de produtos em mercados *online*, combinado com a inteligência artificial para o

relacionamento desses dados. Dessa forma, o usuário poderá, na interface no sistema, comparar e escolher em qual mercado deseja adquirir o produto. Os objetivos específicos deste trabalho são: (i) disponibilizar o preço de um mesmo produto em diferentes mercados cadastrados; (ii) Obter informações dos produtos pelo site de cada mercado, usando *web scraping*; (iii) Fazer a relação dos produtos entre os mercados utilizando a *API GPT* da *OpenAI*.

2 OBJETIVOS

2.1 OBJETIVO GERAL

Desenvolver um comparador de preços de produtos de mercados *online* utilizando API GPT da *OpenAI* para relacionar os produtos.

2.2 OBJETIVOS ESPECÍFICOS

- Obter informações dos produtos pelo site de cada mercado, usando web scraping.
- Fazer a relação dos produtos entre os mercados utilizando a API GPT da *OpenAI*.
- Disponibilizar o preço de um mesmo produto em diferentes mercados cadastrados.

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, será abordado a fundamentação teórica, que consiste em um conjunto de princípios e conceitos que fornecem a base sólida para a realização de um projeto de pesquisa. Esta seção é fundamental para compreender os fundamentos teóricos que sustentam o estudo e sua relevância no contexto da pesquisa.

3.1 INTELIGENCIA ARTIFICIAL

Inteligência Artificial ou IA, é uma área que envolve a ciência e engenharia na construção de máquinas inteligentes, com foco especial em programas de computador inteligentes (MCCARTHY, 2007). Para Dora Kaufman em seu livro Inteligência Artificial irá suplantar a inteligência humana? , a IA constitui um campo de conhecimento profundamente entrelaçado com aspectos como linguagem, raciocínio, aprendizagem e solução de problemas. (KAUFMAN, 2018).

A inteligência artificial teve seu primeiro marco no inovador artigo "*Computing Machinery and Intelligence*", publicado em 1950, onde o inglês matemático Alan Turing associou a computação com inteligência ao introduzir o famoso Teste de Turing. (TURING, 1950). Dessa forma, “a ideia básica do teste é que qualquer programa que se define como inteligente deve ser comparado com um humano, a única forma de inteligência conhecida” (JAMES, 1986). Nesta publicação, Turing foi o pioneiro ao introduzir o termo "inteligência artificial" pela primeira vez, estabelecendo-o como um conceito teórico e filosófico.

No ano de 1956, John McCarthy desempenhou um papel pioneiro ao introduzir pela primeira vez o termo "Inteligência Artificial" durante uma conferência histórica no *Dartmouth*

College. Este evento, considerado um marco seminal na história da Inteligência Artificial, reuniu visionários e acadêmicos, incluindo Marvin Minsky, Claude Shannon e Herbert Simon, para delinear as bases conceituais e os objetivos desta nova disciplina interdisciplinar. (MOURO, 2006).

De acordo com Russell e Norvig, autores do livro clássico "*Artificial Intelligence: A Modern Approach*", destacam duas ideias centrais sobre inteligência: a capacidade de aprender e a demonstração de um "comportamento inteligente" (RUSSELL; NORVIG, 2020).

Nesse contexto, Russell e Norvig (2020) delineiam quatro categorias de definições para inteligência artificial, vinculadas às concepções apresentadas: sistemas que imitam a ação humana, sistemas que replicam o pensamento humano, sistemas que adotam um pensamento racional e sistemas que executam ações de maneira racional.

Os algoritmos de inteligência artificial desempenham um papel significativo em nosso cotidiano. Uma parte essencial do sucesso da *Netflix* reside em seu sistema de personalização, no qual algoritmos analisam as preferências dos usuários e de grupos com gostos semelhantes. Com base nessas análises, o sistema sugere filmes e séries personalizadas. Utilizamos sistemas inteligentes para planejar itinerários no *Waze*, realizar pesquisas no *Google* e receber recomendações de músicas no *Spotify*, entre outras aplicações.(KAUFMAN, 2018)

Hoje, a tecnologia faz parte do nosso dia a dia, desde redes sociais até assistentes de voz como Siri e Alexa, e até mesmo no reconhecimento facial dos nossos celulares. Além disso, as inteligências artificiais estão ganhando popularidade, especialmente com ferramentas como *DALL-E 2* e *ChatGPT*, este último é relevante para o trabalho em questão. O *ChatGPT* é uma tecnologia avançada de processamento de linguagem que entende e gera texto de maneira inteligente, proporcionando uma comunicação interativa.

> "Um programa de inteligência artificial projetado para compreender e gerar texto de maneira natural. Foi desenvolvido pela *OpenA*I com base na arquitetura *GPT*-3.5. Em termos simples, sou um tipo de *chatbot* avançado que pode responder a uma ampla variedade de perguntas, participar de conversas e auxiliar com tarefas que envolvem texto." (*ChatGPT*, 2022).

O *ChatGPT*, desenvolvido pela *OpenAI*, utiliza técnicas avançadas de processamento de linguagem natural e se baseia na arquitetura *GPT*-3.5. *GPT* significa "*Generative Pre-trained Transformer*". Esse modelo utiliza aprendizado de máquina pré-treinado para gerar respostas em texto com base em entradas, como perguntas. Ele é treinado em grandes volumes de dados textuais para compreender padrões, estruturas gramaticais e conhecimento linguístico. O modelo é alimentado com a tarefa de prever palavras em um contexto, o que aprimora sua compreensão das relações entre as palavras em uma sequência.

O elemento fundamental dessa interação com o *ChatGPT* é o "*prompt*", que consiste na entrada de texto que o usuário fornece ao modelo para obter uma resposta. Ao iniciar uma conversa com o *ChatGPT*, o *prompt* orientará o modelo sobre o contexto e as informações relevantes para gerar uma resposta significativa. Esse processo reflete a capacidade do modelo de aprendizagem da linguagem e a estrutura da informação a partir do vasto conjunto de dados nos quais foi treinado.

Contudo, o futuro da IA é promissor, à medida que a pesquisa e o desenvolvimento continuam a avançar, possibilitando aplicações mais sofisticadas e úteis. Segundo dados do relatório *AI Readiness Index*, feito pela *Oxford Insights*, o Brasil está em 40º lugar entre 192 países no que diz respeito a soluções de Inteligência Artificial. A previsão é de que a implementação da tecnologia movimente cerca de US$15 trilhões, mundialmente até 2030. E o Brasil, até 2035, tem perspectiva de um aumento de um ponto percentual no PIB com o uso de Inteligência Artificial. A IA é uma ferramenta poderosa que está em constante evolução, e seu potencial é ilimitado.

3.2 WEB SCRAPING

Na *internet*, encontramos uma vasta quantidade de informações vindas de diversas fontes e apresentadas em diversos formatos. Nesse contexto, a necessidade de utilizar ferramentas e métodos para acessar e extrair dados de múltiplos sites ao mesmo tempo tem se tornado cada vez mais imperativa. O *Web Scraping* é uma técnica utilizada na extração de dados com o propósito de ser aplicada em um *site* ou análise, contribuindo para a tomada de decisões. Geralmente, esses dados são recuperados em formatos de arquivos específicos. O autor Ryan Mitchell apresenta uma definição abrangente de *Web Scraping* em seu livro "*Web Scraping* com *Python*", destacando:

> Teoricamente, *web scraping* é a prática de coletar dados por qualquer meio que não seja um programa interagindo com uma *API* (ou, obviamente, por um ser humano usando um navegador *web*). Isso é comumente feito escrevendo um programa automatizado que consulta um servidor *web*, requisita dados (em geral, na forma de *HTML* e de outros arquivos que compõem as páginas *web*) e então faz parte desses dados para extrair as informações necessárias. (MITCHELL, 2018).

Para Sirisuriya (2015) "[...] tal técnica é usada para transformar dados não estruturados na *Web* em dados estruturados que podem ser armazenados e analisados em um banco de dados local central ou planilha". Essas informações podem incluir texto, imagens, vídeos, dados estruturados e outros tipos de conteúdo disponíveis *online*. O objetivo principal do *web scraping* é coletar dados de sites para análise, pesquisa, armazenamento em bancos de dados ou qualquer outra finalidade.

> "Geralmente, a raspagem de dados da *Web* pode ser definida como o processo de extração e combinação de conteúdos de interesse da *Web* de forma sistemática. Nesse processo, um agente de software, também conhecido como robô *Web*, imita a interação de navegação entre os servidores *Web* e o ser humano em uma travessia convencional da *Web*. Passo a passo, o robô acessa quantos sites forem necessários, analisa seus conteúdos para encontrar e extrair dados de interesse e estrutura esses conteúdos conforme desejado." (LOURENÇO, 2013).

O *Web Crawling* é uma estratégia de indexação de conteúdo de sites que permite a exploração automatizada da *web*. *Web Scraping* envolve a extração de dados da internet, enquanto o *Web Crawling* se concentra em rastrear informações *online* de maneira automatizada, geralmente por meio de bots ou agentes de rastreamento.

A técnica "*crawling*" é extremamente empregada por motores de busca renomados, como o Google e o *Bing*. O *Web Crawling* possibilita uma varredura eficiente de páginas *web*, facilitando a indexação de informações relevantes para melhorar os resultados de busca e a experiência do usuário.

> "Os *web crawlers* (rastreadores da *web*) recebem esse nome porque rastreiam (crawl) a web. Em seu núcleo, encontra-se um elemento de recurso. Eles devem obter o conteúdo da página de um *URL*, analisar essa página em busca de outro *URL* e obter essa página, *ad infinitum*". (MITCHELL, 2018).

O *framework Scrapy* oferece uma ampla gama de funcionalidades que simplificam significativamente o processo de rastreamento. Essas capacidades incluem controle de navegação na *web*, bibliotecas de análise em *HTML*(*HyperText Markup Language*), representação de dados dedicados à filtragem e tratamento de dados (SANTANA, 2017).

Diversas linguagens de programação são utilizadas no âmbito do *Web Scraping*, sendo relevante destacar para os propósitos deste comparador, a linguagem de programação *Java*. A linguagem de programação *Java* é extensivamente utilizada para realizar tarefas de *Web Scraping*, e, nesse contexto, a biblioteca *Jsoup* se destaca como uma ferramenta notável.

O termo *Jsoup*, uma abreviação de "Analisador de *HTML Java*", delineia sua função central. Segundo Hedley (2023), o *Jsoup* é uma biblioteca em *Java* que fornece recursos avançados para a manipulação de *HTML*. Essa ferramenta versátil permite a restrição e manipulação de dados, a

análise da estrutura *HTML* de uma *URL (Uniform Resource Locator)* ou sequência de caracteres, bem como a manipulação de elementos *HTML* e texto.

Um dos pontos distintivos do *Jsoup* reside na sua capacidade de localizar elementos estruturais, possibilitando essa ação por meio da navegação pelo *DOM* (*Document Object Model*) e pela utilização de seletores *CSS (Cascading Style Sheets)*.

De acordo com a pesquisa "*Automation and the Future of Work*", conduzida em julho de 2020 pelo *IBM Institute for Business Value*, a automação com suporte de Inteligência Artificial (IA) tem o potencial de gerar bilhões de dólares em economia de mão de obra já em 2022. Nesse contexto, várias empresas estão recorrendo ao *web scraping* como uma prática fundamental para automatizar os processos de coleta de dados em larga escala, permitindo uma coleta eficiente e ágil de informações valiosas.

A técnica de *web scraping* oferece benefícios cruciais para as empresas, destacando-se na análise competitiva, economia de tempo e coleta de dados precisa. Diante do cenário em que o usuário médio da *internet* passa quase sete horas por dia *online*, a concorrência intensa no comércio virtual é evidente. O *web scraping* emerge como uma ferramenta essencial para a sobrevivência empresarial, proporcionando acesso a dados atualizados do mercado e dos concorrentes. Essa prática permite uma tomada de decisões mais acertada, crucial para enfrentar desafios e manter a competitividade em setores com grandes *players*, contribuindo para a eficácia das estratégias empresariais no ambiente digital. (VIVO, 2022).

3.3 *SOFTWARE* DE COMPARAÇÃO DE PREÇOS

Um *site* de comparação de preços é uma ferramenta de pesquisa que capacita os usuários a comparar os preços de produtos específicos em diversas lojas *online*, abrangendo diversos setores de mercado. Além de apresentar informações sobre os preços, algumas plataformas também fornecem detalhes cruciais sobre os produtos, como suas características, avaliações de consumidores e opções de entrega, entre outros. Comparadores de preço são *sites* que pesquisam o preço em diferentes lojas virtuais e os exibem em uma única página, permitindo que o cliente compare os preços facilmente. (MERLYN, 2017).

Um exemplo notável de sites comparadores de preços é a plataforma Buscapé, estabelecida em 1999 e amplamente reconhecida no cenário brasileiro. Acessível via navegador *web* ou aplicativo móvel, o Buscapé oferece uma vasta gama de produtos, desde eletrônicos, roupas, eletrodomésticos e até produtos de beleza. Embora não realize vendas diretas, desempenha um papel fundamental ao ajudar os consumidores a localizar as melhores ofertas disponíveis em várias lojas *online*.

O hábito de comparar preços antes da efetiva compra traz muitos benefícios, e um deles é a economia no bolso, pois é na hora de comparar, que o cliente visualiza qual o menor preço. No artigo “Saiba como um comparador de preços pode ajudar a você a economizar” do próprio *blog* do Serasa Crédito, é visto que:

> "Criar o hábito de fazer um comparativo de preços antes de adquirir qualquer produto, além de contribuir para que você encontre o melhor preço, evita que você realize uma compra por impulso. O ato de pesquisar os preços ajuda nisso, a diminuir a velocidade da compra, a aumentar a reflexão sobre a real necessidade daquele produto. É por isso que usar um comparador de preços só irá trazer benefícios a você."(Elaine Ortiz, 2022).

No campo de compras *online*, um estudo recente do Serviço de Proteção ao Crédito (SPC Brasil) e da Confederação Nacional de Dirigentes Lojistas (CNDL) revelou que "nove em cada 10 consumidores virtuais (93%) estão satisfeitos com as compras feitas pela internet" (SPC Brasil & CNDL, 2014).

No Brasil atual, aproximadamente 45,6% de sua população, o equivalente a cerca de 90

milhões de pessoas, possui acesso à internet. Ao compararmos os anos de 2000 e 2012, é notado que houve um aumento expressivo de aproximadamente 1.500% no número de usuários da internet no país. Os dados indicam um crescimento significativo no acesso à internet nos últimos anos, o que desempenhou um papel crucial na expansão do comércio eletrônico no Brasil. (TEIXEIRA, 2017).

Referente à cultura de consumo dos brasileiros que, durante o processo de compra, conforme destacado no artigo de 2016 do SPC Brasil intitulado "Nove em cada dez consumidores virtuais consultam a internet antes de realizar uma compra", é enfatizado que a internet se consolidou como uma fonte essencial de pesquisa para os consumidores brasileiros ao efetuarem compras, como evidenciado a seguir:

> "A *internet* tem se consolidado como fonte de pesquisa na hora do brasileiro ir às compras. De acordo com um estudo realizado pelo Serviço de Proteção ao Crédito (SPC Brasil) e pelo portal de educação financeira 'Meu Bolso Feliz', nove em cada dez consumidores brasileiros com acesso à internet (90%) assumem o hábito de fazer pesquisas *online* antes de realizar compras em lojas físicas. A pesquisa aponta que o comportamento é frequente em todas as faixas etárias, mas surge com mais força entre os indivíduos com idade entre 18 e 34 anos (93%)."(SPC Brasil ,2016).

