Universidade do Minho
Escola de Engenharia

José Pedro Vilela Bravo

Framework **Para a Criação de Lojas** ***Online***

outubro de 2023

Framework **Para a Criação de Lojas** ***Online***

José Pedro Vilela Bravo

UMinho | 2023

Universidade do Minho
Escola de Engenharia

José Pedro Vilela Bravo

Framework Para a Criação de Lojas *Online*

Dissertação de Mestrado
Mestrado Integrado em Engenharia de Telecomunicações e Informática

Trabalho efetuado sob a orientação do
Professor Doutor José Manuel Tavares Vieira Cabral e Professor Doutor Sérgio Adriano Fernandes Lopes

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Agradecimentos

A concretização desta dissertação foi possível devido ao apoio incansável e fundamental de diversas pessoas, às quais desejo expressar o meu mais profundo agradecimento.

Em primeiro lugar, gostaria de expressar a minha gratidão aos meus irmãos, pais e avós, cuja educação sólida, motivação incessante e apoio incondicional foram alicerces essenciais ao longo desta etapa da minha vida académica.

Aos meus orientadores, o Professor José Manuel Tavares Vieira Cabral e o Professor Sérgio Adriano Fernandes Lopes, desejo expressar o meu reconhecimento pela sua disponibilidade constante, paciência incansável e partilha valiosa de sugestões ao longo de todo o processo de elaboração desta dissertação.

Por último, gostaria de estender os meus agradecimentos a todos os meus amigos, pelo apoio contínuo e presença nos momentos mais desafiantes.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

Atualmente, para se obter um modelo de negócio de sucesso, é necessário um investimento no comércio eletrónico e nas tecnologias inerentes. No desenvolvimento de uma loja *online*, é necessário que seja efetuada uma estruturação da mesma, de forma que o utilizador final consiga encontrar os produtos e/ou serviços desejados efetuando o menor número de cliques, tornando este processo o mais simples e cómodo possível. De forma semelhante, o processo de criação deste tipo de plataformas deve ser um processo relativamente simples e acessível a qualquer individuo.

Neste contexto, surge a plataforma para geração simples e intuitiva de lojas *online* consoante as preferências do utilizador. O funcionamento da plataforma consiste na manipulação e integração de um conjunto de componentes customizáveis ou *templates*, devidamente generalizados, de modo a ser possível gerar uma loja *online* com as funcionalidades e propriedades definidas pelo utilizador. O objetivo é o desenvolvimento de uma aplicação de generalização que permita automatizar o processo de criação de lojas *online*.

O resultado deste projeto permitirá a um programador *web* criar uma loja *online* de forma rápida, através do *download* do código fonte da loja gerada, tendo apenas de efetuar uma parametrização de um conjunto de atributos específicos consoante o caso concreto a implementar, de modo a satisfazer as necessidades dos clientes.

Palavras-chave: estruturação, *framework*, desenvolvimento *web*, *e-commerce*

Abstract

Currently, to achieve a successful business model, investment in e-commerce and the technologies inherent to it is necessary. With technological evolution, the development of online commerce is a necessity to survive in such a competitive market. Adding to this reality the fact that there are increasingly more users of these platforms around the world, the need to make online purchases securely assumes new importance.

In the development of an online store, it is necessary to structure it in such a way that the end user can find the desired products and/or services with the least number of clicks, making this process as simple and comfortable as possible. Similarly, the process of creating these types of platforms must be a relatively simple process accessible to any individual. The platform's operation involves the manipulation and integration of a set of templates, sufficiently generalized to enable the creation of an online store with the functionalities and properties defined by the user. The objective is the development of a generalized application that automates the process of creating online stores.

In this context, the application of generalization arises for the development of online stores. The goal is the development of a generalization application that allows automating the process of creating online stores. The result of this project will allow a web programmer to quickly create an online store, having only to carry out a parameterization of a set of specific attributes according to the concrete case to be implemented, in order to meet the needs of the customers.

Keywords: structuring, framework, web development, e-commerce.

Índice de conteúdos

Lista de figuras

Lista de tabelas

Lista de acrónimos

AJAX	*Asynchronous JavaScript and XML*
API	*Application Programming Interface*
API REST	*Representational State Transfer Application Programming Interface*
CMS	*Content Management System*
CRUD	*Create Read Update Delete*
CRM	*Customer Relationship Management*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
E/S	*Entrada/Saída*
ERP	*Enterprise Resource Planning*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
JS	*JavaScript*
JSON	*JavaScript Object Notation*
JWT	*JSON Web Token*
MERN	*MongoDB, Express, React, Node.js*
NPM	*Node Package Manager*
REST	*Representational State Transfer*
SDK	*Software Development Kit*

SEO	*Search Engine Optimization*
SPA	*Single Page Application*
SQL	*Structured Query Language*
SSL	*Secure Sockets Layer*
UI	*User Interface*
UX	*User Experience*

1. Introdução

No atual panorama do comércio eletrónico, onde a presença digital é fundamental para o êxito comercial, deparamo-nos com desafios significativos para aqueles que procuram criar uma loja *online*. Desenvolver uma plataforma de comércio eletrónico robusta e eficaz pode ser uma tarefa árdua e demorada, especialmente para programadores e desenvolvedores que desejam proporcionar experiências de compra distintas aos seus clientes. As aplicações existentes podem oferecer soluções parciais, mas muitas vezes carecem da flexibilidade necessária para satisfazer as diversas necessidades das empresas modernas. À medida que o comércio eletrónico continua a prosperar, a necessidade de integração e interligação de diferentes soluções aumenta.

Esta dissertação faz parte de um projeto colaborativo que se propõe criar uma plataforma abrangente para o desenvolvimento de lojas *online*. Esta plataforma visa proporcionar uma infraestrutura sólida e flexível para a construção de soluções de comércio eletrónico eliminando a necessidade de começar do zero a cada nova implementação, permitindo a reutilização de código e acelerando significativamente o processo de desenvolvimento.

A ideia fundamental por de trás da plataforma é a de oferecer um ambiente estruturado onde programadores e desenvolvedores possam criar lojas *online* com eficiência e agilidade. A plataforma pode, conceptualmente, ser dividida em duas partes distintas: a *framework e* a ferramenta de geração de lojas. A *framework* engloba um conjunto de componentes customizáveis ou *templates*, cada um destinado a tratar de aspetos específicos da experiência de compra *online*, desde a gestão de produtos até à finalização do processo de encomendas. A ferramenta de geração de lojas consiste numa aplicação *React* que utiliza os *templates* disponibilizados pela *framework,* efetuando as alterações necessárias de acordo com as preferências do utilizador.

Em trabalhos anteriores, foram abordadas as funcionalidades de gestão de produtos, permitindo uma administração eficiente dos itens disponíveis nas lojas *online*, bem como a funcionalidade de gestão de utilizadores, simplificando a gestão e supervisão das contas de clientes e administradores na plataforma. Cada membro da equipa concentra-se em desenvolver

um componente específico, contribuindo para uma perspetiva detalhada e especializada do todo. Este trabalho tem como objetivo dar continuidade a esse processo, com um foco particular nas funcionalidades de encomendas e na gestão do carrinho de compras, partes essenciais da experiência de compra em qualquer plataforma de comércio eletrónico.

Este capítulo introdutório não só apresenta a *framework* como um todo, mas também salienta o contributo individual deste trabalho para o aperfeiçoamento da experiência do utilizador e a eficácia das operações proporcionadas por esta plataforma.

1.1 Objetivos

O principal objetivo do desenvolvimento da presente dissertação é a criação de componentes essenciais, nomeadamente gestão de encomendas, carrinho de compras e pagamentos, visando automatizar e simplificar a criação de lojas *online*. Este processo culmina na conceção de uma *framework* abrangente para o desenvolvimento de lojas *online*, que visa proporcionar uma infraestrutura sólida e flexível para a construção de soluções de comércio eletrónico. Neste sentido, foram definidos os seguintes objetivos:

1. Conceção de uma loja *online* baseada em tecnologias *MERN (MongoDB, ExpressJS, React e NodeJS)* [1] utilizando componentes devidamente generalizados de forma a facilitar a sua posterior integração. A loja *online* desenvolvida deve disponibilizar as seguintes funcionalidades:
    - Autenticação e Gestão de Utilizadores
    - *BackOffice* para administradores do sistema
    - Gestão de Encomendas, carrinho de compras e pagamentos

2. Desenvolvimento de uma plataforma de geração de lojas *online*, que utiliza componentes generalizados para cada funcionalidade, sendo que deve permitir:
    - Descarregar o código-fonte da loja *online* gerada de forma automática para posterior customização.