Portanto, visto que a prática de comparar preços é benéfica para os consumidores, pois está contribuindo para uma experiência de compra mais eficaz, econômica e mutuamente satisfatória no cenário do comércio *online*. Além disso, é importante estar ciente de que a inflação é uma preocupação constante para os consumidores, pois pode afetar o poder de compra. "A inflação é o aumento de preços de forma sucessiva, o que leva à perda do poder de compra do consumidor", explica Reinaldo Domingos, presidente da Associação Brasileira de Profissionais de Educação Financeira (Abefin). Portanto, ao fazer comparações de preços, os consumidores também podem buscar proteger seu poder de compra e economizar em um cenário de preços em constante mudança.

3.4 PESQUISA DE MERCADO

Antes de qualquer ideia é extremamente importante validar a viabilidade de sucesso de qualquer projeto, desta forma fizemos a análise de mercado do segmento de mercado, aliás o *site* SUPER PREÇO utiliza dados de produtos de supermercados, nada mais válido que avaliar como está o segmento em questão.

Os segmentos de mercado desempenham um papel essencial na vida de todos, uma vez que é onde adquirimos os alimentos fundamentais para o nosso dia a dia - afinal, a comida é uma necessidade básica que não pode faltar! Nos últimos anos, o setor de mercado tem vivenciado desenvolvimentos significativos, incluindo a abertura de novas filiais em diversas cidades e a adoção de plataformas digitais que apresentam aos usuários seus produtos e ofertas. É interessante destacar as informações fornecidas pela Associação Brasileira de Supermercados que:

> "Em 2022, o setor supermercadista registrou um faturamento impressionante de R$ 695,7 bilhões, abrangendo todos os seus formatos e canais de distribuição, incluindo supermercados, hipermercados, atacarejos, mercados de regiões e e-commerce. , equivalente a 7,03% do Produto Interno Bruto (PIB) nacional."(ABRAS, 2022).

Visto que esse dados tão positivo é constituído pela variedade e diversidade dos canais de distribuição. Além disso, uma apresentação realizada pelo Departamento de Economia e Pesquisa, em agosto de 2023, revelou que os preços da cesta de alimentos básicos no Brasil diminuíram 2,07% em julho em comparação com o mês anterior. Isso resultou em um aumento notável no

consumo, com uma alta de 4,24% nos lares brasileiros. Sobre essa queda de preços, Marcio Milan, vice-presidente da ABRAS, comentou:

> "A queda expressiva nos preços dos alimentos para consumo no domicílio em julho sinaliza a necessidade de manter as medidas de combate à inflação, uma vez que mais da metade dos brasileiros buscam produtos com preços mais acessíveis ao compor suas cestas de compras." (ABRAS, 2023)

Além disso, a informação enfatiza a preocupação generalizada entre os brasileiros em relação aos preços acessíveis. O fato de que mais da metade da população está avançando em produtos com preços mais acessíveis ao compor suas cestas de compras destaca a importância de políticas econômicas que não apenas controlam a inflação, mas também garantem a acessibilidade e o poder de compra da população.

Em consonância com esse cenário positivo na indústria de alimentos em todo o Brasil, dados do portal de notícias da NSC Total revelam que as maiores redes de supermercados de Santa Catarina faturaram impressionantes R$ 33 bilhões e abriram 37 novas lojas em 2022, representando um aumento de mais de 25% em relação ao ano anterior (Pedro Machado, 2023). Junto disso,uma pesquisa mensal conduzida pela Associação Catarinense de Supermercados (Acats) traz notícias animadas sobre o consumo nos supermercados de Santa Catarina em julho deste ano. Comparando com o mesmo período de 2022, observamos um notável crescimento de 4,98%. Além disso, ao comparar julho com junho de 2023, o aumento foi de 3,17%. Esse impulso positivo também se reflete no acumulado do ano, com um avanço significativo de 7,06% de janeiro a julho, diminuindo uma tendência promissora no padrão de consumo das famílias catarinenses.

Portanto, no geral, o segmento do mercado de alimentos no Brasil e em Santa Catarina é positivo, com crescimento econômico, expansão do setor e preços mais acessíveis para os consumidores, o que contribui para o aumento do consumo.

3.5 RELATÓRIO DE PESQUISA

Foi realizada uma pesquisa quantitativa na sede CEDUP Timbó entre as datas de 20/10/2023 à 27/10/2023, com consumidores finais, onde foram obtidas 35 respostas. A maioria dos entrevistados são residentes da cidade de Timbó, sendo 65,7%. Na análise foi visto que a maioria dos entrevistados, 45,7%, possuem faixa etária de 19 a 23 anos e na segunda posição, 20% das respostas correspondem à faixa etária de 0 a 18 anos.

Abordando a frequência de idas ao mercado, observa-se que quase metade dos entrevistados, 45,7%, vão de duas à três vezes por semana e na segunda posição, 25,7% vão uma vez por semana. Foi observado que o supermercado *Koch* e Cooper foram os mais escolhidos para se fazer compras do mês, onde *Koch* em primeiro lugar com 31.4% e Cooper em segundo lugar com 25.7% dos entrevistados.

É perceptível que os preços baixos e a localização foram os fatores que mais influenciam na hora de ir às compras, sendo que 54.3% entrevistados preferem os preços baixos na escolha de mercado e que 25.7% escolhem o mercado que está mais perto de sua casa.

O questionário também apresentou um item para avaliar o hábito de comparar preços antes de efetivar a compra, visto que a maioria, sendo 40% dos entrevistados, sempre comparam os preços de produtos e em segundo lugar com 34,3% comparam os preços eventualmente. Desta forma conclui-se que a maioria das pessoas pesquisadas têm o hábito de comparar os preços, sendo de qualquer tipo de produto.

No questionário onde foi solicitado aos entrevistados a utilização de alguma ferramenta para comparar os preços de produtos de mercados, a grande maioria com 48,6% informou não utilizar nenhuma ferramenta, já na segundo posição, os entrevistados informaram utilizar as mídias sociais

sendo instagram, *facebook* e *whatsapp* para comparar. Por último, perguntou-se aos entrevistados se eles fariam uso de um site ou aplicativo que comparasse os preços de produtos de mercados da região. A grande maioria, com 91,4% dos entrevistados, afirmou que usaria o site, e apenas 8,6% afirmou que não usaria. Concluímos que houve um interesse real em um sistema de comparação de preços de mercados *online*.

3.6 CANVAS

Na figura 1 abaixo é ilustrado o projeto CANVAS, sendo um planejamento de um possível construção do negócio, onde é possível visualizar o comparador SUPER PREÇO como uma empresa e como seria seu desenvolvimento como um negócio real.

Figura 1: Projeto CANVAS

Fonte: Elaborado pelos Autores.

4 ESPECIFICAÇÃO

A especificação do sistema proposto é apresentada através de modelos e/ou diagramas que representam logicamente o comparador desenvolvido, usando ferramentas de desenvolvimento e análise. Apenas os diagramas mais relevantes ou mais complexos são exibidos, sempre com descrições textuais.

O site proposto neste projeto oferece um comparador único onde os usuários podem acessar uma ampla variedade de produtos de diferentes mercados, tudo em um só lugar, eliminando a necessidade de visitar vários *sites*. Além disso, o site permite que os usuários selecionem um produto específico e o comparem com produtos similares disponíveis em diferentes mercados.

Para tornar essa comparação de produtos mais eficiente e precisa, o site utiliza a tecnologia de inteligência artificial, especificamente o modelo *GPT*. Este modelo é capaz de entender e comparar informações.

Os requisitos funcionais, apresentados na tabela 1, para o projeto foram elaborados com essas considerações em mente. Eles foram projetados para garantir que o site possa oferecer aos usuários uma experiência de comparação conveniente, permitindo-lhes acessar uma variedade de produtos de diferentes mercados em um só lugar.

Tabela 1 – Requisitos Funcionais (RF)

Requisito Funcional
RF01 - O site deve ser capaz de listar produtos de diferentes mercados para os usuários.
RF02 - Os usuários devem ser capazes de selecionar um produto da lista para comparação.
RF03 - O sistema deve usar o modelo GPT para comparar o produto selecionado com produtos similares em diferentes mercados.

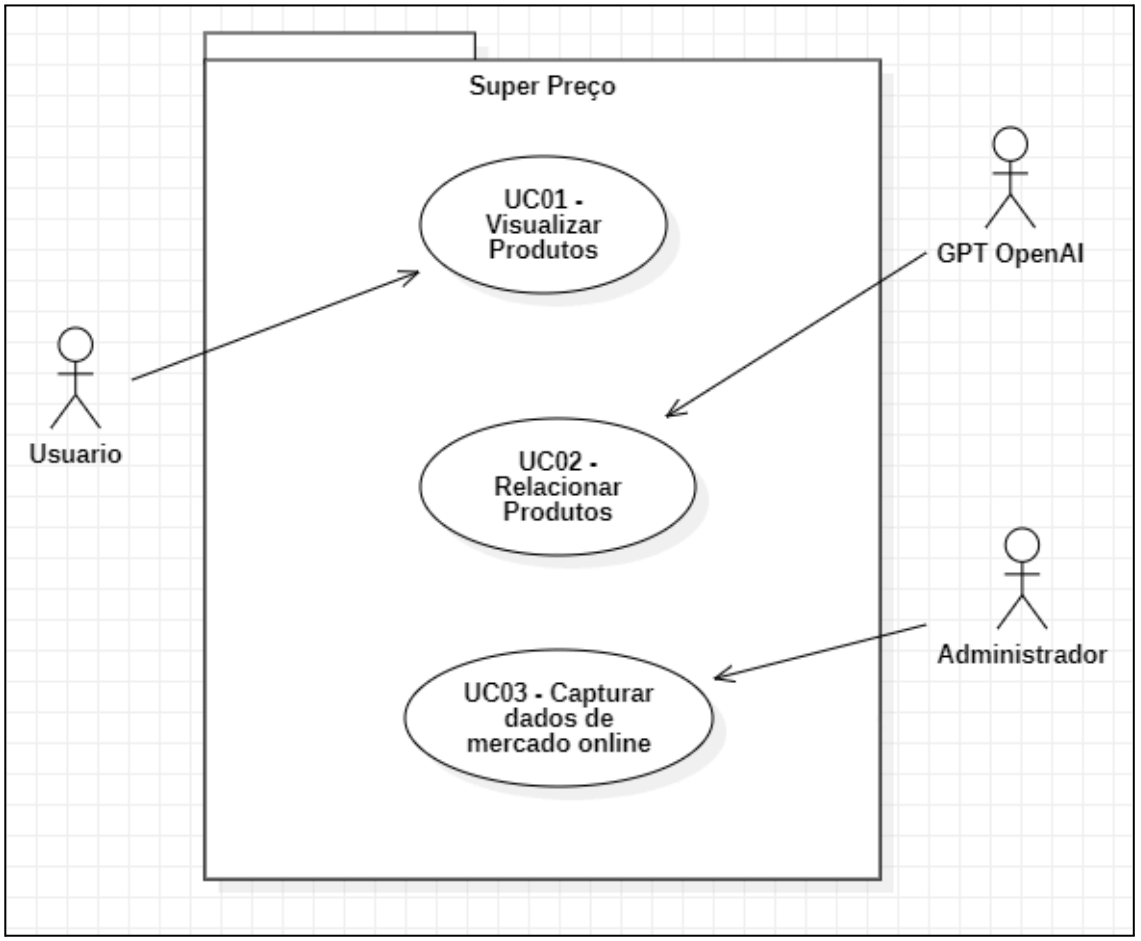
Fonte: Elaborado pelos Autores.

4.1 DIAGRAMA DE *USE-CASE*

Após a identificação e a análise dos requisitos, torna-se fundamental desenvolver uma representação visual da interação entre os diversos atores e o projeto em questão. Isso é realizado através da diagramação e modelagem das ações envolvidas, culminando na criação do Diagrama de Casos de Uso.

Na Figura 2 é apresentado o caso de uso UC01 onde o usuário pode visualizar os produtos disponíveis no *site* e compará-los, além de poder selecionar os produtos que deseja comparar e visualizar as informações relevantes sobre. No UC02 a *API GPT* é responsável por receber os produtos selecionados pelo usuário e fazer o relacionamento entre eles. A *API GPT* utiliza técnicas de processamento de linguagem natural para identificar as características dos produtos e determinar a similaridade entre eles. E por último no UC03 o administrador dispara uma requisição para capturar os produtos dos *sites* dos mercados e salvá-los no banco de dados. O administrador pode especificar os *sites* dos mercados que deseja capturar os produtos e definir as informações relevantes que devem ser salvas no banco de dados. Segue abaixo a Figura 2 do diagrama de Caso de Uso:

Figura 2: Diagrama de Use Case

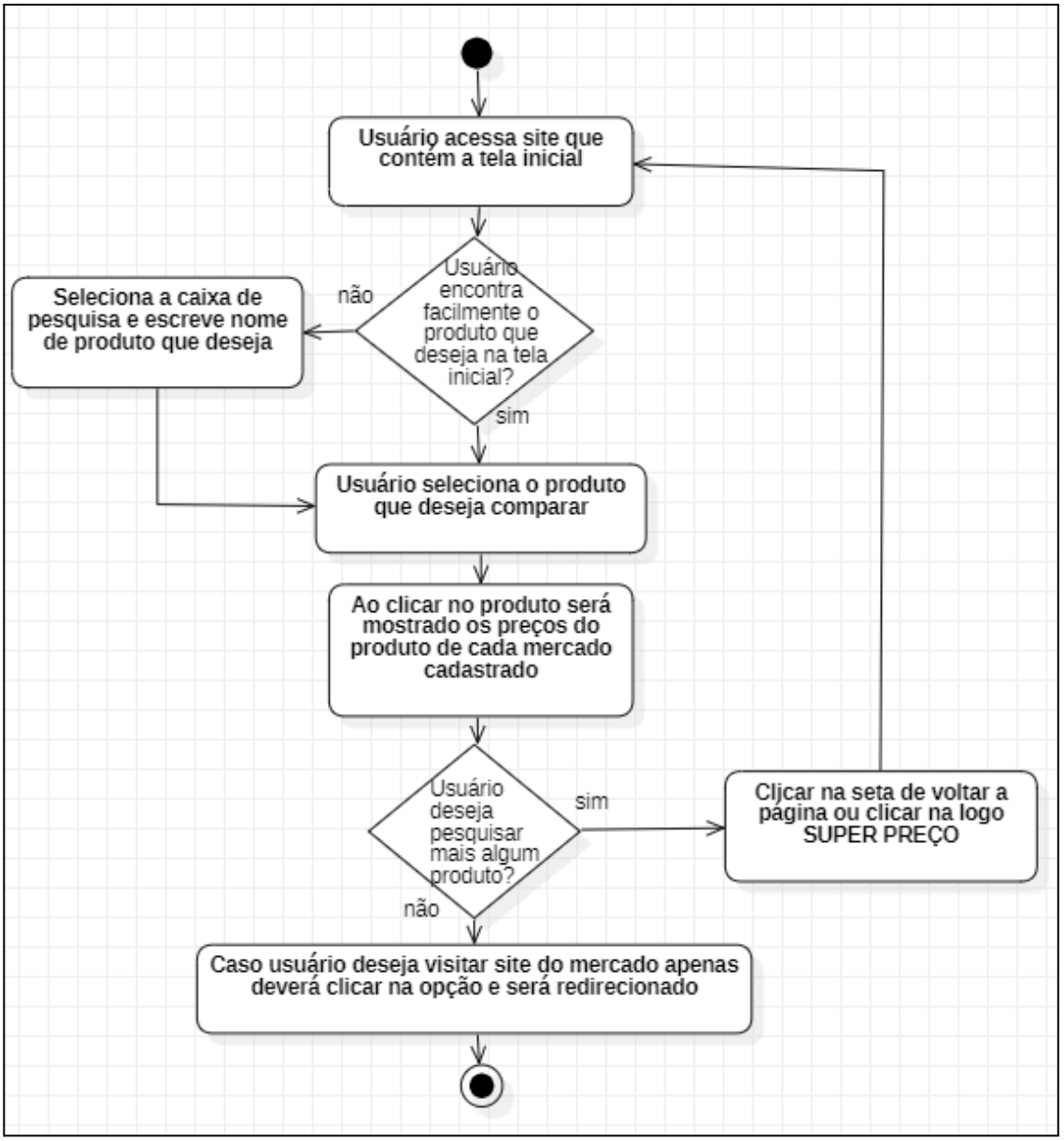


Fonte: Elaborado pelos Autores.

4.2 DIAGRAMA DE ATIVIDADE

A Figura 3 apresenta um diagrama de atividade, que mostra o fluxo de ações que podem ser realizadas pelo usuário ao acessar o site. O planejamento do *site* e das possíveis ações, foi pensado para ser objetivo na ação proposta, onde logo ao acessar o *site*, uma lista de produtos vai estar disponível e para comparar, basta clicar no botão com o mesmo nome onde vai ser redirecionado para uma próxima página, com informações do produto em destaque e os mercados onde está disponível. Para facilitar a busca por produtos específicos, o *site* conta com uma barra de pesquisa que fornece sugestões de autocompletar à medida que o usuário digita um termo desejado.

Figura 3: Diagrama de Atividade



Fonte: Elaborado pelos Autores.

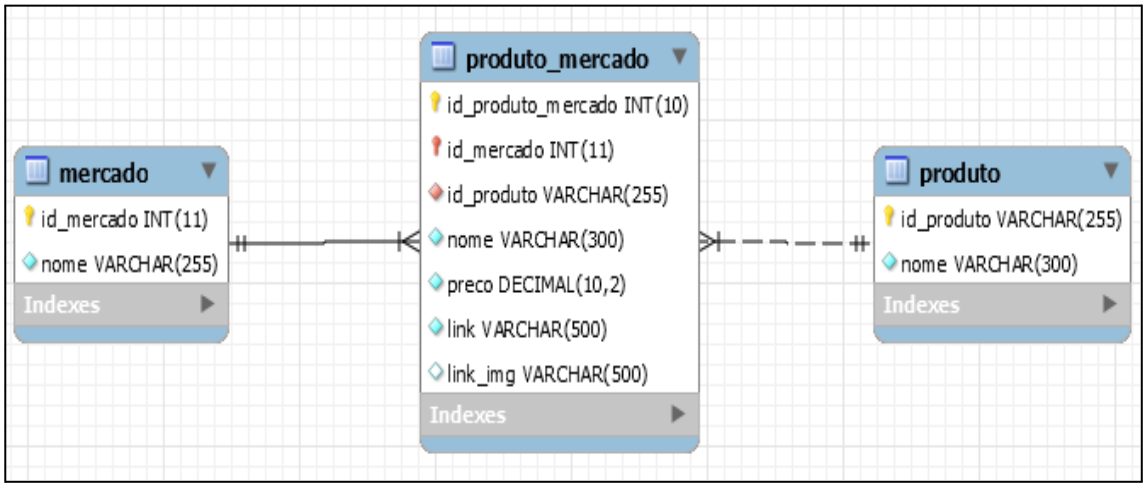
4.3 BANCO DE DADOS

No contexto do banco de dados *super_preco*, temos três entidades principais que armazenam informações sobre mercados e produtos, conforme ilustrado na Figura 4.

A entidade mercado cataloga os mercados de origem dos produtos. Cada registro único é identificado por uma chave primária *id_mercado* e inclui o nome do mercado. A entidade produto agrupa produtos semelhantes, independentemente de sua origem. Cada produto é unicamente identificado por uma chave primária *id_produto* e o nome do produto é armazenado.

A entidade *produto_mercado* estabelece a conexão entre as entidades mercado e produto. Cada registro representa um produto específico de um mercado específico, permitindo rastreabilidade. As informações armazenadas incluem *id_produto_mercado* (a chave primária), *id_mercado* e *id_produto* (ambas chaves estrangeiras que referenciam as entidades mercado e produto, respectivamente), nome (o nome do produto no mercado específico), preco (o preço do produto no mercado específico), link (um link para o produto no site do mercado) e link_img (um link para uma imagem do produto).

Figura 4: Banco de Dados



Fonte: Elaborado pelos Autores.

5 DESENVOLVIMENTO

Este capítulo descreve o desenvolvimento do comparador como a escolha de mercados com loja *online*, realização da *PoC(Proof of Concept)* com o *ChatGPT,* análise da estrutura de cada *site* utilizado para elaboração do algoritmo responsável pelo *scraping* dos dados, modelagem do banco de dados, preparação dos dados para serem consumidos pela *API GPT*-4 da *openAI*, para em seguida disponibilizar os resultados no *frontend.*

Também é abordado sobre limitações e características vistas ao se trabalhar com inteligência artificial*,* como o modelo *GPT* da *OpenAI*. Por fim, são pautadas as etapas e ferramentas utilizadas para a elaboração e resultados alcançados.

5.1 *PoC*

Para validar o desempenho do *GPT* como recurso para realizar o relacionamento de produtos iguais, foi realizada uma *PoC* em setembro, onde foram pesquisados dois mercados com lojas *online* da região de Santa Catarina: *Cooper* e *Koch*. Esses mercados foram escolhidos por serem supermercados que disponibilizam os seus produtos e preços na internet, facilitando o *web scraping* dos dados. Foram coletados manualmente 126 produtos, sendo 67 produtos do supermercado *Cooper* e 59 produtos da o supermercado *Koch*, que pertenciam às mesmas categorias e tinham características semelhantes, como nome, descrição e preço. Como pode ser visto na tabela 2 , os dados foram relacionados em uma planilha, com uma coluna para mercado e seus produtos, onde produtos iguais foram posicionados na mesma linha. A descrição/nome do produto foi o único atributo utilizado para a *PoC*, pois foi considerado o mais relevante para a comparação dos produtos, já que contém as informações sobre as características, a marca e o modelo dos mesmos, que pode ser observada na tabela do *Anexo A*.

Com os produtos organizados, foi utilizado o *ChatGPT* para testar um conjunto de instruções a fim de obter um resultado satisfatório. O *ChatGPT* é um *chatbot* movido a inteligência artificial, baseado no modelo *GPT*-3.5 Turbo da *OpenAI*, que é capaz de gerar textos coerentes e relevantes a partir de um *prompt*. O *prompt* é uma frase que define o objetivo e o contexto da conversa. Após várias tentativas, chegamos em um *prompt* que alcançou 70% de acerto na relação de produtos iguais, sendo: “Faça uma tabela com uma coluna para cada nome de mercado. Analisando os itens a seguir e levando em consideração a marca, sessão, sabor e massa, coloque em cada linha um produto do mercado 1 e o seu correspondente no outro mercado.” mais os produtos da tabela 2, divididos em uma lista para cada mercado. Observamos que incluir o “Análise todos os produtos” e o que deveria ser levado em consideração para comparação fazia total diferença na qualidade da resposta.

Apesar do resultado satisfatório, um artigo feito em 2023, denominado “*How Is ChatGPT's Behavior Changing over Time?*”, mostrou que os modelos *GPT* podem apresentar uma grande variação de desempenho e comportamento ao longo do tempo, dependendo da capacidade de seguir as instruções do usuário (Stanford e Berkeley, 2023).

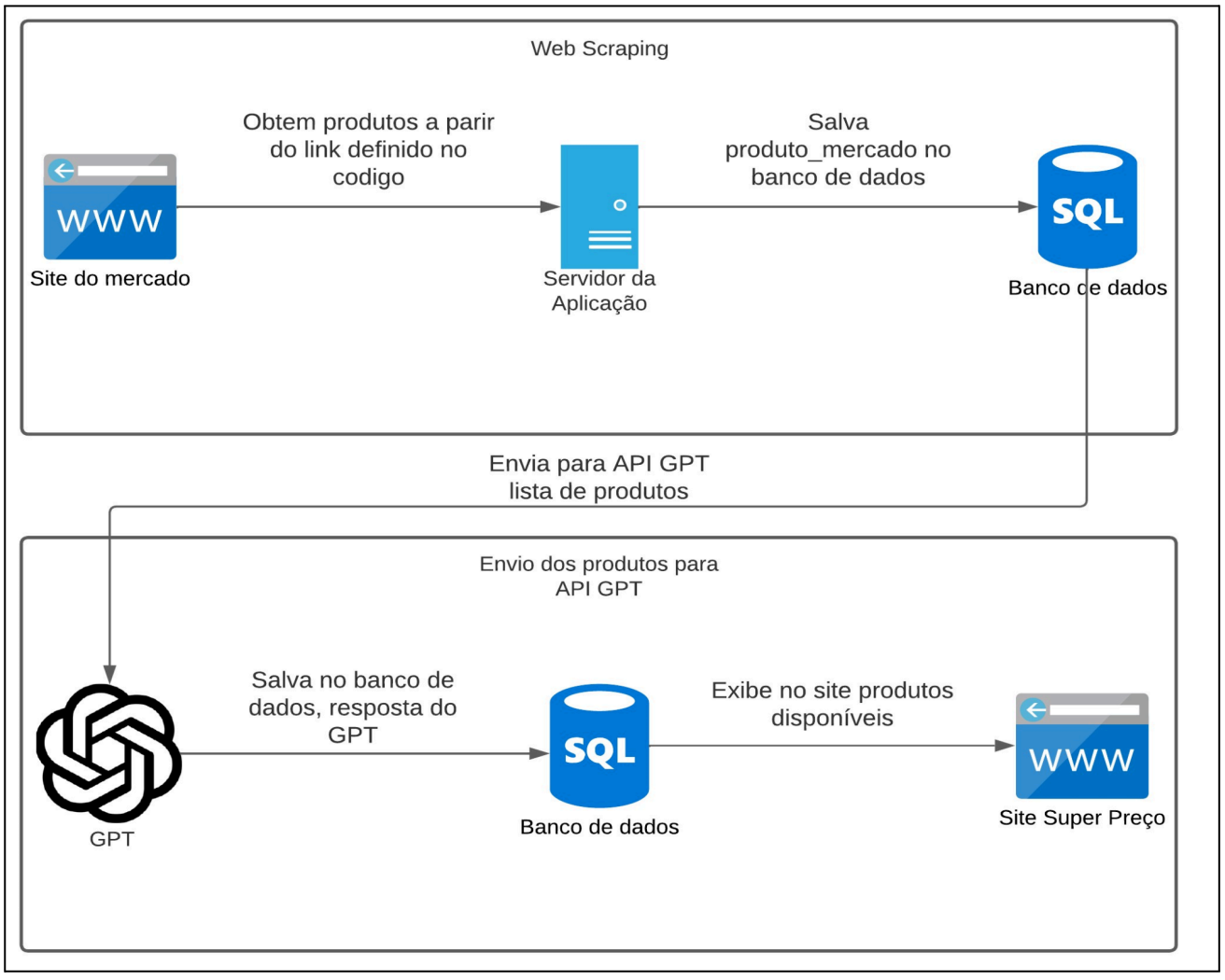
Por exemplo, o *GPT*-4 (março de 2023) era razoável em identificar números primos *versus* compostos (84% de acurácia), mas o mesmo modelo (junho de 2023) era ruim nessas mesmas questões (51% de acurácia). Isso se explica parcialmente por uma queda na capacidade do *GPT*-4 de seguir o encadeamento de pensamento solicitado pelo usuário. Curiosamente, o *GPT*-3.5 melhorou muito nessa tarefa em junho. Os autores sugerem que mudanças na capacidade desses modelos de seguir instruções do usuário podem ser um fator comum por trás dos desvios de comportamento em diferentes tarefas. Eles criaram um conjunto de instruções independentes de tarefas e avaliaram as versões de março e junho do *GPT*-4 e do *GPT*-3.5 nele. No geral, eles observaram uma grande diminuição da capacidade do *GPT*-4 de seguir muitas instruções.

O *GPT*-4 em março era geralmente bom em seguir as instruções do usuário (por exemplo, gerar respostas seguindo formatos especificados), mas em junho ele falhou em seguir a maioria dessas instruções simples. Os resultados dos autores mostram que o comportamento do 'mesmo' serviço pode mudar substancialmente em um período relativamente curto de tempo, destacando a necessidade de monitorar continuamente esses modelos.

5.2 *Backend*

Na aplicação, construída seguindo o *design* de projeto *MVC (Model-View-Controller)*, o *backend* é responsável por diversas tarefas. Utilizando o *Spring Boot*, com o auxílio da biblioteca *Spring Web* e *Jsoup*, o *backend* realiza o *web scraping* dos produtos e envia esses dados para a *API GPT*. A *API GPT*, por sua vez, retorna uma relação de produtos que o *backend* processa, salva no banco de dados, onde o *frontend* acessa e exibe os produtos consumindo a *API Rest* '/produto/produtos/'. A figura 5 ilustra esse fluxo de processamento de dados, destacando a interação entre os componentes e a sequência de operações realizadas.

Figura 5: Fluxo de Processamento dos Dados

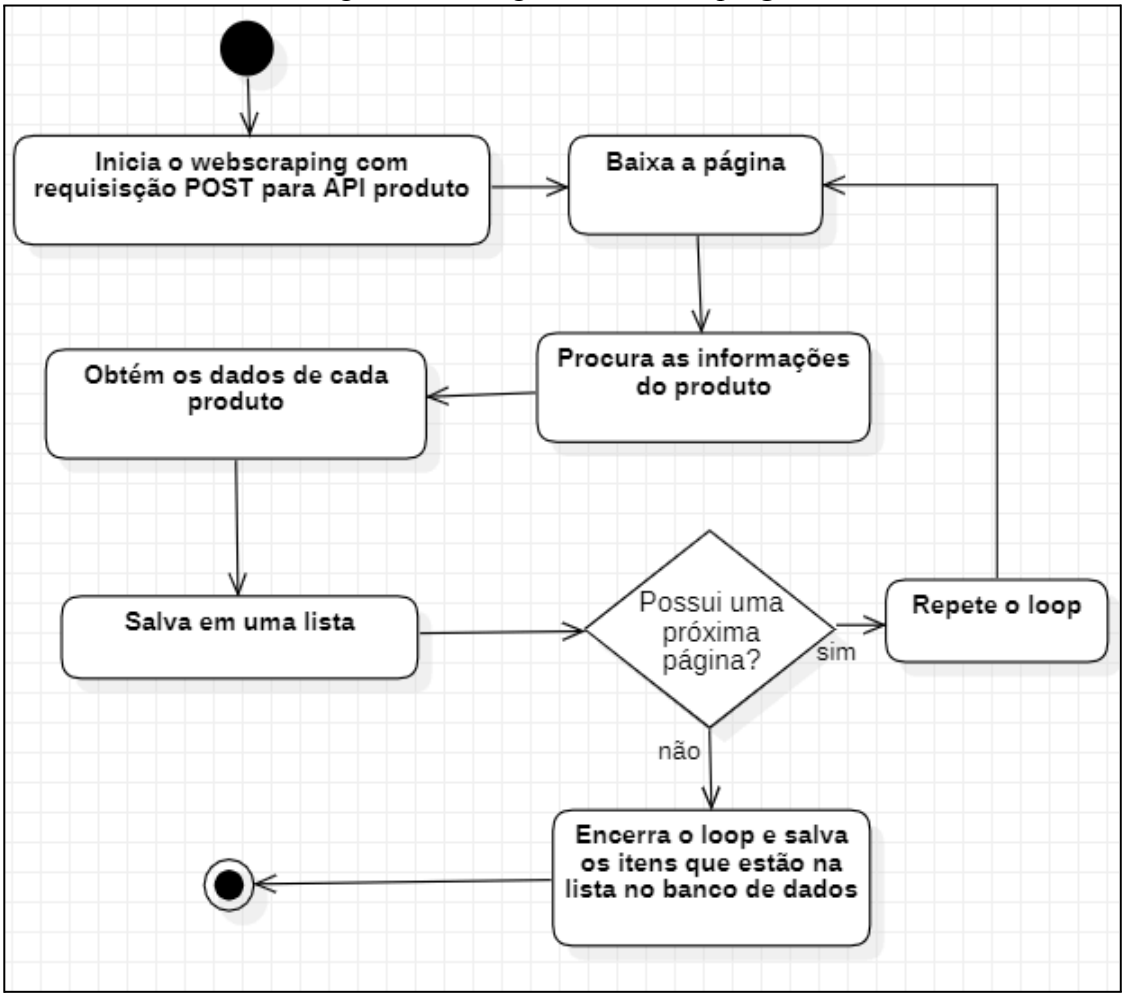


Fonte: Elaborado pelos Autores.