- A integração opcional dos diferentes componentes desenvolvidos anteriormente, com foco particular nos componentes do carrinho de compras, encomendas e pagamentos.
- Documentar as parametrizações e procedimentos necessários para a integração de cada componente numa loja *online*.

1.2 Estrutura da dissertação

Nesta secção, são enumeradas e descritas as etapas e os objetivos envolvidos na implementação da loja *online* e da *framework* para criação de lojas.

O segundo capítulo, denominado Estado da Arte, concentra-se na revisão da literatura e no estudo de trabalhos relacionados. São analisadas algumas soluções e plataformafas existentes para lojas *online* e é efetuada uma breve revisão relativa às tecnologias utilizadas no desenvolvimento da loja *online.*

No terceiro capítulo, intitulado Desenvolvimento de uma Loja *Online*, são apresentados os requisitos do sistema, definindo as funcionalidades e as especificações necessárias para o correto funcionamento da loja. É detalhada a arquitetura do sistema, incluindo o modelo de dados, a API e os *middlewares* utilizados para proteger as rotas e garantir a autenticação e autorização dos utilizadores. Além disso, é explicada a estrutura do código, a organização e a hierarquia de páginas no *frontoffice* e *backoffice* da aplicação.

O quarto capítulo aborda a plataforma para criação de lojas onde são exploradas as funcionalidades do sistema e como essas funcionalidades são suportadas pela *framework* desenvolvida. É detalhada a arquitetura da *framework*, incluindo a API e o modelo de dados utilizados para a implementação das lojas *online*. Além disso, é descrito o funcionamento da ferramenta responsável por manipular os componentes customizáveis disponibilizados pela *framework*

No quinto capítulo, é efetuada uma demonstração da utilização da plataforma apresentando o processo de configuração, *download* e instalação de uma loja gerada pela ferramenta.

Por fim, no sexto capítulo, são discutidos os resultados obtidos ao longo da dissertação. São analisados os desafios enfrentados e as soluções encontradas durante o desenvolvimento da

loja *online* e sobretudo da *framework*. São apresentadas as conclusões finais do trabalho, enfatizando a contribuição do estudo para o campo das lojas *online* e as possíveis direções para trabalhos futuros.

2. Estado da arte

Neste capítulo é efetuada uma revisão da literatura, analisando e comparando aplicações e trabalhos relacionados disponíveis, sendo que é efetuada uma breve explicação relativa às tecnologias utilizadas no desenvolvimento da presente dissertação.

2.1 Plataformas existentes

Nesta secção é efetuada uma breve análise e descrição de algumas plataformas existentes permitindo uma compreensão mais abrangente das necessidades tecnológicas e operacionais que a plataforma se propõe a resolver.

2.1.1 Shopify

A *Shopify [2]* é uma plataforma de *e-commerce* que permite aos utilizadores criarem e gerirem facilmente a sua própria loja *online*. Oferece uma ampla variedade de opções de personalização e integrações com diversos métodos de pagamento e envio, além de fornecer uma plataforma de gestão de stocks e relatórios de vendas. Além disso, a plataforma também oferece opções de venda em *marketplaces* e redes sociais, bem como a possibilidade de criar aplicações personalizadas para expandir as funcionalidades da loja.

As vantagens da utilização da *Shopify* incluem a facilidade de utilização, a possibilidade de personalização, a ampla gama de integrações disponíveis e o suporte a vários canais de venda. Além disso, também oferece opções de pagamento flexíveis e uma plataforma de pagamento própria, o que pode simplificar o processo de *checkout* para os utilizadores.

As desvantagens da utilização da *Shopify* incluem o custo mensal, que pode ser considerado alto para alguns utilizadores. Além disso, também cobra taxas por transações, o que pode afetar as margens de lucro dos utilizadores.

2.1.2 Wix eCommerce

A *Wix eCommerce [3]* é uma plataforma de criação de lojas *online* que permite aos utilizadores criarem facilmente a sua própria loja virtual, sem necessidade de conhecimentos técnicos avançados. Oferece uma ampla variedade de modelos pré-criados e opções de personalização, bem como integrações com diversos métodos de pagamento e envio. Inclui ainda uma plataforma de gestão de stocks e relatórios de vendas.

As vantagens da utilização da *Wix eCommerce* incluem a facilidade de utilização, a possibilidade de personalização com muitas opções de modelos pré-criados, e a integração com vários métodos de pagamento e envio. Esta plataforma oferece ainda opções de venda em *marketplaces* e redes sociais, bem como a possibilidade de criar aplicações personalizadas para expandir as funcionalidades da loja.

As desvantagens da utilização da *Wix eCommerce* incluem o facto de que as opções de personalização são limitadas em comparação com outras plataformas de *e-commerce*. Além disso, a *Wix eCommerce* pode não disponibilizar todas as funcionalidades avançadas disponíveis noutras plataformas de *e-commerce*, como a possibilidade de criar promoções complexas ou integrações com aplicações externas. Alguns utilizadores também podem sentir falta de opções de pagamento mais flexíveis ou de uma plataforma de pagamento própria.

2.1.3 WordPress

O *WordPress [4]* é uma plataforma de publicação de conteúdo que também oferece funcionalidades de *e-commerce* através de *plugins* específicos. A sua versão mais recente, permite aos utilizadores adicionar produtos, processar pagamentos e gerir os stocks da loja *online*. Além disso, o *WordPress* oferece uma ampla variedade de temas pré-concebidos e opções de personalização, assim como integrações com diversos métodos de pagamento e envio.

As vantagens da utilização do *WordPress* para criar uma loja *online* incluem a sua facilidade de uso e a possibilidade de criar um site profissional com recursos avançados, como *blogs* e páginas de destino. Além disso, o *WordPress* também oferece opções de SEO avançadas, o que podem ajudar a melhorar o posicionamento do site nos motores de busca.

As desvantagens da utilização do *WordPress* como plataforma de *e-commerce* incluem a necessidade de instalar e configurar *plugins* adicionais, como o *WooCommerce*, para adquirir as funcionalidades de loja *online*. Além disso, a segurança do site pode ser um problema se não for devidamente controlada, e os utilizadores podem encontrar dificuldades em personalizar certos aspetos da loja *online* sem conhecimentos de programação.

2.1.4 Magento

Magento [5] é uma plataforma de comércio eletrónico de código aberto, amplamente reconhecida pela sua escalabilidade. Oferece uma vasta gama de recursos e funcionalidades que permitem aos utilizadores criar lojas *online* altamente personalizadas e adaptadas às suas necessidades específicas.

Uma das principais vantagens da utilização do *Magento* é a sua alta escalabilidade, permitindo que a loja *online* cresça à medida que o negócio se expande, suportando grandes volumes de tráfego e transações. Além disso, o *Magento* oferece recursos avançados de personalização, possibilitando que os utilizadores ajustem o tema e as funcionalidades da loja de acordo com as necessidades.

A plataforma *Magento* também se destaca pelo seu ecossistema de extensões e módulos, que disponibiliza diversos recursos adicionais, como métodos de pagamento, opções de envio, integração com sistemas de gestão empresarial (ERP), entre outros.

No entanto, é importante mencionar que o *Magento* pode ser mais complexo em comparação com outras plataformas, exigindo um conhecimento técnico mais avançado para a sua configuração e gestão.

2.1.5 BigCommerce

A *BigCommerce [6]* é uma plataforma de comércio eletrónico altamente escalável, destinada a médias e grandes empresas. Oferece uma solução robusta para a criação e gestão de lojas *online* personalizadas, respondendo às necessidades de empreendedores e empresas que pretendem expandir a sua presença no mundo digital.

Uma das principais vantagens da *BigCommerce* é a escalabilidade, tornando-se uma escolha sólida para empresas em crescimento que desejam uma plataforma capaz de lidar com grandes volumes de tráfego e transações. Além disso, a *BigCommerce* oferece uma ampla variedade de opções de personalização, permitindo que os utilizadores ajustem facilmente o aspeto e as funcionalidades da loja de acordo com as suas preferências e necessidades específicas.

A utilização da *BigCommerce* é relativamente acessível, especialmente para utilizadores com algum conhecimento prévio em desenvolvimento de lojas *online*. A plataforma apresenta uma interface intuitiva, simplificando a configuração e gestão da loja. Além disso, fornece recursos abrangentes de suporte técnico e uma vasta base de conhecimento, garantindo apoio e orientação aos utilizadores em todas as etapas do processo de criação e gestão da loja *online*.