5.2.1 *Web Scraping*

Utilizando a biblioteca *Jsoup*, o algoritmo de *web scrapin*g é iniciado com uma *URL* base do *site* selecionado a partir de uma requisição *POST* para *API Rest* /produto/scraping. A Figura 6 ilustra etapas desse processo.

Figura 6: Fluxograma *Web Scraping*



Fonte: Elaborado pelos Autores.

A partir desta *URL*, o algoritmo itera pelas *URL*s das categorias de produtos previamente descritas no código. Para cada categoria, uma conexão é estabelecida com a *URL* correspondente e o documento *HTML* da página é obtido.

O próximo passo é a busca pelos seletores *CSS* que contém as informações do produto para o enviar ao *GPT*, para a comparação dos preços, exibição dos produtos no site e redirecionamento para sua página correspondente no site de cada mercado. Foi escolhido nome/descrição, preço, *URL* da página de detalhes de cada produto e *URL* da imagem de cada produto. Neste estágio, cada produto recebe um *id_mercado*, que varia para cada mercado, e um *id_produto*, que neste caso é sempre zero.

O algoritmo então procura na página por algum elemento *CSS* que represente um botão da próxima página. Se tal elemento for encontrado, a *URL* da categoria é atualizada e o *loop* é repetido. Este processo continua até que a última página da categoria seja alcançada.

O Quadro 1, apresenta a programação do processo de *scraping*, uma vez que os dados são obtidos, eles são armazenados em uma lista, cada classe de *web scraping, cooperScrappe*r e *kochScrapper*, possui seu próprio método para realizar o *scraping* para um determinado mercado, retornando uma lista de *Produto_MercadoEntity* . Na classe *Produto_MercadoController*, instâncias das duas classes de *web scraping* são criadas e as duas listas são combinadas em uma. Utilizando o padrão de *design DAO (Data Access Objec)*, o conjunto de classes e interfaces *JDBC* (Java *Database Connectivity*) e um laço de repetição, essa lista de produtos é então salva no banco de dados. Por último, retornando o status *HTTP 201 created,* indicando que o processo foi realizado, a figura 7 mostra o formato como os dados obtidos vão ser salvos no banco de dados .

Quadro 1 - Método de *scraping*

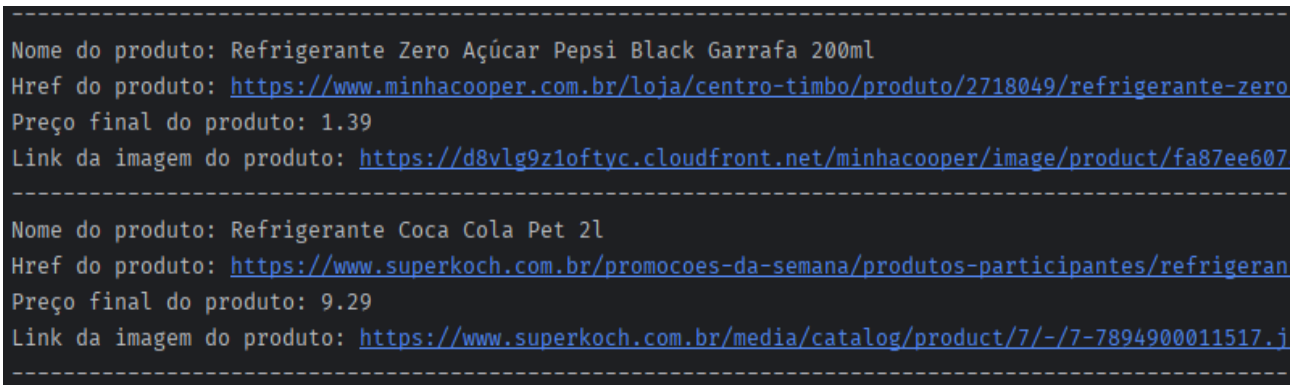
```
@PostMapping("/scraping/")
   public ResponseEntity<List<Produto_MercadoEntity>> getScraping() throws SQLException {
      List<Produto_MercadoEntity> produtosCooper = cooperScrapper.scrapeProducts();
      List<Produto_MercadoEntity> produtosKoch = kochScrapper.scrapeProducts();

      List<Produto_MercadoEntity> allProdutos = new ArrayList<>();
      allProdutos.addAll(produtosCooper);
      allProdutos.addAll(produtosKoch);

      for (Produto_MercadoEntity produtoEntity : allProdutos) {
         produtoMercadoDAO.addProduto(produtoEntity);
      }
      return ResponseEntity.status(HttpStatus.CREATED).build();
   }
```

Fonte: Elaborado pelos Autores.

Figura 7: Impressão no *Console* dos Dados Obtidos



Fonte: Elaborado pelos Autores.

5.2.2 *API OpenAI GPT*

A engenharia de *prompt* é uma parte crucial do desenvolvimento, onde foi necessário fazer algumas alterações para atender às necessidades específicas da aplicação. O *prompt* é a instrução que damos à *API* do *OpenAI* para orientar a geração de texto. Durante o desenvolvimento do projeto, foi utilizado algumas táticas sugestionadas pela *OpenAI* para construir um *prompt* eficaz, sendo:

Uso de delimitadores: Feita a utilização de delimitadores para indicar claramente partes distintas da entrada. Isso auxilia o *GPT* a compreender a estrutura do *prompt*, permitindo que ele gere uma resposta adequada. Essa técnica foi utilizada para instruir o *GPT* a formatar a resposta em um *Json*, da seguinte forma: *{id_grupo: insira o id(número) gerado aqui, id_produto: [insira o/os id(s) dos produtos aqui]}*. Indicando onde um valor específico deve ser inserido na resposta.

Especificação das etapas necessárias: No *prompt*, é especificado as etapas que o *GPT* deve seguir para concluir a tarefa. Isso inclui a análise dos produtos, a geração de um *id_grupo* e a atribuição deste *id_grupo* para produtos correspondentes entre os mercados.

Incluir detalhes para obter respostas mais relevantes: No *prompt*, é importante incluir detalhes relevantes para ajudar o *GPT* a gerar respostas mais precisas e adequadas. Especificar os diferentes sabores disponíveis e suas características. Por exemplo: “Entre os sabores, haverá opções sem açúcar e zero, que são a mesma coisa” para indicar que esses dois sabores são equivalentes.

Roles: São usadas para indicar “quem está falando” ou “agindo” em uma determinada parte da conversa. Existem dois *roles* principais: *SYSTEM* e *USER*.

System: Esta *role* é geralmente usada para instruções iniciais ou configurações de *chat*. Ela pode definir o comportamento do modelo ou fornecer instruções sobre o que o modelo deve fazer.

User: Esta *role* é usada para indicar a entrada do usuário na conversa. É a *role* que o modelo usa para entender o que o usuário está solicitando ou perguntando.

Esses *roles* ajudam a estruturar a conversa e permitem que o modelo entenda melhor o contexto da conversa.

A estrutura da requisição no código para *API GPT* pode ser observada no Quadro 2:

Quadro 2: Requisição para *API GPT*

```
private static final String OPENAI_MODEL = "gpt-4-1106-preview";
   String systemMessage = "Preciso que analise e agrupe TODOS produtos sem exceção." +
         "Entre os sabores vai ter sem açúcar e zero, que são a mesma coisa." +
         "Os volumes vão estar em ml= miligrama e l = litro  ";
   String userMessage = "Você vai receber uma lista de produto e deve analisar todos os produtos, levando em consideração a marca, a seção, o sabor e o volume de cada um." +

          " Cada grupo deve conter obrigatoriamente apenas produtos que são exatamente iguais, têm o mesmo sabor e volume, mas estão escritos de maneiras diferentes." +
          " Gere um id_grupo e atribua o mesmo id_grupo para produtos correspondentes entre os mercados e um id_grupo único para aquele produto que não teve nenhuma correspondência." +
          " Por favor, quero somente a resposta no formato json puro sem mais nenhum outro texto, exatamente dessa forma:" +
          " {id_grupo: insira o id(número) gerado aqui, id_produto: [insira o/os id(s) dos produtos aqui]}";

      private final RestTemplate restTemplate = new RestTemplate();
   public String getOpenAIResponse(String prompt) {
      String requestJson = "{"
```

```
+ "\"model\":\"" + OPENAI_MODEL + "\","
+ "\"messages\":["
+ "{\"role\":\"system\",\"content\":\"" + systemMessage + "\"},"
+ "{\"role\":\"user\",\"content\":\"" + userMessage + "\"},"
+ "{\"role\":\"user\",\"content\":\"" + prompt + "\"}"
+ "]};
```

Fonte: Elaborado pelos Autores.

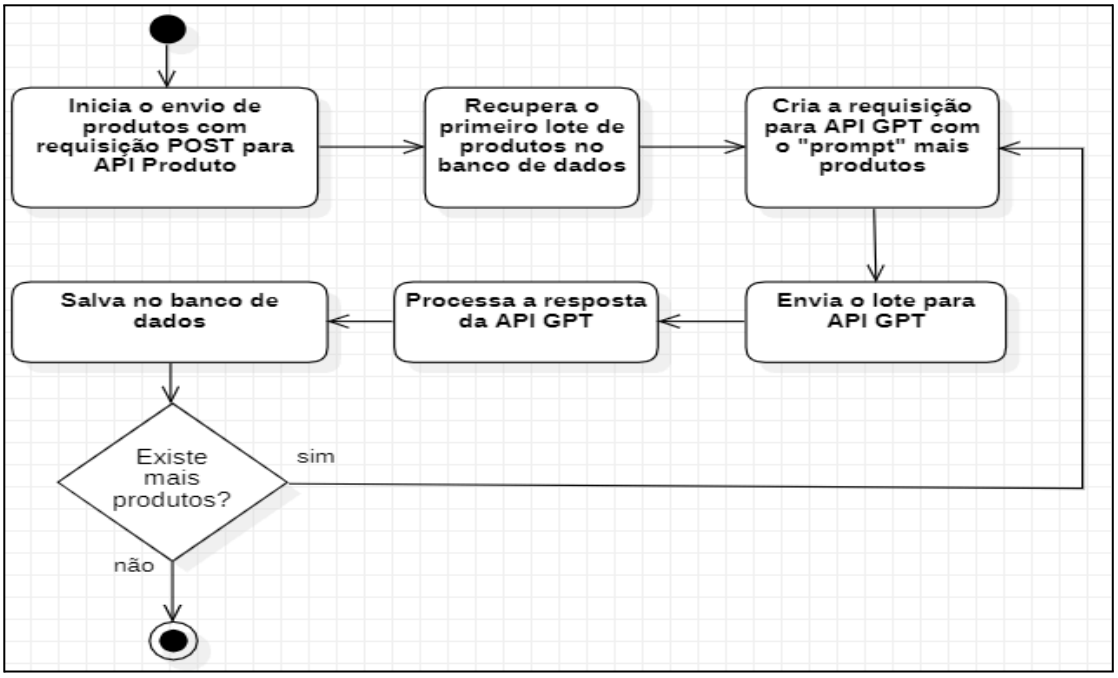
A requisição é feita passando uma lista de mensagens no corpo da requisição. A primeira mensagem instrui o *GPT* a retornar a resposta no formato especificado. As mensagens subsequentes contêm a lista de produtos a serem analisados.

Com a estrutura da interação com o *GPT* montada, foi aplicado um teste com os mesmos produtos utilizados na *PoC,* onde foram enviados todos 126 produtos da tabela 2, utilizando os modelos *gpt*-3.5-turbo-1106, *gpt*-4 e *gpt-4-1106-preview* mais todas táticas citadas antes.

Foi observado que apesar o modelo *gpt*-3.5-turbo-1106 ter o menor custo entre os modelos, seu desempenho comparado com modelos mais atuais era bastante inferior para o objetivo de relacionar produtos semelhantes, os modelos *gpt*-4 e *gpt*-4-1106-*preview* se mostraram bem mais competentes para a tarefa solicitada, chegando a um nível de acerto de 90% no relacionamento de produtos. Isso também possibilitou um maior entendimento sobre os custos e consumo de *tokens* nas requisições. Em comparação, o modelo *GPT*-4 consumiu um total de 4449 *tokens* e teve um custo de 0,78 reais, enquanto o modelo *gpt*-4-1106-*preview* consumiu 4829 *tokens* e custou 0,39 reais. Isso destaca a eficiência do modelo *gpt*-4-1106-*preview* por ser o modelo com a melhor relação custo-benefício.

Apesar do modelo *gpt-4-1106-preview* ter um limite de 150.000 *tokens* (pequenas unidades de informação) por minuto, 500 requisições por minuto e 10.000 requisições por dia, foi observado que o modelo pode "alucinar" quando a quantidade de produtos enviados por requisição excede 40. Para evitar isso, os produtos são enviados em lotes, como pode ser observado na Figura 8.

Figura 8: Fluxograma *API OpenAI GPT*



Fonte: Elaborado pelos Autores.

No Quadro 3, temos o método *sentGPT*() responsável pelo envio dos produtos mais *prompt*, analisar e salvar no banco de dados a resposta da *API GPT* . O desenvolvimento do método para implementar o envio por lote mencionado anteriormente, foi usando uma consulta *SQL* ao banco de dados usando *ORDER BY*, *LIMIT* e *OFFSET* , trazendo produtos por lotes definidos no código e ordenados por nome.

O *prompt* com os produtos é criado usando o método *criaPromptListaProduto()*, este recebe uma lista de produtos determinada pelo tamanho do lote e retorna um texto já formatado com o *id_produto* e nome em letras minúsculas.

A resposta da *API* é processada pelo método *criaGruposProdutosDaAPI()*, este método recebe a resposta, extrai os grupos de produtos e cria novos grupos de produtos com base nessas informações. Durante esse processo, o *id_grupo* gerado pelo *GPT* é descartado e um novo *id_grupo* é gerado usando um valor *UUID* (Identificador Único Universal). Os novos grupos de produtos são então inseridos no banco de dados. Isso garante que cada grupo de produtos tenha um identificador único.