Em suma, a *BigCommerce* é uma plataforma poderosa e altamente escalável, ideal para empresas que procuram criar e gerir lojas *online* personalizadas. As suas vantagens incluem a facilidade de escalabilidade, personalização e suporte ao cliente. No entanto, as desvantagens incluem o custo e a complexidade para utilizadores menos experientes.

2.1.6 Análise e comparação dos trabalhos relacionados

Nesta secção é efetuada uma comparação dos trabalhos relacionados abordados na presente dissertação. A análise das diferentes soluções é baseada em diversas categorias e/ou parâmetros considerados como essenciais na avaliação de plataformas deste tipo. A Tabela 1 visa sintetizar a comparação das diferentes ferramentas, nas diversas áreas de interesse.

Tabela 1 - Análise comparativa das plataformas selecionadas

Plataforma	Escalabilidade	Personalização	Utilização	Suporte	Segurança	Preço
Shopify	Elevada	Alta	Simples	Bom	Bom	Pago
Wix	Média	Alta	Simples	Bom	Bom	Pago
WordPress	Alta	Alta	Média	Bom	Bom	Gratuito
Magento	Alta	Alta	Complexa	Variável	Bom	Pago
BigCommerce	Elevada	Alta	Simples	Bom	Bom	Pago

Em termos de escalabilidade, o *Shopify, Magento e BigCommerce* destacam-se como as plataformas mais robustas, adequadas para empresas de médio a grande porte. Estas têm a capacidade de lidar com grandes volumes de tráfego e transações, tornando-as ideais para negócios em crescimento. A *Wix* e o *WordPress* (com o *plugin WooCommerce*) são mais adequados para pequenas e médias empresas, embora também ofereçam opções para expansão.

Relativamente à personalização, o *Magento* e o *WordPress* sobressaem, proporcionando um alto grau de flexibilidade e controlo sobre o *design* e funcionalidades da loja. A *Shopify* e a *BigCommerce* também oferecem opções de personalização, embora com algumas limitações em comparação com as plataformas mencionadas anteriormente. A *Wix* é mais direcionado para utilizadores menos técnicos, oferecendo modelos de *design* pré-concebidos e opções de personalização mais simples

Na facilidade de utilização, a *Wix* e a *Shopify* destacam-se, apresentando interfaces intuitivas e amigáveis, ideais para principiantes. O *WordPress* pode ser ligeiramente mais complexo para utilizadores menos técnicos, embora seja popular devido à sua familiaridade para muitos utilizadores. O *Magento* e a *BigCommerce* requerem mais conhecimentos técnicos para a configuração e gestão da loja, sendo mais adequados para utilizadores com experiência em desenvolvimento *web*.

Em resumo, cada plataforma tem as suas vantagens e desvantagens, e a escolha ideal dependerá das necessidades e objetivos específicos de cada negócio. As plataformas mais robustas, como a *Shopify, Magento e BigCommerce*, são indicadas para empresas em crescimento que exigem escalabilidade e funcionalidades avançadas. Por outro lado, o *Wix* e o *WordPress* (com o *plugin WooCommerce*) são mais indicados para pequenas e médias empresas com orçamentos limitados e que procurem facilidade de utilização e configuração mais rápida. A análise detalhada destas plataformas permitirá aos utilizadores fazerem uma escolha informada, alinhada com os seus objetivos de negócio e recursos disponíveis.

2.2 Revisão da literatura

Nesta secção, é efetuada uma revisão da literatura relevante, de modo a fornecer uma compreensão do contexto e das práticas atuais no campo do comércio eletrónico. Este estudo procura não apenas enquadrar o projeto, mas também identificar as tendências e lacunas que podem ser pertinentes para a solução proposta.

O artigo *"Analysis and Development of E-Commerce Web Application"* [8] descreve minuciosamente a implementação de uma aplicação de comércio eletrónico com base na arquitetura cliente-servidor, utilizando uma abordagem holística que envolve diversos componentes-chave. Na construção da interface do utilizador, o artigo faz uso do *bootstrap* [9]. No que diz respeito às operações de *backend*, o estudo recorre à *framework Django* [10], que oferece uma abordagem baseada em modelos para gerir dados, tornando a implementação de funcionalidades como a gestão de produtos e o processamento de pedidos mais organizada e coerente. Além disso, o artigo menciona a integração de um sistema de pagamento seguro, onde o *PayPal* [11] atua como um *gateway* de pagamento confiável para facilitar transações seguras entre os clientes e a plataforma.

O artigo *"Design and Implementation of B2B E-commerce Platform Based on Microservices Architecture"* [12] aborda os desafios enfrentados pelo comércio eletrónico *B2B* diante das transformações da indústria e propõe uma solução inovadora baseada em *microservices*. Esta abordagem visa a fragmentação do sistema em unidades independentes, cada uma dedicada a um serviço específico, como gestão de produtos, pedidos e pagamentos. Isto oferece escalabilidade e flexibilidade excecionais, permitindo o desenvolvimento, implementação e escalação independentes de cada serviço. A arquitetura de *microservices* também lida eficazmente com questões de escalabilidade e comunicação entre serviços, sendo que a solução proposta utilizou tecnologias como *Docker* [13] e *Kubernetes* [14] para a criação de elementos que hospedam os *microservices*, enquanto a *framework Spring Cloud* [15] facilitou o desenvolvimento e a integração desses serviços.

O artigo *"Microservices Enabled E-Commerce Web Application" [16]* aborda o desenvolvimento de uma aplicação *web* de *e-commerce* utilizando uma arquitetura de microserviços e a *stack* de tecnologias *MERN*. A aplicação é dividida em serviços autónomos

hospedados com *Docker* e manipulados através do *Kubernetes*, facilitando a sua gestão e escalabilidade. A comunicação entre os serviços é gerida por um servidor *NATS* [17], e a *MongoDB* é utilizado como base de dados. Em termos de conclusões, o artigo valida a eficácia da arquitetura de microserviços em aplicações de *e-commerce*. Sublinha a importância do *Docker* na simplificação do desenvolvimento, implementação e distribuição, mas também alerta para a necessidade de planeamento cuidadoso na estruturação dos microserviços. O estudo compara ainda o desempenho de arquiteturas monolíticas e de microserviços, concluindo que ambas podem ser eficazes, mas que a última oferece maior flexibilidade e facilidade de manutenção. Para aplicações menores, com menos de 100 utilizadores, sugere que uma abordagem monolítica pode ser mais eficiente.

2.3 Tecnologias utilizadas

No desenvolvimento desta dissertação, foi utilizado o conjunto de tecnologias *MERN,* ilustrado na Figura 1 (imagem obtida em [18]). Em conjunto, a *stack MERN* oferece uma solução completa para o desenvolvimento de aplicações *web*, desde o armazenamento de dados na *MongoDB* até a construção de interfaces interativas no *React*, passando pela gestão de rotas e pedidos com o *ExpressJS* e a execução do código *JavaScript* no servidor com o *Node.js*.

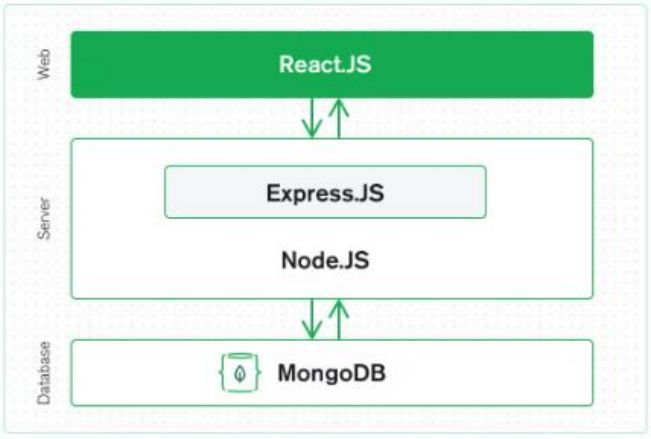

Figura 1 - Tecnologias *MERN*

Estas tecnologias são frequentemente combinadas devido à sua compatibilidade e capacidade de proporcionar uma experiência de desenvolvimento coerente e eficiente. Na próxima secção é efetuada uma breve explicação inerente a cada uma destas tecnologias.

2.3.1 MongoDB

MongoDB é um sistema de gestão de base de dados *NoSQL* orientado a documentos, amplamente utilizado no desenvolvimento de aplicações *web*. A sua abordagem baseada em documentos permite armazenar dados de forma flexível, sem a necessidade de estruturas de tabela rígidas. Os dados são organizados em documentos *JSON*, o que proporciona uma grande flexibilidade para lidar com esquemas dinâmicos e dados semiestruturados.