Quadro 3: Método responsável por interação com *API GPT*

```
@PostMapping("/gpt/")
   public ResponseEntity<String> sentGPT() throws SQLException {
      int loteSize = 35;

      int totalProdutos = produtoMercadoDAO.getTotalProdutos();
      String response = null;

      for (int i = 0; i < totalProdutos; i += loteSize) {
                                    List<Produto_MercadoEntity>   loteProdutos   =
produtoMercadoDAO.getByMercado(loteSize, i);

         String promptListaProduto = criarPromptListaProduto(loteProdutos);
         response = chatGPT.getOpenAIResponse(promptListaProduto);
         criaGruposProdutosDaAPI(response);
      }
      return ResponseEntity.status(HttpStatus.CREATED).body(response);
   }
```

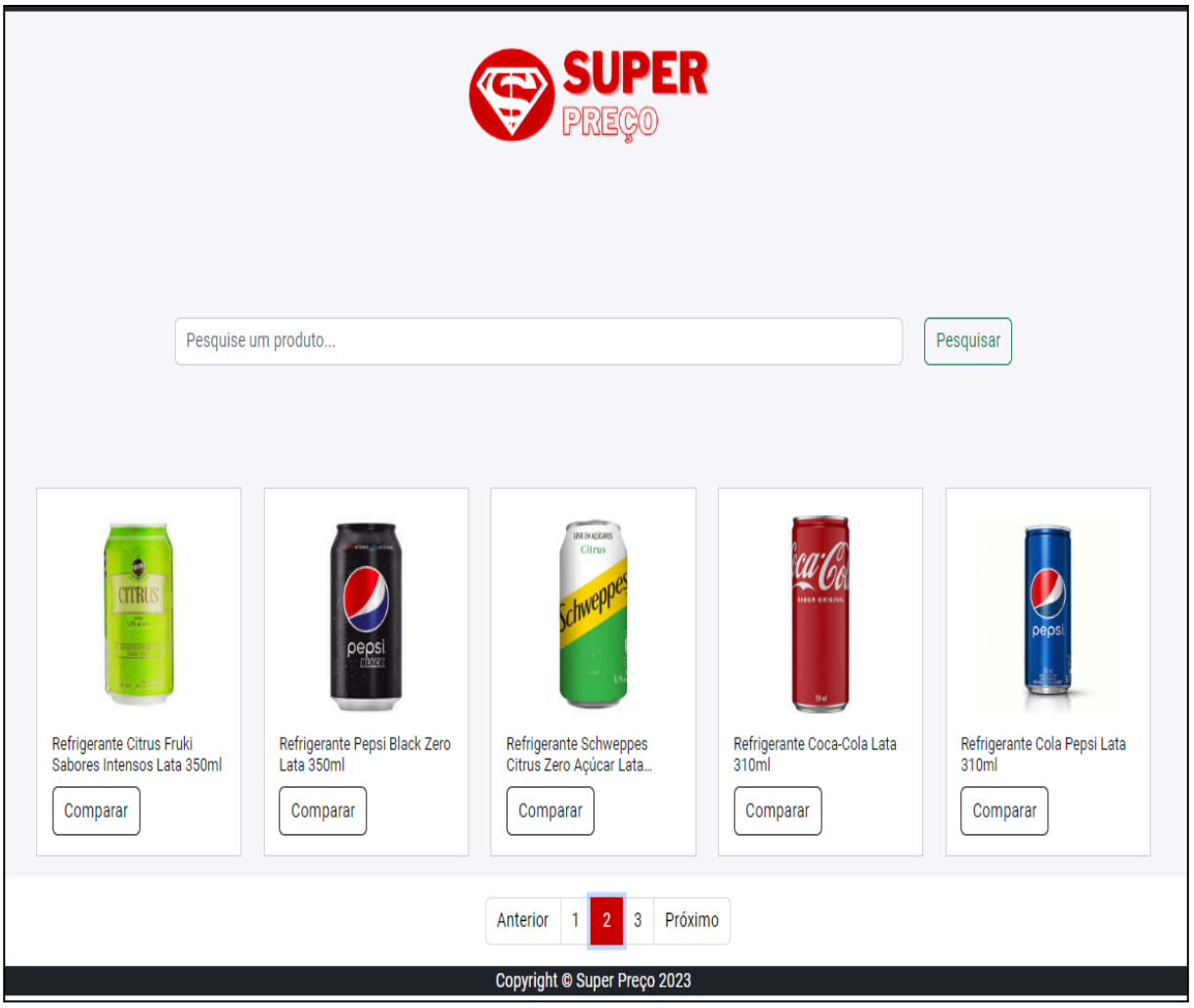
Fonte: Elaborado pelos Autores.

Foi selecionado 2 categorias para fazer o *scraping* de produtos e envio para análise, foi levado em consideração categorias onde a variedade de marcas era a menor possível, ou seja que a disponibilidade desses mesmos produtos em diferentes mercados era maior. As categorias escolhidas foram: enérgico e molho de tomate.

5.3 *Frontend*

Na figura 9 é mostrado a tela inicial do Super Preço, onde o usuário pode navegar na página visualizando produtos pré selecionados para o mesmo comparar, se para o usuário na tela inicial tiver o produto de sua escolha, apenas deve clicar em “comparar” que será redireionado na página do produto da figura 10. Caso tenha uma grande quantidade de produtos, o usuário pode clicar na opção de “próxima página” para visualizar todas as opções de produtos. Para o usuário tem a opção, de pesquisar o produto na barra de pesquisa como mostrado abaixo, apenas terá de digitar as iniciais do nome do produto, onde será mostrado o que já tem cadastrado como o nome digitado, ao selecionar o nome do produto, o usuário clica em pesquisar. Ao término da pesquisa é mostrado o produto escolhido, usuário apenas clica em “comparar” e será redirecionado na figura 10 da página do produto.

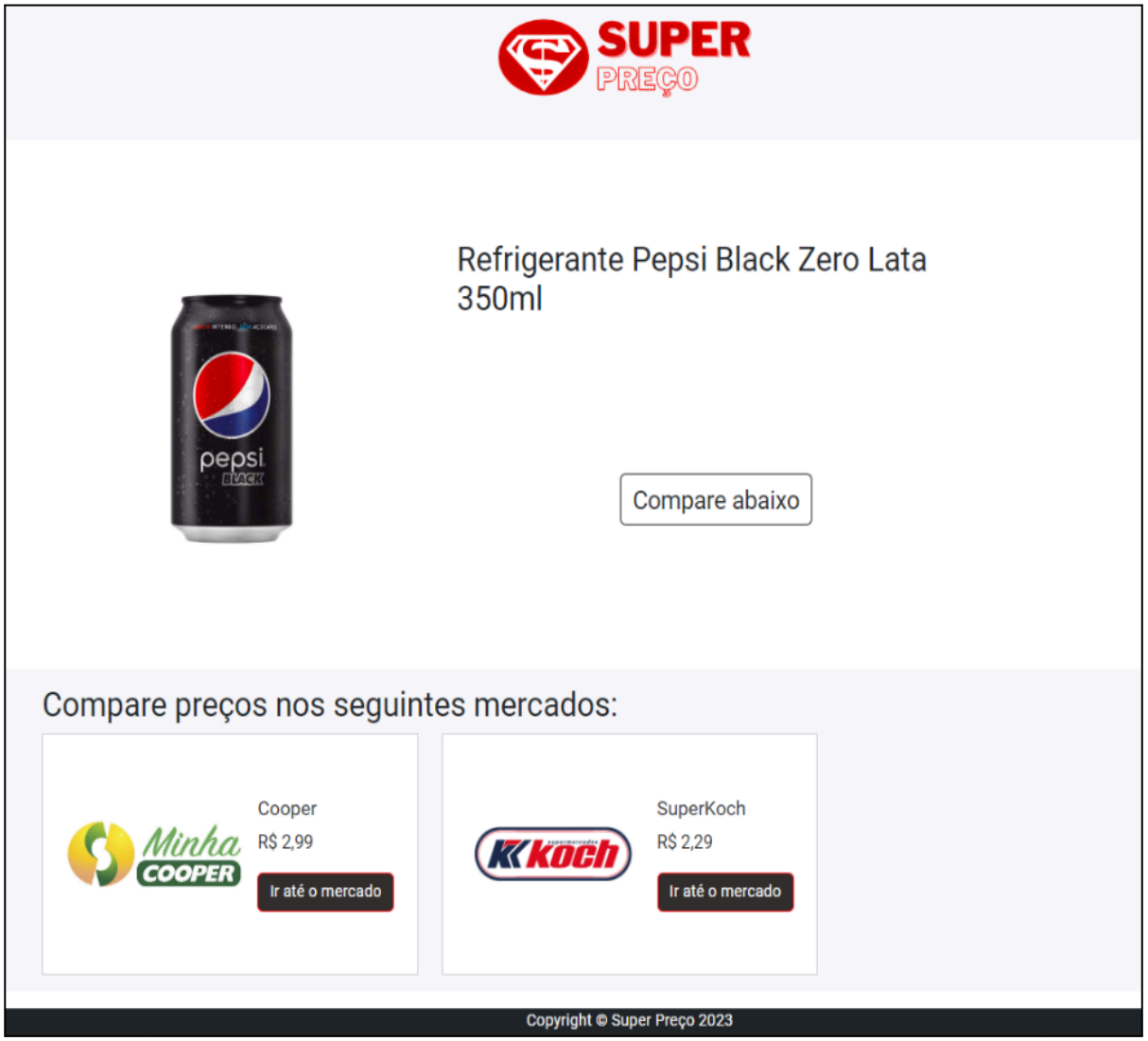
Figura 9: Página Inicial Super Preço



Fonte: Elaborado pelos Autores.

Ao selecionar o produto na página inicial na opção “comparar”, o usuário será redirecionado na Figura 10, onde terá a imagem do produto e seu nome, marca, quantidade / massa para visualizar. Logo abaixo da imagem do produto escolhido é mostrado o preço desse produto nos mercados cadastrados, caso usuário desejar, poderá ir na opção “ir ao mercado” para acessar o site do mercado em questão.

Figura 10: Página do Produto



Fonte: Elaborado pelos Autores.

6. CONCLUSÃO

O desenvolvimento desse trabalho mostrou os desafios e as oportunidades apresentadas pela implementação da *API GPT* da *OpenAI*, demonstrando a viabilidade da combinação de *web scraping* e inteligência artificial para simplificar a comparação de produtos em diferentes mercados.

O objetivo geral de desenvolver um comparador de preços de produtos de mercados *online* utilizando *API GPT* da *OpenAI* para relacionar os produtos, foi completamente atingido.

Os objetivos de fazer a relação dos produtos entre os mercados utilizando a *API GPT* da *OpenAI* e disponibilizar o preço de um mesmo produto em diferentes mercados cadastrados foram totalmente alcançados. O processo de ajuste e teste do *prompt* apresentou um desafio significativo, especialmente devido às atualizações contínuas feitas pela *OpenAI* em seu modelo e a complexidade de controle sobre o resultado esperado, aplicando apenas uma técnica de pré-processamento de dados, a taxa de acerto no pior dos casos varia entre 70% e 80%.

A obtenção de informações dos produtos pelo site de cada mercado usando *web scraping* foi um objetivo completamente atendido. Apesar da complexidade inerente à técnica de *web scraping*, sua eficácia e confiabilidade são bem estabelecidas, tornando-a uma escolha segura para a coleta de dados.

Este trabalho demonstra o potencial do modelo *GPT* da *OpenAI* como uma ferramenta para o relacionamento de dados, reforçando que a implementação da inteligência artificial pode simplificar significativamente o processo e torná-lo mais eficiente. Também propõe uma alternativa para o campo do desenvolvimento de *software*, especificamente no que diz respeito ao relacionamento de dados. As técnicas convencionais para relacionar produtos que fazem uso do *Global Trade Item Number* (GTIN) ou da *Stock Keeping Unit* (SKU), apresentam desafios significativos devido à sua disponibilidade limitada e à variação entre diferentes sistemas e mercados, sendo muitas vezes necessário aplicar diferentes lógicas de relacionamento de dados, como a tokenização, um processo que envolve a conversão de um dado (como um texto) em uma série de *tokens*, que podem ser usados para comparar a semelhança entre eles.

Uma oportunidade de melhoria que foi identificada, envolve aprimorar o trabalho de pré-processamento de dados, utilizando o *GPT* para identificar características únicas de cada produto, como sua marca, sabor, tamanho, seção e subseção. Ao alimentar o modelo com uma lista dessas características, ele poderá identificar e classificar cada produto de acordo. Ou até mesmo usando uma técnica chamada *fine-tuning* da *OpenAI*, que permite adaptar um modelo *GPT* já existente para atender a um caso de uso específico.

REFERÊNCIAS

SILVEIRA, Carolina. **Pesquisa revela que 56% dos consumidores usam sites comparadores de preços antes de comprar**. 2016. Disponível em: https://blog.allin.com.br/pesquisa-revela-que-56-dos-consumidores-usam-sites-comparadores-de-precos-antes-de-comprar/. Acesso em: 10 nov. 2023.

ABRAS. **87% dos brasileiros compram online, usando cada vez mais as redes sociais**. Disponível em: https://www.abras.com.br/clipping/geral/111374/87-dos-brasileiros-compram-online-usando-cada-vez-mais-as-redes-sociais#:~:text=Segundo%20dados%20da%20pesquisa%2C%2087,idade%20acima%20de%2016%20anos. Acesso em: 10 out. 2023.

NEIVA, Anna Carolina. **E-commerce no Brasil: dados de um mercado em expansão**. 2023. Disponível em: https://edrone.me/pt/blog/dados-ecommerce-brasil. Acesso em: 15 nov. 2023

Serasa. **Pesquisar Preços**. 2023 Disponível em: https://www.serasa.com.br/credito/blog/pesquisar-precos/. Acesso em: 15 out. 2023.

SANTANA, Marlesson. **Utilizando o Scrapy do Python para monitoramento em sites de notícias (Web Crawler)**, 2017. Disponível em: https://medium.com/@marlessonsantana/utilizando-o-scrapy-do-python-para-monitoramento-em-sites-de-not%C3%ADcias-web-crawler-ebdf7f1e4966. Acesso em: 01 out. 2023.

MOOR, James. **Conferência de Inteligência Artificial do Dartmouth College: Os Próximos Cinquenta Anos**. Revista AI , 2006. Disponível em: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/1911. Acesso em: 01 dez. 2023.

Terra. **A importância e o desenvolvimento da Inteligência Artificial no mundo.** 2021. Disponivel em: https://www.terra.com.br/noticias/dino/a-importancia-e-o-desenvolvimento-da-inteligencia-artificial-no-mundo,0d0b85a0334cf3ee87222d570c6b08adefmk3zea.html. Acesso em: 15 out. 2023

TechTarget. **O que é e como funciona a web scrapin**g. 2023. Disponível em: https://www.computerweekly.com/br/definicoe/O-que-e-e-como-funciona-a-web-scraping#:~:text=Uma%20ferramenta%20de%20raspagem%20da,em%20seu%20computador%20sem%20codifica%C3%A7%C3%A3o.](https://www.computerweekly.com/br/definicoe/O-que-e-e-como-funciona-a-web-scraping#:~:text=Uma%20ferramenta%20de%20raspagem%20da,em%20seu%20computador%20sem%20codifica%C3%A7%C3%A3o. Acesso em: 10 out. 2023.

FERNANDES, Flávia. **O que é Inteligência Artificial: veja como surgiu, exemplos e polêmicas**. 2023. Disponível em: https://www.techtudo.com.br/guia/2023/03/o-que-e-inteligencia-artificial-veja-como-surgiu-exemplos-e-polemicas-edsoftwares.ghtml. Acesso em: 05 out. 2023.

OLIVEIRA, Flávia. **História da Inteligência Artificial (IA)**. Disponível em: https://tinbot.com.br/blog/historia-da-inteligencia-artificial-ia/. Acesso em: 25 out. 2023.

FCDL Santa Catarina. **7 benefícios da inteligência artificial no varejo**. Disponível em: https://www.fcdl-sc.org.br/fcdl-noticias/7-beneficios-da-inteligencia-artificial-no-varejo/. Acesso em: 25 set. 2023.

MERLYN, Luana. **Conheça as vantagens dos comparadores de preço**. 2017. Disponível em: https://blog.yooper.com.br/conheca-as-vantagens-dos-comparadores-de-preco/. Acesso em: 29 set. 2023.

SUTTO, Giovanna. **Quase 40% dos consumidores já buscam preços por compras mais seguras na Black Friday, mostra pesquisa.** 2023. Disponível em: https://www.infomoney.com.br/consumo/quase-40-dos-consumidores-ja-buscam-precos-por-compras-mais-seguras-na-black-friday-mostra-pesquisa/. Acesso em: 13 out. 2023.

CHEN, Lingjiao; ZAHARIA, Matei; ZOU, James. **How Is ChatGPT's Behavior Changing over Time?.** 2023. Disponível em: https://arxiv.org/pdf/2307.09009.pdf. Acesso em: 12 nov. 2023.

GRANDA, Alana. **Preços determinam decisão de compra dos consumidores, indica pesquisa | Agência Brasil.** 2015. Disponível em: https://agenciabrasil.ebc.com.br/economia/noticia/2015-03/precos-determinam-decisoes-de-compra-dos-consumidores-brasileiros-indica. Acesso em: 12 nov. 2023.

SPC BRASIL. **Nove em cada dez consumidores virtuais consultam a internet antes de realizar uma compra.** 2015. Disponível em: https://www.spcbrasil.org.br/uploads/st_imprensa/release_compras_online_offline_maio_2015_v51.pdf. Acesso em: 11 nov. 2023.

Open AI. **Apresentando *ChatGPT***. 2022. Disponível em: https://openai.com/blog/chatgpt . Acesso em: 01 nov. 2023.

TEIXEIRA, Tarcisio. **Comércio Eletrônico: conforme o Marco Civil da Internet e a regulamentação do e-commerce no Brasil**. Tarcisio Teixeira - São Paulo : Saraiva, 2015. Disponível em: https://bdjur.stj.jus.br/jspui/bitstream/2011/96224/com%c3%a9rcio_eletr%c3%b4nico_conforme.pdf. Acesso em: 01 nov. 2023.

KAUFMAN, Dora. **A inteligência artificial irá suplantar a inteligência humana?** / Dora Kaufman - Barueri, SP: Estação das Letras e Cores, 2018. Disponível em: https://books.google.com.br/books?hl=en&lr=&id=Fh-WDwAAQBAJ&oi=fnd&pg=PT5&dq=A+intelig%C3%AAncia+artificial+ir%C3%A1+suplantar+a+intelig%C3%AAncia+humana%3F&ots=owNKIgCa61&sig=Rjmib1vip_xhp_aovZ1x5SkZJVk&redir_esc=y#v=onepage&q=A%20intelig%C3%AAncia%20artificial%20ir%C3%A1%20suplantar%20a%20intelig%C3%AAncia%20humana%3F&f=false. Acesso em: 10 out. 2023.

JAMES, Mike. **Inteligência artificial em BASIC** / Mike James - 2.ed. - Rio de Janeiro: Campus, 1986. Disponível em: https://ia802300.us.archive.org/12/items/inteligencia-artificial-em-basic-2ed/Inteligencia_Artificial_em_Basic-2ed.pdf. Acesso em: 10 out. 2023.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approac**h. / Stuart J. Russell and Peter Norvig - 4.ed. - Upper Saddle River: Pearson Education Limited. 2010. Acesso em: 20 out. 2023.

BAKOS, Yannis. **The Emerging Landscape for Retail E-Commerce**. Yannis Bakos - Journal of Economic Perspectives - Vol 15, Number 1 - 2001. Disponível em: https://pubs.aeaweb.org/doi/pdfplus/10.1257/jep.15.1.69. Acesso em: 07 out. 2023.

MITCHELL, Ryan. **Web Scraping with Python**. Collection More Data from the Modern Web / Ryan Mitchell - 2.ed. - O'Reilly Media. - 2018. Disponível em: https://books.google.com.br/books?id=U4tSDwAAQBAJ&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false. Acesso em: 02 out. 2023.

TURING, Alan. **Computing machinery and intelligence** / Alan Turing - Vol 59, p. 433-460, 1950. Disponível em: https://academic.oup.com/mind/article/LIX/236/433/986238. Acesso em: 15 set. 2023.

MCCARTHY, John. **What is Artificial Intelligence**. 2007. Disponível em: https://www-formal.stanford.edu/jmc/whatisai.pdf. Acesso em: 28 set. 2023.

FOROUZAN, Behrouz. **Fundamentos da Ciência da Computação** / Cengage Universitário - 2.ed. 2011.Acesso em: 28 out. 2023.

CARVALHO, Marco Antônio Brandão. **Sistema web para comparação dos preços de supermercados online**. 2022. 56 f. Monografia (Graduação em Sistemas de Informação) - Instituto de Ciências Exatas e Aplicadas, Universidade Federal de Ouro Preto, João Monlevade, 2022. Acesso em: 01 out. 2023.

REDAÇÃO VIVO MEU NEGÓCIO. **Web Scraping: como usar a coleta de dados a favor dos negócios**. 2022. Disponível em: https://vivomeunegocio.com.br/conteudos-gerais/expandir/web-scraping/. Acesso em: 26 out. 2023.

Weni. **Prompts para ChatGPT: conheça os tipos e crie conversas inteligentes**. 2023. Disponível em: https://weni.ai/blog/prompts-chatgpt/. Acesso em: 01 dez. 2023.

SIRISURIYA, S. **A Comparative Study on Web Scraping** / S Sirisuriya - 8th International Research Conference, KDU, 2015.

LIMA, Fabiana. **O que é e para que serve o Web Scraping?**. 2022. Disponível em: https://www.remessaonline.com.br/blog/web-scraping/#:~:text=O%20Web%20Scraping%20%C3%A9%20uma%20t%C3%A9cnica%20utilizada%20diariamente%20para%20copiar,Web%20Scraping%20s%C3%A3o%20as%20empresas. Acesso em: 02 out. 2023.

LOURENÇO, Anália *et al*. **Tecnologias de web scraping em um mundo de API** / Anália Lourenço - Briefings in Bioinformatics - Vol 15, 5.ed. 2014. Disponível em: https://academic.oup.com/bib/article/15/5/788/2422275. Acesso em: 10 set. 2023.

ANEXOS

Anexo A: Planilha Produtos

Cooper	**Superkoch**
Refrigerante sem Açúcar Coca-Cola Garrafa 1,5L	Refrigerante Coca Cola Zero Pet 1.5l
Refrigerante sem Açúcar Coca-Cola Zero Garrafa 2L	Refrigerante Coca Cola Zero Pet 2 Litros
Refrigerante Coca-Cola Sem Açucar Retornável Pet 2L	
Refrigerante Coca-Cola Sem Açúcar Lata 310ml	Refrigerante Coca Cola Zero Lata 310ml
Refrigerante Coca-Cola Lata 310ml	Refrigerante Coca Cola Sleek Lata 310ml
Refrigerante Limão Sprite Garrafa 2L	Refrigerante Sprite Limão Pet 2l
Refrigerante Pepsi Black Zero Pet 2L	Refrigerante Pepsi Black Sem Açúcar Garrafa 2l
Refrigerante Fanta Laranja Pet 600ml	Refrigerante Fanta Laranja Pet 600ml
Refrigerante Schweppes Citrus Original Lata 350ml	Refrigerante Schweppes Citrus Original Lata 350ml
Chá Mate Natural Matte Leão Copo 300ml	
Chá Mate Limão Matte Leão Garrafa 1,5L	Chá Leao Matte Limão Pet 1.5
Chá Mate Original Matte Leão Garrafa 1,5L	Chá Leão Matte Original Pet 1.5l
Bebida à Base de Soja Maçã Ades Caixa 1L	Alimento Soja Ades Tradicional Maçã Tetra Pack 1l
Bebida à Base de Soja Uva Ades Caixa 1L	Alimento Soja Ades Tradicional Uva Tetra Pack 1l
Água de Coco Esterilizada Kero Coco Caixa 200ml	Agua De Coco Kero Coco 200ml
Água de Coco Esterilizada Kero Coco Caixa 1L	Água De Coco Kero Coco Tetra Pack 1
Cerveja Antarctica Lata 350ml	Cerveja Antarctica Pilsen Lata 350ml
Cerveja Skol Lata 350ml	Cerveja Skol Pilsen Lata 350ml
Cerveja Corona Garrafa 330ml	Cerveja Corona Extra Long Neck 330ml
Cerveja Lager Heineken Garrafa 330ml	Cerveja Heineken Long Neck 330ml
Coquetel Alcoólico Maracujá Natasha Hits Garrafa 900ml	
Vodka Smirnoff 600ml	Vodka Smirnoff 600ml
Vodka Natasha Garrafa 900ml	Vodka Natasha Garrafa 900ml
Vodka Destilada Absolut Garrafa 750ml	Vodka Absolut Garrafa 750ml
Feijão Vermelho Tipo 1 Urbano Pacote 1kg	Feijão Urbano Vermelho Pacote 1kg
Feijão Azuki Caldão Pacote 500g	Feijão Caldão Azuki Vermelho Pc 500g
Feijão Preto Tipo 1 Urbano Pacote 1kg	Feijão Urbano Preto Pacote 1kg
Feijão Carioca Tipo 1 Urbano Seleção Especial Pacote 1kg	Feijão Urbano Carioca Pacote 1kg
Arroz Parboilizado Tipo 1 Dalfovo Pacote 1kg	

Arroz Branco Tipo 1 Tio João 100% Grãos Nobres Pacote 1kg	Arroz Tio João Branco Pacote 1kg
Arroz Parboilizado Tipo 1 Urbano Pacote 1kg	Arroz Urbano Parboilizado Pacote 1kg
Marshmallow de Baunilha Camping Fini Pacote 200g	Marshmallow Fini Camping Tradicional Pacote 250g
Marshmallow Torção Baunilha Fini Pacote 80g	Marshmallow Fini Torção Pacote 60g
Bala de Gelatinas Bananas Fini Pacote 90g	Bala Fini Gelatina Banana Pacote 100g
Bala de Morango e Nata Regaliz Tubes Azedinhos Fini Pacote 80g	Bala Fini Tubinho Doce Morango Pacote 80g
Chicle de Melancia Azedinha Fini Pacote 80g	
Bala de Gelatinas Morango e Nata Beijos Fini Pacote 90g	Bala Fini Gelatina Beijo Morango Pacote 100g
Chocolate em Barra Meio Amargo com Amêndoas Garoto Talento 85g	Chocolate Garoto Talento Meio Amargo Com Amêndoas Barra 85g
Chocolate em Barra ao Leite com Castanha-do-Pará Garoto Talento 85g	Chocolate Garoto Talento Castanha Para Barra 85g
Chocolate em Barra ao Leite Garoto 80g	Chocolate Garoto Ao Leite Br 80g
Biscoito Maria Parati Pacote 370g	Biscoito Parati Maria Pacote 370g
Biscoito Wafer Recheio Brigadeiro Minueto Pacote 115g	Biscoito Minueto Wafer Brigadeiro Pacote 115g
Biscoito Cookie Chocolate Bauducco Pacote 100g	Biscoito Bauducco Cookies Chocolate Pacote 100g
Biscoito de Arroz Integral Pimenta Camil Pacote 150g	
Tempero em Pó para Feijão Ajinomoto Sazón Pacote 60g 12 Unidades	Tempero Sazon Marrom Feijão 60g
Tempero em Pó para Carnes Ajinomoto Sazón Pacote 60g 12 Unidades	Tempero Sazon Vermelho Carnes 60g
Óleo de Milho Tipo 1 Liza Especiais Garrafa 900ml	Oleo De Milho Liza Pet 900ml
Óleo de Soja Soya Garrafa 900ml	Óleo De Soja Soya Pet 900ml
Pizza de Mussarela Congelada Sadia Caixa 440g	Pizza Sadia Mussarela 440g
Pizza de Calabresa Congelada Sadia Caixa 460g	Pizza Sadia Calabresa 460g
Requeijão Cremoso Tradicional Tirol Copo 180g	Requeijão Tirol Cremoso Tradicional 180g
Requeijão Cremoso Lac Lélo Copo 180g	Requeijão Lac Lelo Cremoso Tradicional 180g
Filé de Tilápia Congelado Copacol Pacote IQF 800g	Filé De Tilápia Copacol Congelada 800g
Coxa com Sobrecoxa à Passarinho de Frango Congelado Copacol Pacote IQF 800g	Coxa E Sobrecoxa De Frango Copacol Apas 800g
Filezinho de Peito de Frango Sassami Congelado Sadia Pacote IQF 1kg	Sassami Frango Sadia Sem Tempero 1kg
Pilha Alcalina AA Rayovac 2 Unidades 1,5V	Pilha Rayovac Alcalina Aa Pequena Com 2

Pilha Alcalina AA Duracell 1,5V com 2 Unidades	Pilha Duracell Alcalina Peq Aa C/2
Lava-Roupas Concentrado Ariel Expert Frasco 1,2L	Lava Roupa Ariel 1.2l
Lava-Roupas em Pó Omo Lavagem Perfeita Sanitiza & Higieniza Caixa 1,6kg	Lava Roupa Omo Lavagem Perfeita Sanitiza/Hig 1.6kg Em Pó
Shampoo Anticaspa Clear Men Limpeza Diária Frasco 200ml	Shampoo Clear 2x1 Limpeza Diaria Frasco 200ml
Kit Shampoo 275ml + Condicionador 175ml Niely Gold Pós-Quimica Óleo de Argan	Kit Shampoo 275ml+Condicionador 175ml Niely Gold Pós Química
Gel Fixador Aspecto Molhado Fixação Média 2 Bozzano Pote 300g	Gel Fixador Bozzano Incolor Pote 300g
Gel Fixador Capilar Fixação 4 Ny.Looks Frasco 240g	Gel Fixador Ny Looks Azul Frasco 240g
Antitranspirante Spray Pegador Old Spice 150ml	
Antitranspirante Aerossol sem Perfume Rexona Clinical 150ml	Desodorante Rexona Clinical Sem Perfume Aerosol 150ml
Antitranspirante Aerossol Marine Axe 152ml	
Antitranspirante Aerossol Axe Urban 152ml	Desodorante Axe Masculino Urban Aerosol 152ml