Uma das principais vantagens do *MongoDB* é a sua escalabilidade horizontal, que permite distribuir os dados por vários servidores, proporcionando maior capacidade de processamento e armazenamento. Além disso, oferece suporte a recursos avançados, como replicação automática, gestão de carga e consultas complexas, tornando-o adequado para aplicações que lidam com grandes volumes de dados.

Ao utilizar o *MongoDB* na *stack MERN*, podemos armazenar e gerir eficientemente os dados da aplicação *web*. A integração com o *Node.js* é direta, permitindo que a aplicação comunique de forma assíncrona e eficiente com a base de dados.

2.3.2 ExpressJS

O *ExpressJS* é uma *framework web* rápida e minimalista, construído sobre o Node.js. Esta fornece uma camada de abstração que facilita o desenvolvimento de servidores *web* robustos e escaláveis. Com o *ExpressJS*, é possível configurar rotas, gerir pedidos e respostas *HTTP* de forma eficiente, bem como implementar *middlewares* para a execução de lógica adicional.

A simplicidade e flexibilidade permitem que os desenvolvedores criem *APIs RESTful* e aplicações *web* de forma rápida e eficaz. Fornece uma estrutura modular que permite a adição de funcionalidades através de *plugins* e *middlewares* de terceiros. Além disso, o *ExpressJS* suporta a criação de servidores *HTTP* e *HTTPS*, tornando-o adequado para a implementação de aplicações seguras.

Ao integrar o *ExpressJS* na *stack* MERN, é possível criar uma camada de servidor eficiente e escalável para a aplicação *web*, garantindo um fluxo consistente de pedidos e respostas entre o cliente e o servidor.

2.3.3 React

O React é uma biblioteca *JavaScript* de código aberto, amplamente utilizada no desenvolvimento de interfaces de utilizador (UI). Com o *React*, é possível criar componentes reutilizáveis e organizar o código de forma eficiente, tornando o desenvolvimento de interfaces mais fácil e produtivo.

Uma das principais características do *React* é o seu modelo de programação baseado em componentes, que permite dividir a interface do utilizador em partes isoladas e independentes. Esses componentes podem ser combinados para construir interfaces complexas e dinâmicas, que respondem rapidamente às interações do utilizador. Além disso, o *React* utiliza a técnica de *Virtual DOM*, que otimiza o desempenho das aplicações atualizando apenas os elementos que foram modificados, em vez de recriar toda a interface. Isto resulta numa renderização mais rápida e uma experiência de utilização mais fluida.

Ao integrar o *React* na *stack MERN*, é possível desenvolver o *front-end* da aplicação *web* com uma abordagem modular e reativa. O *React* permite também efetuar o controlo de estados com bibliotecas como o *Redux*, facilitando o controlo e a manipulação dos dados na interface de utilizador.

2.3.4 NodeJS

O *Node.js* é um ambiente de execução *JavaScript* assíncrono e baseado em eventos, que permite a execução de código *JavaScript* no servidor. Utiliza um modelo de E/S não bloqueante, o que o torna eficiente e adequado para aplicações escaláveis e de alto desempenho.

Uma das principais vantagens do *Node.js* é a sua capacidade de lidar com um grande número de conexões simultâneas, tornando-o ideal para aplicações em tempo real, como *chats*, *streaming* de dados e jogos *online*. Além disso, o *Node.js* possui um ecossistema robusto de pacotes e bibliotecas, disponíveis através do gestor de pacotes *npm*, que facilita o desenvolvimento de aplicações com funcionalidades adicionais.

Ao utilizar o *Node.js* na *stack MERN*, é possível criar servidores *web* escaláveis e eficientes, aproveitando o poder do *JavaScript* tanto no lado do cliente como no lado do servidor. O *Node.js*

permite o desenvolvimento de *APIs RESTful*, manipulação de ficheiros, integração com bases de dados e outras tarefas do servidor de forma rápida e simplificada.

2.3.5 Redux

O *Redux [7]* é uma biblioteca de gestão de estados utilizada em aplicações *JavaScript*, especialmente em combinação com a *framework React*. Tem como principal objetivo centralizar o estado da aplicação, permitindo que todos os componentes acedam e partilhem os dados de maneira previsível e eficiente. Isto simplifica o fluxo de dados e facilita o desenvolvimento de aplicações complexas. A arquitetura do *Redux* é baseada no conceito de imutabilidade, o que significa que o estado da aplicação não pode ser modificado diretamente. Em vez disso, as alterações são feitas através de funções denominadas como *reducers*, que especificam como o estado deve ser atualizado em resposta a determinadas ações, garantindo que todas as mudanças no estado sejam rastreáveis e previsíveis.

Uma das principais vantagens do *Redux* é a sua capacidade de gerir estados globais, conforme ilustrado na Figura 2 (imagem obtida em [19]), o que torna mais fácil partilhar informações entre componentes diferentes, mesmo quando eles estão em diferentes partes da árvore de componentes. Isto evita a chamada *prop drilling*, onde os dados precisam ser passados através de múltiplos níveis de componentes.

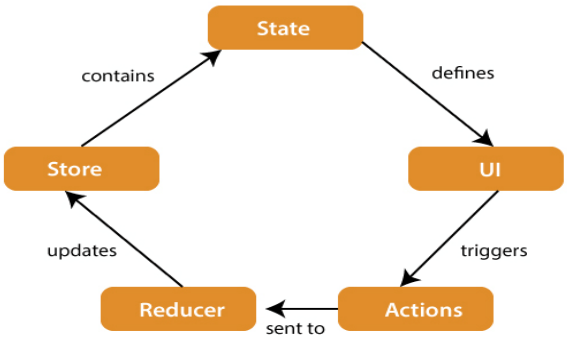

Figura 2 - Gestão dos estados com Redux

O Redux também é altamente extensível, permitindo a integração de *middleware* para lidar com tarefas assíncronas, como pedidos *HTTP*. Além disso, a arquitetura baseada em ações e *reducers* torna a gestão de erros mais fácil, pois cada ação é registada e pode ser facilmente identificada.

3. Desenvolvimento de uma loja *online*

Neste capítulo, são explorados os requisitos, a arquitetura e a interação do utilizador numa loja *online*, tanto do ponto de vista do cliente como do administrador. O objetivo principal é realizar um levantamento dos componentes necessários para a estrutura de uma loja *online*, de forma a desenvolver uma *framework* apropriada. O *website* criado para esta dissertação servirá como base para a construção da *framework*. O estudo centra-se numa loja fictícia de produtos informáticos, abrangendo o *frontoffice* para consulta e encomenda de produtos, um *backoffice* com ferramentas de gestão dos registos da base de dados. A presente dissertação foca as funcionalidades de carrinho de compras, encomendas e pagamentos, reconhecendo, contudo, a interdependência de outros componentes fundamentais, tais como gestão de produtos e utilizadores mesmo que não constituam o seu foco central. Para efetuar uma compra, o utilizador necessita de acrescentar produtos ao carrinho. Deste modo, a gestão de produtos, embora não seja o objeto central, requer robustez suficiente para que os produtos possam ser adequadamente exibidos e adicionados ao carrinho. Da mesma forma, a funcionalidade de encomendas depende da existência de um carrinho de compras bem definido, uma vez que é neste que os produtos se agregam antes do pagamento. O mesmo raciocínio é aplicável à gestão de utilizadores, uma vez que as encomendas estão associadas a contas de utilizador. Portanto, apesar de não se constituir como o foco central desta dissertação, esses componentes são desenvolvidos e mantidos de forma a serem simples e funcionais, contribuindo para viabilizar as funcionalidades cruciais de encomendas e pagamentos.

3.1 Requisitos funcionais

A loja *online* desenvolvida através da utilização de tecnologias *MERN*, disponibiliza as seguintes funcionalidades:

1. ***Frontoffice***
    - **a.** Registo e Autenticação de Utilizadores;

b. Perfil de Utilizador

i. Informação Pessoal;

ii. Histórico de Encomendas;

c. Listagem de Produtos

i. Filtrar por categoria ou nome;

ii. Filtrar produtos favoritos;

iii. Consultar detalhes do produto;

d. Encomendas e Pagamentos

i. Adicionar e remover produtos ao carrinho de compras;

ii. Selecionar método de pagamento;

iii. Efetuar encomenda.