Fonte: Elaborado pelos Autores

CÓDIGO

```
package com.cedup.super_preco.controller.produto;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;


@Component
public class ChatGPT {

    // limite de token = https://platform.openai.com/docs/models/gpt-3-5
    private static final String endpoint = "https://api.openai.com/v1/chat/completions";
    private static final String OPENAI_MODEL = "gpt-4-1106-preview";
    String systemMessage = "Preciso que analise e agrupe TODOS produtos sem exceção." +
            "Entre os sabores vai ter sem açúcar e zero, que são a mesma coisa." +
            "Os volumes vão estar em ml= miligrama e l = litro  ";
     String userMessage = "Voce vai receber uma lista de produto e deve analisar todos os produtos, levando em consideração a marca, a seção, o sabor e o volume de cada um." +
             " Cada grupo deve conter obrigatoriamente apenas produtos que são exatamente iguais, têm o mesmo sabor e volume, mas estão escritos de maneiras diferentes." +
              " Gere um id_grupo e atribua o mesmo id_grupo para produtos correspondentes entre os mercados e um id_grupo único para aquele produto que não teve nenhuma correspondência." +
             " Por favor, quero somente a resposta no formato json puro sem mais nenhum outro texto, exatamente dessa forma:" +
              " {id_grupo: insira o id(numero) gerado aqui, id_produto: [insira o/os id(s) dos produtos aqui]}";

    @Value("${openai-api-key}")
    private String apiKey;

    private final RestTemplate restTemplate = new RestTemplate();

    public String getOpenAIResponse(String prompt) {
        // Crie o corpo da solicitação JSON
        String requestJson = "{"
                + "\"model\":\"" + OPENAI_MODEL + "\","
                + "\"messages\":["
                + "{\"role\":\"system\",\"content\":\"" + systemMessage + "\"},"
                + "{\"role\":\"user\",\"content\":\"" + userMessage + "\"},"
                + "{\"role\":\"user\",\"content\":\"" + prompt + "\"}"
                + "]"
                + "}";
```

```
        //System.out.println(prompt);

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        headers.set("Authorization", "Bearer " + apiKey);

        HttpEntity<String> entity = new HttpEntity<>(requestJson, headers);
        ResponseEntity<String> response = restTemplate.postForEntity(endpoint, entity, String.class);

        return response.getBody();
    }
}
```

```
package com.cedup.super_preco.controller.produto;

import com.cedup.super_preco.model.mercado.MercadoEntity;
import com.cedup.super_preco.model.produto.ProdutoEntity;
import com.cedup.super_preco.model.produto.Produto_MercadoEntity;
import org.jsoup.Connection;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

@Component
public class CooperScrapper {

    /**
     * Scrapes products from a website.
     *
     * @return a list of Produto_MercadoEntity objects representing the scraped products
     */
    public List<Produto_MercadoEntity> scrapeProducts() {

        List<Produto_MercadoEntity> produtos = new ArrayList<>();
        try {
            // para contar o total de produtos que vão ser raspados
            int totalProductCount = 0;

            // URL base do site
            String baseUrl = "https://www.minhacooper.com.br/loja/centro-timbo/produto/listar/";
            // Lista de IDs das categorias que foram tiradas da url do site
```

```
        List<String> categoryIds = Arrays.asList("205"); //, "40", "197"

        // Iterando sobre cada ID de categoria
        for (String categoryId : categoryIds) {
            // começa na página em 1
            int pageNumber = 1;
            // para contar o total de produtos que vão ser raspados por categoria/link
            int productCount = 0;
             // variável inicia como verdadeira, indicando que existe uma próxima página, ou seja, que existe um elemento com a classe 'ajax-pagination'
            boolean hasNextPage = true;

                 // loop para fazer a rapagem, onde interrompe quando um elemento com a classe 'ajax-pagination' não for encontrado mais
            while (hasNextPage) {
                 // Construindo a URL da categoria atual, concatenando url base + id da categoria da lista 'idsCategorias' + parte da url que está relacionado com a pagina atual (?page=) + numero da pagina gerado pelo contador
                String url = baseUrl + categoryId + "?page=" + pageNumber;

                // Conectando à página atual
                Connection connection = Jsoup.connect(url)
                        .userAgent("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36");

                // cookie necessário para a pagina da cooper funcionar
                connection.cookie("subsidiaryId", "centro-timbo");

                Document doc = connection.get();

                  // Selecione os elementos com a classe .product-variation__details, que é a div pai dos elementos product-variation__...
                Elements products = doc.select(".product-variation");

                        // Iterando sobre cada produto = para cada elemento achado com a classe 'product-variation__details' faça isso:
                for (Element product : products) {
                       // atribui para elementoNomeProduto o primeiro elemento achado com a classe "product-variation__name"
                    Element productNameElement = product.selectFirst(".product-variation__name");
                    //
                    String productName = productNameElement.text();
                                    String productHref = "https://www.minhacooper.com.br" + productNameElement.attr("href");
                    // Alguns produtos podem ter um desconto = html com preço é diferente
                                                    Element productFinalPriceElement = product.selectFirst(".product-variation__final-price");
                    if (productFinalPriceElement == null) {
                           // Se o produto tem um preço com desconto, selecione o elemento com a classe '.preco-desconto'
```

```
                    productFinalPriceElement = product.selectFirst(".preco-desconto");
                }
                // formatar o texto do preço para apenas o número
                String productFinalPrice = productFinalPriceElement.text();
                // Remover todos os caracteres não numéricos
                productFinalPrice = productFinalPrice.replaceAll("[^\\d.,]", "");
                // Substituir vírgulas por pontos
                productFinalPrice = productFinalPrice.replace(',', '.');
                // Converter para double
                double priceDouble = Double.parseDouble(productFinalPrice);
                                                    String productImageLink = "https:" + product.parent().selectFirst(".product-variation__image-container img").attr("src");

                // cria uma nova instância de Produto_MercadoDTO
                        Produto_MercadoEntity productInfo = new Produto_MercadoEntity(0, new MercadoEntity(1), new ProdutoEntity("0"), productName, priceDouble, productHref, productImageLink);
                // adiciona o produto à lista
                produtos.add(productInfo);

                System.out.println("Nome do produto: " + productName);
                System.out.println("Href do produto: " + productHref);
                System.out.println("Preço final do produto: " + productFinalPrice);
                System.out.println("Link da imagem do produto: " + productImageLink);

System.out.println("----------------------------------------------------------------------------------------------------");
                productCount++;
                totalProductCount++;
            }

             // usando jsoup seleciona todos elementos com a classe ajax-pagination e armazena em 'nextPageElement'
            Elements nextPageElement = doc.select(".ajax-pagination");
            // operador lógico 'NOT':
                // quando 'nextPageElement' for true (lista nextPageElement está vazia) '!' inverte e hasNextPage recebe 'false', indicando que NÂO tem uma próxima página
              // quando 'nextPageElement' for false (lista nextPageElement não está vazia) '!' inverte e hasNextPage recebe 'true', indicando que TEM uma próxima página
            hasNextPage = !nextPageElement.isEmpty();

            // Incrementando o número da página para a próxima iteração
            pageNumber++;
        }
         System.out.println("Número de produtos raspados para a categoria " + categoryId + ": " + productCount);

System.out.println("############################################################################################################");
    }
```

```
            System.out.println("Total de produtos raspados: " + totalProductCount);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return produtos;
    }
}
```

—----------

```
package com.cedup.super_preco.controller.produto;

import com.cedup.super_preco.model.mercado.MercadoEntity;
import com.cedup.super_preco.model.produto.ProdutoEntity;
import com.cedup.super_preco.model.produto.Produto_MercadoEntity;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

@Component
public class KochScrapper {

    public List<Produto_MercadoEntity> scrapeProducts() {
        List<Produto_MercadoEntity> produtos = new ArrayList<>();
        try {
            // para contar o total de produtos que vão ser raspados
            int totalProductCount = 0;

            // URL base do site
            String baseUrl = "https://www.superkoch.com.br";
            // Lista de URLs usando o método 'asList'
            List<String> categoryPaths = Arrays.asList(
                    "/bebidas/refrigerante"
            );

            // Iterando sobre cada URL
            for (String categoryPath : categoryPaths) {
                // começa na página em 1
                int pageNumber = 1;
                // para contar o total de produtos que vão ser raspados por categoria/link
                int productCount = 0;
                 // variável inicia como verdadeira, indicando que existe uma próxima página, ou seja, que
existe um elemento com a classe 'pages-item-next'
```

```
        boolean hasNextPage = true;

                // loop para fazer a rapagem, onde interrompe quando um elemento com a classe 'pages-item-next' não for encontrado mais
        while (hasNextPage) {

                // Construindo a URL da categoria atual, concatenando url base + caminho da próxima categoria na lista 'categoryPaths' + parte da url que está relacionado com a pagina atual (?p=) + numero da pagina gerado pelo contador
            String url = baseUrl + categoryPath + "?p=" + pageNumber;
            // inicia a conexão com a página a ser raspada,
            Document doc = Jsoup.connect(url)
                // cabeçalho do agente
                  .userAgent("Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36")
                .get();

                  // Seleciona os elementos com a classe .product-item-info, que é a div pai dos elementos product-item-...
            Elements products = doc.select(".product-item-info");

            // Itera sobre cada produto
            for (Element product : products) {
               Element productNameElement = product.selectFirst(".product-item-link");
               String productName = productNameElement.text();
               String productHref = productNameElement.attr("href");
                    // no koch pode ter 2 preços com a mesma classe 'price' para o mesmo produto, porem que importa pra gente é o segundo
                     // então vamos obter todos os elementos com a classe 'price' e armazenar numa variável do tipo Elements
            Elements productPrices = product.select(".price");
                     // agora vamos declarar uma variável do tipo String que vai receber o preço do produto
            String productPrice;
                    // Verifica se na lista 'productPrices" tem mais de dois preços (elementos com a classe price)
            if (productPrices.size() == 2) {
                        // Se tiver, pegue o texto do segundo, na lista get(0) = primeiro // get(1) == segundo
                productPrice = productPrices.get(1).text();
            } else {
               // Se não, pegue o texto do primeiro
               productPrice = productPrices.first().text();
            }

            // formatar o texto do preço para apenas o número
            // Remover todos os caracteres não numéricos
            productPrice = productPrice.replaceAll("[^\\d.,]", "");
            // Substituir vírgulas por pontos
            productPrice = productPrice.replace(',', '.');
```

```
                    // Converter para double
                    double priceDouble = Double.parseDouble(productPrice);

                    // imagem em baixa resolução/otimizada
                                                                String productImageLink =
product.parent().selectFirst(".product-image-photo").attr("src");

                    // cria nova instância de Produto_MercadoDTO
                            Produto_MercadoEntity productInfo = new Produto_MercadoEntity(0, new
MercadoEntity(2), new ProdutoEntity("0"), productName, priceDouble, productHref,
productImageLink);
                    // adiciona o produto à lista 'produto"
                    produtos.add(productInfo);

                    System.out.println("Nome do produto: " + productName);
                    System.out.println("Href do produto: " + productHref);
                    System.out.println("Preço final do produto: " + productPrice);
                    System.out.println("Link da imagem do produto: " + productImageLink);

System.out.println("----------------------------------------------------------------------------------------------
----------");
                    productCount++;
                    totalProductCount++;
                }

                // Verificando se há uma próxima página
                Elements nextPageElement = doc.select(".pages-item-next");
                hasNextPage = !nextPageElement.isEmpty();

                // Incrementando o número da página para a próxima iteração
                pageNumber++;
            }
            System.out.println("Número de produtos raspados para a categoria " + categoryPath + ": "
+ productCount);
        }
        // Imprimindo o número total de produtos raspados
        System.out.println("Total de produtos raspados: " + totalProductCount);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
    return produtos;
  }
}
—----------------------------------------------------------------------------------------------------------------
package com.cedup.super_preco.controller.produto;


import java.util.List;

public class Produto_MercadoDTO {
```

```
    public int id_produto_mercado;
    public int id_mercado;
    public String id_produto;
    public String nome;
    public double preco;
    public String link;
    public String link_img;

     public Produto_MercadoDTO(int id_produto_mercado, int id_mercado, String id_produto, String
nome, double preco, String link, String link_img) {
        this.id_produto_mercado = id_produto_mercado;
        this.id_mercado = id_mercado;
        this.id_produto = id_produto;
        this.nome = nome;
        this.preco = preco;
        this.link = link;
        this.link_img = link_img;
    }

    public Produto_MercadoDTO(int id_produto_mercado, int id_mercado, String nome) {
        this.id_produto_mercado = id_produto_mercado;
        this.id_mercado = id_mercado;
        this.nome = nome;
    }

    @Override
    public String toString() {
        return "\n id_produto: " + id_produto_mercado + " nome: " + nome;

    }

    public int getId_produto_mercado() {
        return id_produto_mercado;
    }

    public void setId_produto_mercado(int id_produto_mercado) {
        this.id_produto_mercado = id_produto_mercado;
    }

    public int getId_mercado() {
        return id_mercado;
    }

    public void setId_mercado(int id_mercado) {
        this.id_mercado = id_mercado;
    }

    public String getNome() {
        return nome;
    }
```

```
    public void setNome(String nome) {
        this.nome = nome;
    }

    public double getPreco() {
        return preco;
    }

    public void setPreco(double preco) {
        this.preco = preco;
    }

    public String getLink() {
        return link;
    }

    public void setLink(String link) {
        this.link = link;
    }

    public String getLink_img() {
        return link_img;
    }

    public void setLink_img(String link_img) {
        this.link_img = link_img;
    }

    public String getId_produto() {
        return id_produto;
    }

    public void setId_produto(String id_produto) {
        this.id_produto = id_produto;
    }
}
```

—----------

```
package com.cedup.super_preco.controller.produto;

import com.cedup.super_preco.model.produto.ProdutoDAO;
import com.cedup.super_preco.model.produto.ProdutoEntity;
import com.cedup.super_preco.model.produto.Produto_MercadoDAO;
import com.cedup.super_preco.model.produto.Produto_MercadoEntity;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
```

```
import org.springframework.web.bind.annotation.*;

import java.sql.SQLException;
import java.util.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

@RestController
@RequestMapping("/produto/")
@CrossOrigin(origins = "*")
public class ProdutoController {
    @Autowired
    Produto_MercadoDAO produtoMercadoDAO;
    @Autowired
    ProdutoDAO produtoDAO;
    @Autowired
    CooperScrapper cooperScrapper;
    @Autowired
    KochScrapper kochScrapper;
    @Autowired
    ChatGPT chatGPT;
    @Autowired
    ProdutoConverter produtoConverter;


    @PostMapping("/scraping/")
    public ResponseEntity<List<Produto_MercadoEntity>> getScraping() throws SQLException {
        // Chama o método de web scraping em cada serviço e obtenha as listas de produtos
        List<Produto_MercadoEntity> produtosCooper = cooperScrapper.scrapeProducts();
        List<Produto_MercadoEntity> produtosKoch = kochScrapper.scrapeProducts();

        // Combina as duas listas em uma
        List<Produto_MercadoEntity> allProdutos = new ArrayList<>();
        allProdutos.addAll(produtosCooper);
        allProdutos.addAll(produtosKoch);

        // Passa cada produto da lista combinada para o método addProduto()
        for (Produto_MercadoEntity produtoEntity : allProdutos) {
            produtoMercadoDAO.addProduto(produtoEntity);
        }
        return ResponseEntity.status(HttpStatus.CREATED).build();
    }


    @GetMapping
    public ResponseEntity<List<Produto_MercadoDTO>> getProdutos() throws SQLException {

        return ResponseEntity.ok().body(produtoConverter.toDTO(produtoMercadoDAO.getAll()));

    }
```

```
    @GetMapping("/autocomplete/")
        public ResponseEntity<List<Produto_MercadoDTO>> autocomplete(@RequestParam String
searchTerm) throws SQLException {
                                                                                    return
ResponseEntity.ok().body(produtoConverter.toDTO(produtoMercadoDAO.autocomplete(searchTer
m)));
    }

    @GetMapping("/produtos/")
    public ResponseEntity<List<Produto_MercadoDTO>> getUniqueProdutos(
            @RequestParam(defaultValue = "1") int page,
            @RequestParam(defaultValue = "20") int size) throws SQLException {

                                                    List<Produto_MercadoDTO>        produtos        =
produtoConverter.toDTO(produtoMercadoDAO.getUniqueProdutos(page, size));
        return ResponseEntity.ok().body(produtos);
    }

    @GetMapping("/grupo/{id_grupo}")
        public ResponseEntity<List<Produto_MercadoDTO>> getProdutosPorGrupo(@PathVariable
ProdutoEntity id_grupo) throws SQLException {

                                                                                    return
ResponseEntity.ok().body(produtoConverter.toDTO(produtoMercadoDAO.getProdutosPorGrupo(id
_grupo)));
    }

    @PostMapping("/gpt/")
    public ResponseEntity<String> sentGPT() throws SQLException {
        int loteSize = 200;

        // Obtenha o total de produtos do banco de dados
        int totalProdutos = produtoMercadoDAO.getTotalProdutos();
        String response = null;

        for (int i = 0; i < totalProdutos; i += loteSize) {
            // Obtenha o lote de produtos do banco de dados usando LIMIT e OFFSET
                                                    List<Produto_MercadoEntity>     loteProdutos     =
produtoMercadoDAO.getByMercado(loteSize, i);

            // Cria o prompt para a API da OpenAI com o lote de produtos
            String promptListaProduto = criarPromptListaProduto(loteProdutos);

            // Envia o prompt para a API da OpenAI e recebe a resposta
            response = chatGPT.getOpenAIResponse(promptListaProduto);

            // Desativa envio para API GPT
            //response = "{\"choices\": []}";
```

```
            // Processa a resposta da API
            criaGruposProdutosDaAPI(response);
        }

        return ResponseEntity.status(HttpStatus.CREATED).body(response);
    }

    public String criarPromptListaProduto(List<Produto_MercadoEntity> produtos) {
        StringBuilder promptListaProduto = new StringBuilder();
        for (Produto_MercadoEntity produto : produtos) {
                String nomeProduto = produto.nome.toLowerCase(); // remove o volume do nome do
produto

            promptListaProduto.append("\\n")
                    .append("id_produto: ")
                    .append(produto.id_produto_mercado)
                    .append(", nome: ")
                    .append(nomeProduto.trim()); // adicionado trim() para remover espaços em branco que
podem ter sido deixados após a remoção do volume

//            System.out.println(nomeProduto.trim()); // adicionado trim() aqui também
//            System.out.println(" Volume: " + produto.volume);
              System.out.println("ID produto: " + produto.id_produto_mercado);


        }
        return promptListaProduto.toString();
    }

    public void criaGruposProdutosDaAPI(String response) {
        try {
            ObjectMapper mapper = new ObjectMapper();
            JsonNode rootNode = mapper.readTree(response);
            JsonNode firstChoiceNode = rootNode.path("choices").get(0);
            String conteudoResposta = firstChoiceNode.path("message").get("content").asText();
                conteudoResposta = conteudoResposta.substring(conteudoResposta.indexOf('\n') + 1); //
remove first line
              conteudoResposta = conteudoResposta.substring(0, conteudoResposta.lastIndexOf('\n')); //
remove last line
            JsonNode gruposDeProdutoMercado = mapper.readTree(conteudoResposta);

            for (JsonNode grupo : gruposDeProdutoMercado) {
                ProdutoDTO novoProduto = criaGrupoProdutosDoMercado(grupo);
                ProdutoConverter produtoConverter = new ProdutoConverter();
                ProdutoEntity entity = produtoConverter.toEntity(novoProduto);
                ProdutoDAO produtoDAO = new ProdutoDAO();
                produtoDAO.addGrupo(entity);
                System.out.println("novoProduto" + novoProduto);
                atualizaIdProdutoDoGrupo(grupo, novoProduto);
            }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private ProdutoDTO criaGrupoProdutosDoMercado(JsonNode grupo) throws SQLException {
        ProdutoDTO novoGrupoProdutoMercado = new ProdutoDTO();
        String idGrupoUUID = UUID.randomUUID().toString();
        novoGrupoProdutoMercado.setId(idGrupoUUID);
        JsonNode idProdutos = grupo.get("id_produto");
        if (idProdutos != null && idProdutos.isArray() && !idProdutos.isEmpty()) {
            int idPrimeiroProduto = idProdutos.get(0).asInt();
                                                        Produto_MercadoDTO produto =
produtoConverter.toDTO(produtoMercadoDAO.getProduto(idPrimeiroProduto));
            if (produto != null && produto.nome != null) {
                novoGrupoProdutoMercado.setNome(produto.nome);
            }
        }
        return novoGrupoProdutoMercado;
    }

      private void atualizaIdProdutoDoGrupo(JsonNode grupo, ProdutoDTO novoProduto) throws
SQLException {
        JsonNode idsProdutoMercado = grupo.get("id_produto");
        if (idsProdutoMercado != null && idsProdutoMercado.isArray()) {
            for (JsonNode idProduto_nercado : idsProdutoMercado) {
                System.out.println("ID do Produto_Mercado: " + idProduto_nercado.asInt());
                produtoMercadoDAO.updateIdProduto(idProduto_nercado.asInt(), novoProduto.getId());
            }
        }
    }

}
```