2. ***Backoffice***

a. Gestão de Utilizadores

i. Adicionar, remover e editar utilizadores;

b. Gestão de Produtos

i. Adicionar, remover e editar produtos;

ii. Adicionar, remover e editar categorias;

c. Gestão de Encomendas

i. Adicionar, remover e editar encomendas;

ii. Visualizar produto mais vendido;

iii. Visualizar vendas por categoria.

Quando os utilizadores iniciam uma sessão na plataforma, têm a possibilidade de aceder a recursos adicionais, tais como explorar o catálogo completo de produtos, visualizar detalhes individuais de cada item e adicionar produtos ao carrinho de compras para preparar as suas encomendas antes de finalizar a compra. Além disso, os utilizadores autenticados podem efetuar encomendas, escolher o método de pagamento e acompanhar o histórico das suas compras anteriores. Têm também a liberdade de atualizar as suas informações pessoais para melhor gestão do perfil de utilizador.

Por outro lado, os utilizadores que não iniciaram sessão na plataforma têm a opção de iniciar sessão, caso já tenham uma conta, ou criar uma conta para se registarem na loja *online*. Mesmo sem iniciar sessão, têm acesso ao catálogo de produtos e podem visualizar informações básicas sobre cada item disponível, conforme ilustrado na Figura 3.

Os administradores, por sua vez, têm permissões adicionais e podem aceder aos mesmos casos de uso que os utilizadores autenticados. Além disso, possuem funcionalidades adicionais que lhes permitem gerir utilizadores, produtos e encomendas.

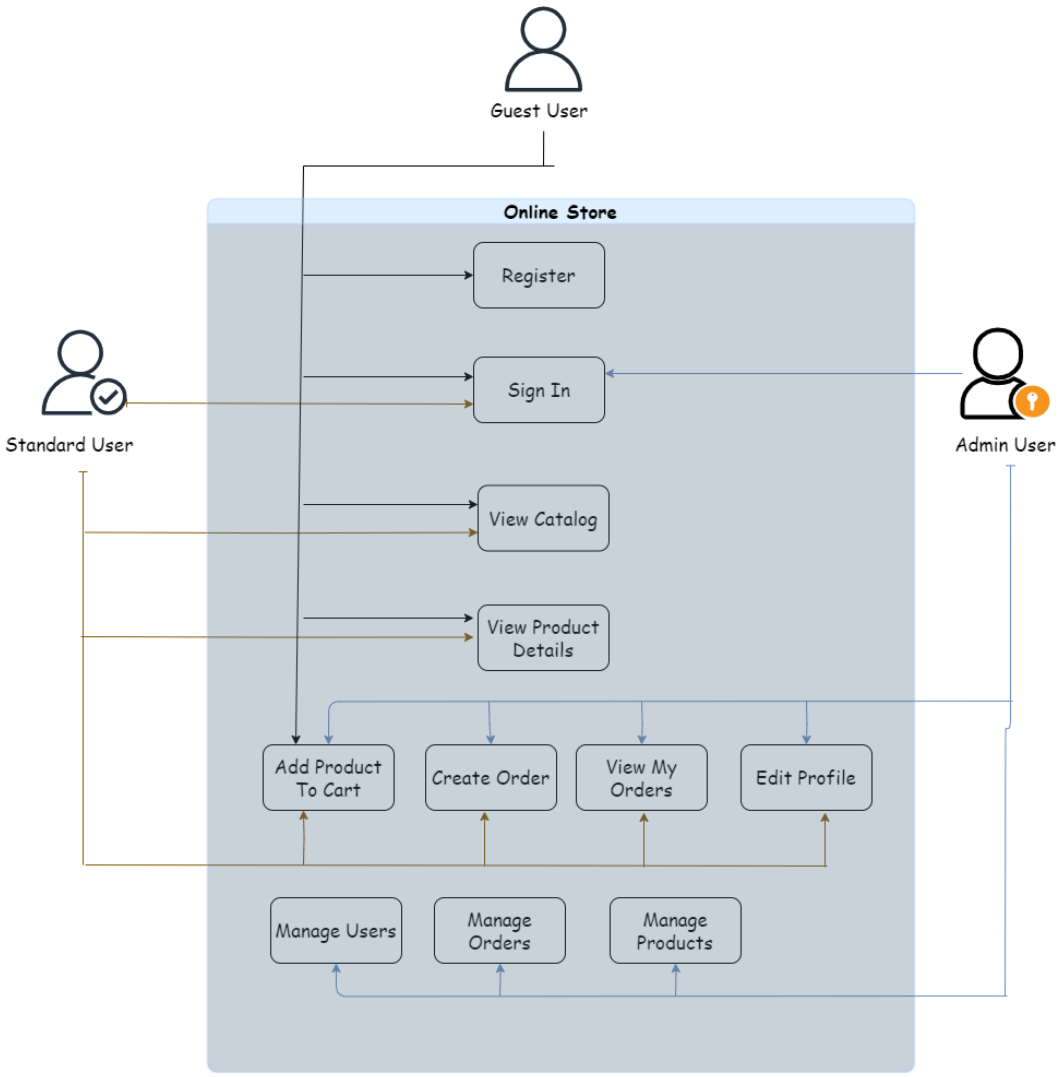

Figura 3 - Diagrama de casos de uso

3.2 Arquitetura do sistema

A arquitetura do sistema da loja *online* foi implementada, tendo como base a *stack MERN*, permitindo o desenvolvimento de uma aplicação escalável, com um *frontend* responsivo e uma base de dados não relacional, conforme representado na Figura 4.

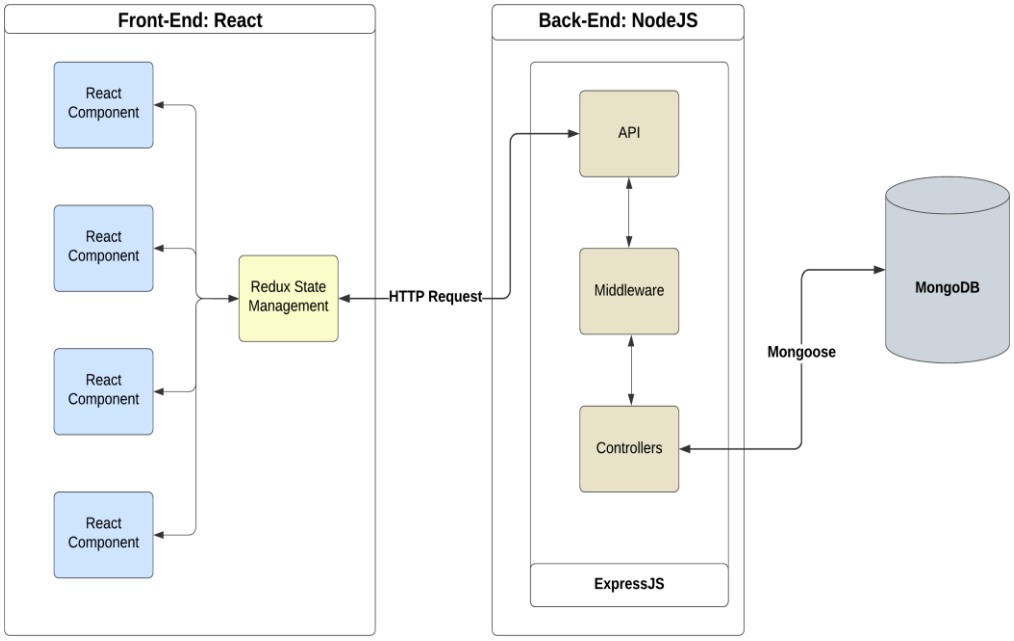

Figura 4 - Arquitetura do sistema

No lado do servidor, o *Node.js* foi utilizado para criar o servidor da aplicação, o que possibilitou a execução de *JavaScript* no *backend*, garantindo uma maior uniformidade na linguagem de programação em todo o projeto. Além disso, o *Node.js* oferece um desempenho elevado e capacidade de lidar com um grande número de pedidos simultâneos, tornando-o ideal para uma aplicação de *e-commerce*.

O *framework Express.js* desempenhou um papel fundamental na gestão das rotas e *APIs REST* da aplicação. Com o *Express.js*, foi possível lidar de forma eficiente com as diversas solicitações *HTTP*, simplificando o processo de comunicação entre o cliente e o servidor.

A utilização do *React* permitiu a criação de uma interface de utilizador reativa e interativa, proporcionando uma experiência de utilização fluida e dinâmica. A combinação do *React* com as bibliotecas *react-bootstrap* facilitou a estilização do site e agilizou o desenvolvimento, ao fornecer componentes pré-concebidos e estilos *CSS* personalizados.