—---

```
package com.cedup.super_preco.controller.produto;

import com.cedup.super_preco.model.produto.ProdutoEntity;
import com.cedup.super_preco.model.produto.Produto_MercadoEntity;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.stream.Collectors;

@Component
public class ProdutoConverter {

    public ProdutoEntity toEntity(ProdutoDTO dto) {
        return new ProdutoEntity(dto.id, dto.nome);
    }
```

```
    public List<Produto_MercadoDTO> toDTO(List<Produto_MercadoEntity> entities) {

        return entities //
                .stream() //
                                .map(entity -> new Produto_MercadoDTO(entity.id_produto_mercado,
entity.id_mercado.id_mercado, entity.id_produto.id_produto, entity.nome, entity.preco, entity.link,
entity.link_img)) //
                .collect(Collectors.toList());
    }

    public Produto_MercadoDTO toDTO(Produto_MercadoEntity entity) {
         return new Produto_MercadoDTO(entity.id_produto_mercado, entity.id_mercado.id_mercado,
entity.id_produto.id_produto, entity.nome, entity.preco, entity.link, entity.link_img);
    }


}
```

—----------

```
package com.cedup.super_preco.controller.produto;

import java.util.List;

public class ProdutoDTO {
    public String id;
    public String nome;
    public List<Integer> idProdutos; // nova lista de IDs de produtos

    public ProdutoDTO(String id, String nome) {
        this.id = id;
        this.nome = nome;
    }

    public ProdutoDTO() {
    }

    @Override
    public String toString() {
        return "Grupo ID: " + id + ", Nome: " + nome;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }
```

```
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public List<Integer> getIdProdutos() {
        return idProdutos;
    }

    public void setIdProdutos(List<Integer> idProdutos) {
        this.idProdutos = idProdutos;
    }
}
```

—----------

```
package com.cedup.super_preco.model.mercado;

public class MercadoEntity {
    public int id_mercado;
    public String nome;

    public MercadoEntity(int id_mercado) {
        this.id_mercado = id_mercado;
    }

    public MercadoEntity() {
    }

    public int getId_mercado() {
        return id_mercado;
    }

    public void setId_mercado(int id_mercado) {
        this.id_mercado = id_mercado;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

—----------

```
package com.cedup.super_preco.model.produto;
```

```
import com.cedup.super_preco.ConnectionSingleton;
import com.cedup.super_preco.model.mercado.MercadoEntity;
import org.springframework.stereotype.Component;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

@Component
public class Produto_MercadoDAO {
    public List<Produto_MercadoEntity> getAll() throws SQLException {
        List<Produto_MercadoEntity> produtos = new ArrayList<>();

        String sql = "SELECT * FROM produto_mercado";
        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                int id_produto_mercado = rs.getInt("id_produto_mercado");
                MercadoEntity id_mercado = new MercadoEntity(rs.getInt("id_mercado"));
                ProdutoEntity id_produto = new ProdutoEntity(rs.getString("id_produto"));
                String nome = rs.getString("nome");
                double preco = rs.getDouble("preco");
                String link = rs.getString("link");
                String link_img = rs.getString("link_img");
                produtos.add(new Produto_MercadoEntity(id_produto_mercado, id_mercado, id_produto,
nome, preco, link,
                        link_img));
            }
            return produtos;
        }
    }

    public List<Produto_MercadoEntity> autocomplete(String searchTerm) throws SQLException {
        List<Produto_MercadoEntity> produtos = new ArrayList<>();

        String sql = "SELECT pm.* " +
                "FROM produto_mercado pm " +
                "INNER JOIN " +
                "(SELECT id_produto, MIN(id_produto_mercado) as id_produto_mercado " +
                "FROM produto_mercado " +
                "GROUP BY id_produto) subquery " +
                "ON pm.id_produto = subquery.id_produto " +
                "AND pm.id_produto_mercado = subquery.id_produto_mercado " +
                "WHERE pm.nome LIKE ?";

        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql)) {
```

```
            stmt.setString(1, "%" + searchTerm + "%");
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                int id_produto_mercado = rs.getInt("id_produto_mercado");
                MercadoEntity id_mercado = new MercadoEntity(rs.getInt("id_mercado"));
                ProdutoEntity id_produto = new ProdutoEntity(rs.getString("id_produto"));
                String nome = rs.getString("nome");
                double preco = rs.getDouble("preco");
                String link = rs.getString("link");
                String link_img = rs.getString("link_img");
                produtos.add(new Produto_MercadoEntity(id_produto_mercado, id_mercado,
                        id_produto, nome, preco, link, link_img));
            }
        }
        return produtos;
    }

        public List<Produto_MercadoEntity> getUniqueProdutos(int page, int size) throws
SQLException {
        List<Produto_MercadoEntity> produtos = new ArrayList<>();

        String sql = "SELECT pm.* " +
                "FROM produto_mercado pm " +
                "INNER JOIN " +
                "(SELECT id_produto, MIN(id_produto_mercado) as id_produto_mercado " +
                "FROM produto_mercado " +
                "GROUP BY id_produto) subquery " +
                "ON pm.id_produto = subquery.id_produto " +
                "AND pm.id_produto_mercado = subquery.id_produto_mercado " +
                "ORDER BY pm.id_produto LIMIT ? OFFSET ?";

        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql)) {
            stmt.setInt(1, size);
            stmt.setInt(2, (page - 1) * size);
            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    int id_produto_mercado = rs.getInt("id_produto_mercado");
                    MercadoEntity id_mercado = new MercadoEntity(rs.getInt("id_mercado"));
                    ProdutoEntity id_produto = new ProdutoEntity(rs.getString("id_produto"));
                    String nome = rs.getString("nome");
                    double preco = rs.getDouble("preco");
                    String link = rs.getString("link");
                    String link_img = rs.getString("link_img");
                            produtos.add(new Produto_MercadoEntity(id_produto_mercado, id_mercado,
id_produto, nome, preco, link,
                            link_img));
                }
            }
        }
        return produtos;
```

```
    }

    public List<Produto_MercadoEntity> getByMercado(int limit, int offset) throws SQLException {
        List<Produto_MercadoEntity> produtos = new ArrayList<>();

            String sql = "SELECT id_produto_mercado, id_mercado, nome FROM produto_mercado
ORDER BY nome LIMIT ? OFFSET ?";
        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql)) {
            stmt.setInt(1, limit);
            stmt.setInt(2, offset);
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                int id_produto_mercado = rs.getInt("id_produto_mercado");
                MercadoEntity id_mercado = new MercadoEntity(rs.getInt("id_mercado"));
                String nome = rs.getString("nome");
                produtos.add(new Produto_MercadoEntity(id_produto_mercado, id_mercado, nome));
                // System.out.println(nome);
            }
            System.out.println("limit: " + limit);
            System.out.println("offset: " + offset);

System.out.println("_____
_____");
        }
        return produtos;
    }

    public int getTotalProdutos() throws SQLException {
        String sql = "SELECT COUNT(*) FROM produto_mercado";
        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return rs.getInt(1);
            } else {
                return 0;
            }
        }
    }

    public int getTotalGrupos() throws SQLException {
        String sql = "SELECT COUNT(*) FROM produto";
        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return rs.getInt(1);
            } else {
                return 0;
            }
        }
    }
```

```
    public Produto_MercadoEntity getProduto(int id) throws SQLException {
        Produto_MercadoEntity produto = null;
        String sql = "SELECT * FROM produto_mercado where id_produto_mercado = ?";

        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql)) {
            stmt.setInt(1, id);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    int id_produto_mercado = rs.getInt("id_produto_mercado");
                    MercadoEntity id_mercado = new MercadoEntity(rs.getInt("id_mercado"));
                    ProdutoEntity id_produto = new ProdutoEntity(rs.getString("id_produto"));
                    String nome = rs.getString("nome");
                    double preco = rs.getDouble("preco");
                    String link = rs.getString("link");
                    String link_img = rs.getString("link_img");
                     produto = new Produto_MercadoEntity(id_produto_mercado, id_mercado, id_produto,
nome, preco, link, link_img);
                }
            }
            return produto;
        }
    }

        public List<Produto_MercadoEntity> getProdutosPorGrupo(ProdutoEntity entity) throws
SQLException {
        List<Produto_MercadoEntity> produtos = new ArrayList<>();

        String sql = "SELECT * FROM produto_mercado WHERE id_produto = ?";
        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql)) {
            stmt.setString(1, entity.id_produto);
            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    int id_produto_mercado = rs.getInt("id_produto_mercado");
                    MercadoEntity id_mercado = new MercadoEntity(rs.getInt("id_mercado"));
                    String nome = rs.getString("nome");
                    double preco = rs.getDouble("preco");
                    String link = rs.getString("link");
                    String link_img = rs.getString("link_img");
                        produtos.add(new Produto_MercadoEntity(id_produto_mercado, id_mercado, entity,
nome, preco, link,
                            link_img));
                }
            }
        }
        return produtos;
    }

    public void addProduto(Produto_MercadoEntity produto) throws SQLException {
```

```
        String sql = "INSERT INTO produto_mercado (id_mercado, id_produto, nome, preco, link, link_img) VALUES (?, ?, ?, ?, ?, ?)";

        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql,
                Statement.RETURN_GENERATED_KEYS)) {
            stmt.setInt(1, produto.id_mercado.id_mercado);
            stmt.setString(2, produto.id_produto.id_produto);
            stmt.setString(3, produto.nome);
            stmt.setDouble(4, produto.preco);
            stmt.setString(5, produto.link);
            stmt.setString(6, produto.link_img);

            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
                rs.next();
                produto.id_produto_mercado = rs.getInt(1);
            }
        }
    }

    public void updateIdProduto(int idProdutoMercado, String idProduto) throws SQLException {
        String sql = "UPDATE produto_mercado SET id_produto = ? WHERE id_produto_mercado = ?";
        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql)) {
            stmt.setString(1, idProduto);
            stmt.setInt(2, idProdutoMercado);
            stmt.executeUpdate();
        }
    }
}
```

```
package com.cedup.super_preco.model.produto;

import com.cedup.super_preco.model.mercado.MercadoEntity;

public class Produto_MercadoEntity {
    public int id_produto_mercado;
    public MercadoEntity id_mercado;
    public ProdutoEntity id_produto;
    public String nome;
    public double preco;
    public String link;
    public String link_img;


        public Produto_MercadoEntity(int id_produto_mercado, MercadoEntity id_mercado, ProdutoEntity id_produto, String nome, double preco, String link, String link_img) {
        this.id_produto_mercado = id_produto_mercado;
```

```
        this.id_mercado = id_mercado;
        this.id_produto = id_produto;
        this.nome = nome;
        this.preco = preco;
        this.link = link;
        this.link_img = link_img;
    }

        public Produto_MercadoEntity(int id_produto_mercado, MercadoEntity id_mercado, String
nome) {
        this.id_produto_mercado = id_produto_mercado;
        this.id_mercado = id_mercado;
        this.nome = nome;
    }

    public Produto_MercadoEntity() {
    }

}
```

—----------

```
package com.cedup.super_preco.model.produto;

import com.cedup.super_preco.ConnectionSingleton;
import org.springframework.stereotype.Component;

import java.sql.PreparedStatement;
import java.sql.SQLException;

@Component
public class ProdutoDAO {

    public void addGrupo(ProdutoEntity grupo) throws SQLException {
        String sql = "INSERT INTO produto (id_produto, nome) VALUES (?, ?)";
        try (PreparedStatement stmt = ConnectionSingleton.getConnection().prepareStatement(sql)) {
            stmt.setString(1, grupo.id_produto);
            stmt.setString(2, grupo.nome);
            stmt.executeUpdate();
        }
    }
}
```

—----------

```
package com.cedup.super_preco.model.produto;

public class ProdutoEntity {
    public String id_produto;
    public String nome;

    public ProdutoEntity(String id_produto, String nome) {
```

```
        this.id_produto = id_produto;
        this.nome = nome;
    }

    public ProdutoEntity() {
    }

    public ProdutoEntity(String id_produto) {
        this.id_produto = id_produto;
    }
}
```

—----------

```
package com.cedup.super_preco;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionSingleton {
    private static Connection connection;

    private ConnectionSingleton() {
        // Singleton class
    }

    /**
     * Obtém a conexão ativa com o banco.
     * Caso não exista nenhuma conexão ativa ainda, cria uma nova.
     */
    public static Connection getConnection() throws SQLException {

        if (connection == null || connection.isClosed()) {
            connection = DriverManager.getConnection( //

"jdbc:mysql://database-tcc.ck3iuzwtvyfk.sa-east-1.rds.amazonaws.com:3306/super_preco", //
                    "admin", //
                    "f9Y4J8XIIEeI");
        }
        return connection;
    }
}
```

—----------

```
package com.cedup.super_preco;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
```

```
public class SuperPrecoApplication {

	public static void main(String[] args) {
		SpringApplication.run(SuperPrecoApplication.class, args);
	}

}
```

—----------