Por outro lado, a *MongoDB*, uma base de dados não relacional, foi adotada devido à sua flexibilidade e capacidade de adaptação às necessidades da aplicação, permitindo um modelo de dados versátil e escalável.

A comunicação entre os diferentes componentes da aplicação é realizada através do protocolo *HTTP*, garantindo uma integração eficiente entre o cliente, o servidor e a base de dados. O uso da biblioteca *Mongoose* possibilitou uma gestão eficaz das interações entre o servidor *Node.js* e a base de dados *MongoDB*, facilitando a manipulação e o armazenamento dos dados.

Em suma, a arquitetura do sistema da loja *online* com tecnologias *MERN* proporcionou uma solução robusta, com um *frontend* intuitivo, uma base de dados flexível e uma gestão eficiente das operações de *backend*. A combinação destas tecnologias permitiu a criação de uma loja *online* completa.

Na secção seguinte é apresentado o modelo de dados utilizado pela loja *online* desenvolvida, descrevendo cada uma das coleções criada na base de dados disponibilizada pelos serviços da *MongoDB*. Posteriormente, é efetuada uma descrição da *API* do servidor da loja *online*, enumerando os diferentes pedidos *HTTP* suportados pela aplicação.

3.2.1 Modelo de dados

Nesta secção, será apresentado o modelo de dados utilizado na loja *online*, que está fundamentado na tecnologia não relacional *MongoDB*. O modelo de dados da loja *online* é composto por três coleções principais: *Orders, Products e Users*, conforme ilustrado na Figura 5.

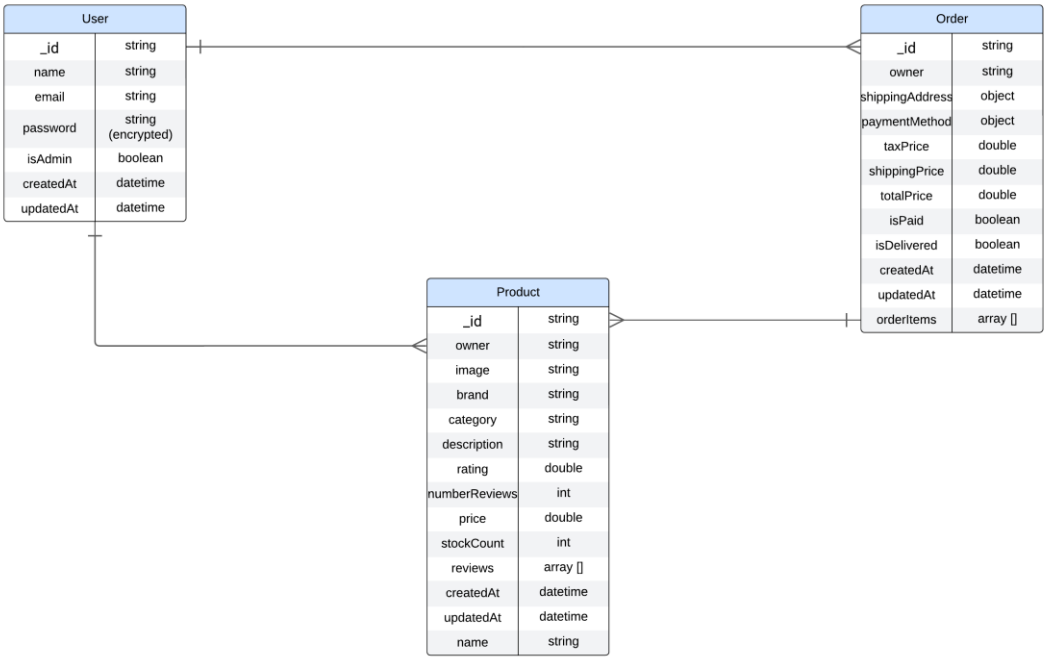

Figura 5 - Modelo de dados da loja *online*

A coleção relativa às encomendas, denominada *Orders,* armazena informações relacionadas às encomendas efetuadas pelos clientes, incluindo dados sobre os produtos, detalhes de pagamento, endereço de entrega e *status* do pedido.

A coleção relativa aos produtos, denominada *Products,* contém informações detalhadas sobre os produtos disponíveis na loja, como nome, imagem, marca, categoria, descrição, preço, stock e avaliações dos clientes.

A coleção desginada por *Users* contém os dados dos utilizadores, como nome, email, palavra-passe e informação sobre se são ou não administradores. Além disso, é importante destacar a relação entre as coleções. Cada encomenda da coleção "*Orders*" está associada a um utilizador específico através do campo *owner*, que armazena o *_id* do utilizador da coleção "*Users*". Da mesma forma, cada produto pertence a um utilizador e é identificado pelo campo *owner* na coleção "*Products*".

O modelo de dados proposto baseado em bases de dados não relacionais garante uma abordagem escalável e adaptável para o crescimento da loja *online* e acomoda as necessidades do sistema de forma ágil e dinâmica.

3.2.2 API

Nesta secção, são abordados e apresentados os diferentes pedidos *HTTP* suportados no *backend* da aplicação. O *backend*, desenvolvido em *Node.js e Express*, é responsável por gerir as solicitações dos clientes e fornecer as respostas adequadas para cada ação realizada na loja *online*.

A comunicação entre o *frontend* e o *backend* é feita através de *APIs RESTful*, que seguem os padrões de *design* de *API* e permitem uma interação clara e bem definida entre as diferentes partes da aplicação. Cada pedido é tratado pelos controladores no *backend*, que processam as informações e acedem à base de dados *MongoDB* para realizar as operações necessárias. É importante salientar que o *backend* implementa algumas medidas de segurança para garantir que apenas utilizadores autorizados possam aceder a determinados recursos e executar certas operações, embora não seja este o objetivo da presente dissertação.

3.2.2.1 Middleware

Nesta secção, são apresentados os *middlewares* implementados na aplicação, com as respetivas funcionalidades:

1. **Proteção de Rotas**: responsável por proteger rotas específicas da aplicação, garantindo que apenas utilizadores autenticados e autorizados tenham acesso a determinadas funcionalidades. Utilizando *JSON Web Tokens* fornecidos no cabeçalho dos pedidos *HTTP*, o *middleware* verifica a autenticidade do utilizador e permite o acesso caso o *token* seja válido. Caso o *token* seja inválido ou inexistente, o *middleware* responde com um código de estado *HTTP 401* (*Unauthorized*) e uma mensagem de erro.

2. **Autorização de Administrador**: utilizado para controlar o acesso a funcionalidades administrativas da aplicação, garantindo que apenas utilizadores com permissões de administrador possam aceder a determinadas rotas. Caso não seja um administrador, o *middleware* responde com um código de estado *HTTP 401 (Unauthorized)* e uma mensagem de erro indicando a falta de autorização.

3. **Tratamento de Erros**: Responsável por lidar com erros durante o processamento de pedidos *HTTP*. Quando o cliente tenta aceder a uma rota que não está definida na aplicação, o *middleware* responde com um código de estado *HTTP 404 (Not Found)* e uma mensagem de erro apropriada.

3.2.2.2 Pedidos HTTP suportados

Nesta secção, são apresentadas as diferentes rotas implementadas na aplicação, que permitem a interação entre o cliente e o servidor, sendo cada uma responsável por executar uma funcionalidade específica. A estruturação adequada das rotas, combinada com a utilização de *middlewares* para proteção e autorização, contribui para uma aplicação coesa, segura e com uma experiência de utilização otimizada. As rotas inerentes aos utilizadores, produtos e encomendas estão representadas na Tabela 2, Tabela 3 e Tabela 4 respetivamente.

Tabela 2 - Rotas de utilizadores e respetivas funcionalidades

Funcionalidade	Rota	HTTP	Middleware
Registar um novo utilizador	/api/users/	POST	Nenhum
Obter todos os utilizadores	/api/users/	GET	Admin Protected
Obter perfil do utilizador	/api/users/profile	GET	Protected
Atualizar perfil do utilizador	/api/users/profile	PUT	Protected
Apagar um utilizador	/api/users/:id	DELETE	Admin Protected
Obter utilizador por ID	/api/users/:id	GET	Admin Protected
Atualizar informações do utilizador	/api/users/:id	PUT	Protected
Login do utilizador	/api/users/login	POST	Nenhum
Logout do utilizador	/api/users/logout	POST	Nenhum

Tabela 3 - Rotas de produtos e respetivas funcionalidades

Funcionalidade	Rota	HTTP	*Middleware*
Obter todos os produtos	/api/products/	GET	Nenhum
Obter produto por ID	/api/ products /:id	GET	*Admin Protected*
Atualizar detalhes do produto	/api/ products /:id	PUT	*Admin Protected*
Apagar um produto	/api/ products /:id	DELETE	*Admin Protected*

Tabela 4 - Rotas de encomendas e respetivas funcionalidades

Funcionalidade	Rota	*HTTP*	*Middleware*
Criar uma encomenda	/api/orders/	*POST*	*Protected*
Obter encomenda por ID	/api/orders/:id	*GET*	*Protected*
Atualizar estado de pagamento da encomenda	/api/orders/:id/pay	*PUT*	*Protected*
Atualizar estado de entrega da encomenda	/api/orders/:id/deliver	*PUT*	*Admin Protected*
Obter as minhas encomendas	/api/orders/myorders	*GET*	*Protected*

3.3 Implementação

Na presente secção, são explorados alguns detalhes da implementação da nossa loja *online*, de modo a demonstrar as funcionalidades no panorama geral da aplicação, quer do ponto de vista de um utilizador, quer de um administrador, descrevendo a estrutura do código da loja e o respetivo mapa de páginas.

As funcionalidades de encomenda e pagamento, são abordadas com maior detalhe no capítulo 4, uma vez que constituem o foco da *framework* desenvolvida.

3.3.1 Estrutura do código

Nesta seção, são descritas as diretorias e ficheiros que compõem a arquitetura da aplicação, demonstrando como as diferentes partes do sistema estão organizadas e estruturadas. Na Figura 6 é apresentada a organização e hierarquia dos diferentes ficheiros que constituem a loja *online* desenvolvida.

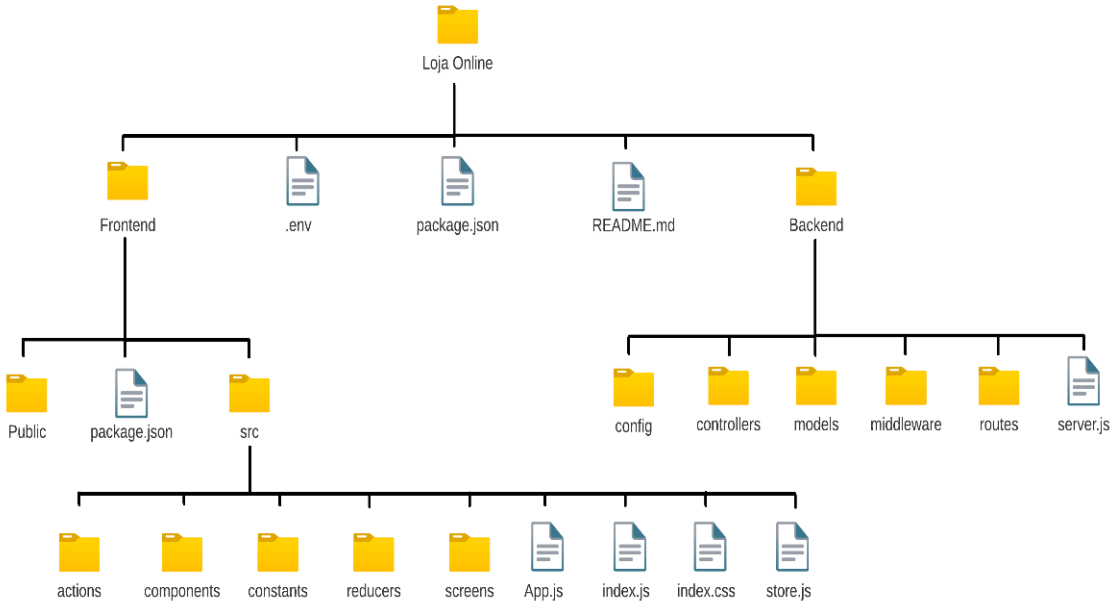

Figura 6 - Estrutura do código da loja *online*

O ficheiro *package.json*, lista as dependências do projeto instaladas com recurso ao comando *npm* e define *scripts* de inicialização. O ficheiro *.env* é reponsável por guardar algumas variáveis sensíveis do ambiente de execução, como chaves de *API*.

O *backend* é responsável pela lógica do servidor, definição de rotas, aplicação de *middlewares* e outras configurações. As seguintes diretorias constituem o núcleo do *backend* da loja desenvolvida, representadas na Figura 7:

- ***models*** – definição dos modelos de dados dos diferentes objetos utilizados nas coleções definidas na base de dados (*Orders, Users, Products*).

- ***config*** - ficheiros de configuração relativos à autenticação e interligação com a base de dados.
- ***routes*** - definição das diferentes rotas para interação com o servidor da loja *online*, sendo que cada rota está implicitamente ligada a um controlador.
- ***controllers*** – contém os ficheiros responsáveis por lidar com os pedidos efetuados ao servidor da aplicação e aplicar a lógica inerente a cada operação.

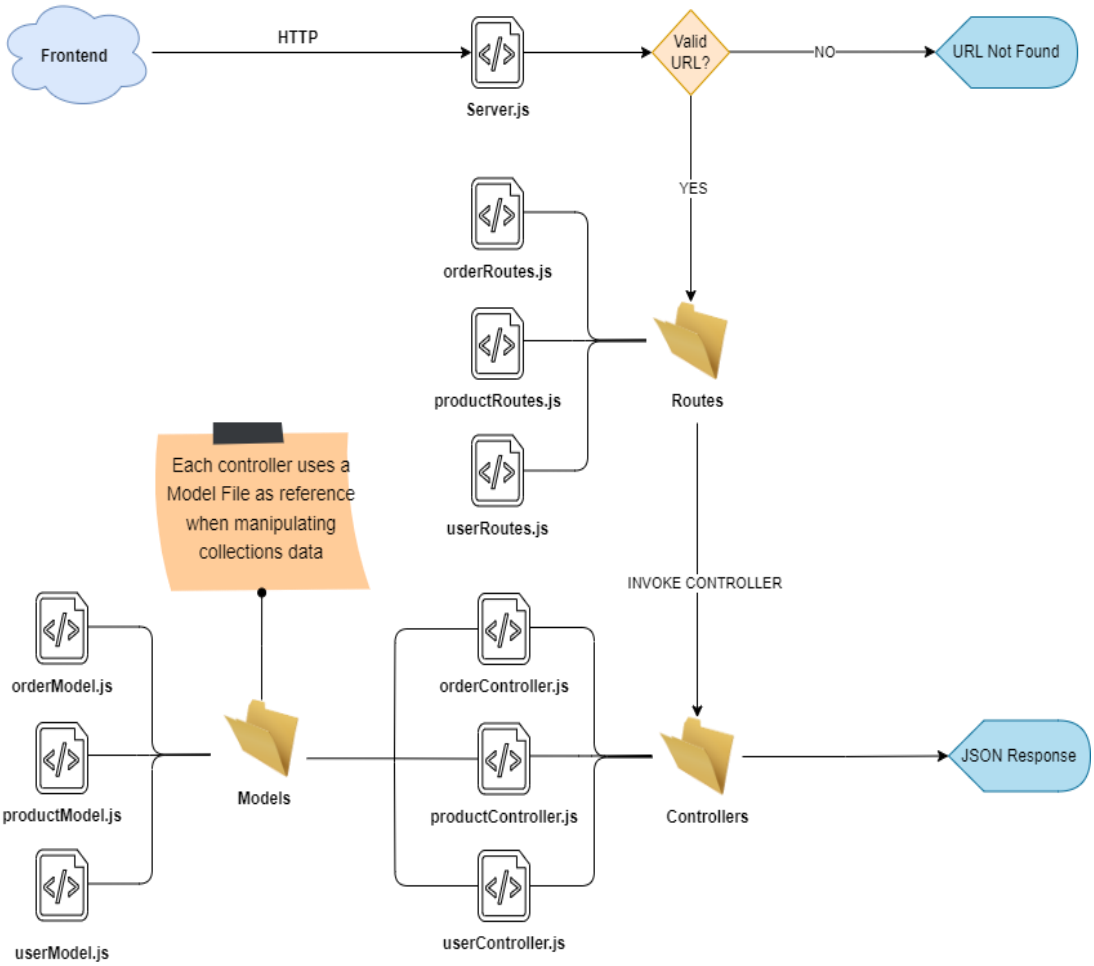

Figura 7 - *Backend* da loja *online*

Relativamente à diretoria *frontend*, é constituida por uma pasta – *public* - que contém o ficheiro *HTML* da aplicação *React* e armazena recursos que podem ser acedidos diretamente pelo *browser*, como imagens e ícones. Por sua vez, a diretoria *src* é constituída pelos ficheiros *store.js,* que configura e gere o estado global da aplicação, *App.js* que atua como componente principal e é renderizado com o *React DOM* através do ficheiro *index.js*. As seguintes diretorias constituem o núcleo do código desenvolvido para o *frontend* da aplicação:

- **actions** - contém ficheiros que definem as ações *Redux*, responsáveis pela atualização do estado global da aplicação.
- **components -** contém todos os componentes *React* utilizados
- **constants -** define constantes utilizadas em toda a aplicação, incluindo os tipos de ações *Redux*.
- **reducers** - contém os redutores *Redux*, que processam ações e atualizam o estado global.
- **screens** – define a *UI* das diferentes páginas da aplicação, cada página corresponde a um ficheiro distinto.

Os principais ficheiros e pastas da loja *online* desenvolvida estão apresentados e descritos na Tabela 5, representando assim o conteúdo da diretoria principal – *src* da loja *online* desenvolvida. Cada entrada na tabela corresponde a um ficheiro utilizado no *frontend* da loja, referindo a sua utilização para operações de *frontoffice e backoffice.*

Nas seguintes secções são apresentadas as diferentes páginas e as funcionalidades da loja *online* desenvolvida para os diferentes tipos de utilizador. O *frontoffice* representa as funcionalidades que um utilizador comum pode efetuar, por outro lado, o *backoffice* é constituído pelas funcionalidades administrativas de gestão dos registos presentes na base de dados da loja. De modo a evitar redundâncias, os componentes-chave serão descritos no capítulo 4 - Plataforma para criação de lojas.

Tabela 5 - *Frontend* da loja *online*

Diretoria	Nome do ficheiro	Descrição
Actions	userActions.js	Definição das ações *Redux* relacionadas com utilizadores, encomendas, carrinho de compras e produtos. **Utilização**: *Frontoffice* e *Backoffice*
	orderActions.js	
	cartActions.js	
	productActions.js	
Components	CheckoutSteps.js	Componentes *React* utilizados em diferentes páginas e contextos da aplicação. **Utilização**: *Frontoffice* e *Backoffice*
	CustomFormContainer.js	
	CustomMessage.js	
	CustomSpinner.js	
	Footer.js	
	Header.js	
	Product.js	
	Rating.js	
	SearchBar.js	
Constants	userConstants.js	Definição de constantes utilizadas em toda a aplicação, incluindo os tipos de ações *Redux*. **Utilização**: *Frontoffice* e *Backoffice*
	orderConstants.js	
	cartConstants.js	
	productConstants.js	
Reducers	userReducers.js	Definição de redutores que processam ações e atualizam o estado global. **Utilização**: *Frontoffice* e *Backoffice*
	orderReducers.js	
	cartReducers.js	
	productReducers.js	
Screens	CartScreen.js	Página do carrinho de compras. **Utilização**: *Frontoffice*
	CreateOrderScreen.js	Página de criação de uma encomenda. **Utilização**: *Frontoffice*

	EditProductScreen.js	Página administrativa - produto. **Utilização**: *Backoffice*
	EditUserScreen.js	Página administrativa - utilizador. **Utilização**: *Backoffice*
	HomeScreen.js	Página principal da aplicação. **Utilização**: *Frontoffice*
	LoginScreen.js	Página de autenticação. **Utilização**: *Frontoffice*
	OrderListScreen.js	Página administrativa - encomendas. **Utilização**: *Backoffice*
	OrderScreen.js	Página de detalhes de uma encomenda. **Utilização**: *Frontoffice* e *Backoffice*
	PaymentScreen.js	Página de detalhes de pagamento. **Utilização**: *Frontoffice*
	ProductListScreen.js	Página administrativa – produtos **Utilização**: *Backoffice*
	ProductScreen.js	Página de detalhes de um produto. **Utilização**: *Frontoffice* e *Backoffice*
	ProfileScreen.js	Perfil do utilizador. **Utilização**: *Frontoffice*
	RegisterScreen.js	Página de registo. **Utilização**: *Frontoffice*
	ShippingScreen.js	Página de detalhes de entrega **Utilização**: *Frontoffice*
	UserListScreen.js	Página administrativa – utilizadores **Utilização**: *Backoffice*
-	App.js	Ponto de entrada da aplicação *React*
-	index.js	Inicializa e interliga a aplicação *React com a store.js.*
-	store.js	Configuração e utilização dos redutores.

3.3.2 Mapa de páginas

Nesta secção são apresentadas as diferentes páginas que constituem a loja *online* desenvolvida. A Figura 8 ilustra o mapa de páginas da loja *online*, representando as páginas inerentes ao *frontoffice e backoffice.*

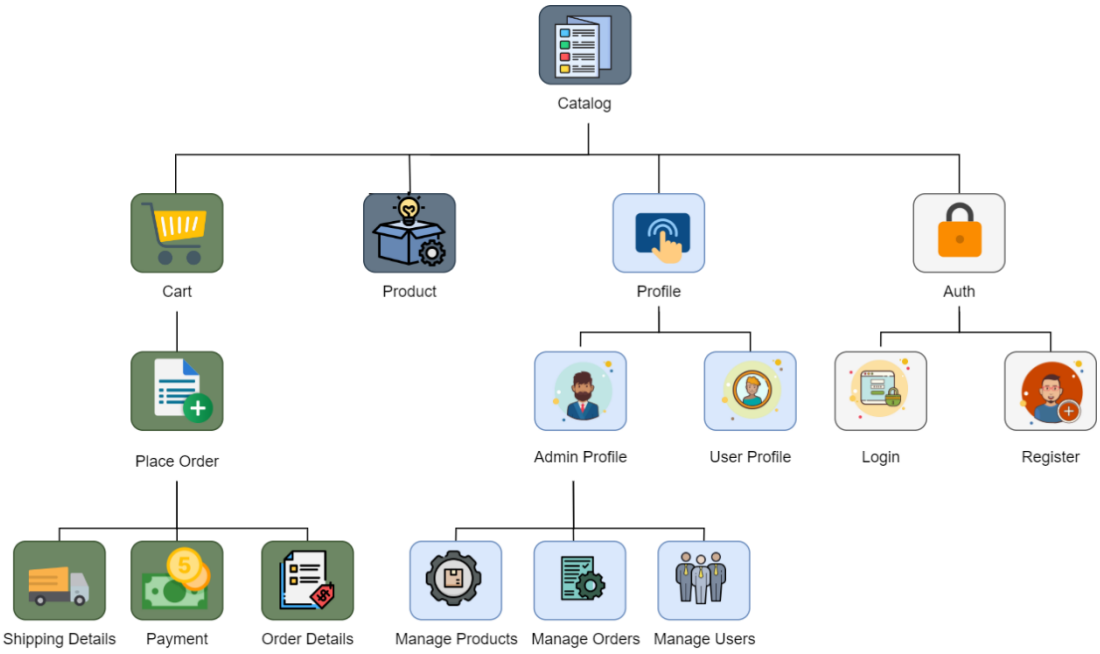

Figura 8 - Mapa de páginas da loja *online*

Relativamente ao *frontoffice,* é constituído pelas páginas acessíveis a um utilizador comum como o catálogo e detalhes dos produtos, a página do carrinho de compras e criação encomendas, a página de registo e autenticação e o perfil de utilizador. O *backoffice* é constituído por todas as páginas com funções administrativas responsáveis por manipular e gerir os dados armazenados nas coleções relativas a encomendas, produtos e utilizadores da base de dados.

Na próxima secção é abordado o funcionamento destas páginas de modo a apresentar a loja *online* desenvolvida de uma forma geral. Como referido anteriormente os componentes-chave serão abordados com maior detalhe no capítulo 4 - Plataforma para criação de lojas.

3.3.3 Frontoffice

A página inicial desempenha um papel crucial na plataforma, servindo como o ponto de partida para os utilizadores explorarem os produtos disponíveis tendo ou não sessão iniciada, conforme representado na Figura 9. Um aspeto fundamental desta abordagem é a implementação de componentes reutilizáveis. Esta estratégia permite dividir a página em partes menores e independentes, facilitando a manutenção e permitindo que elementos similares sejam reaproveitados em diferentes partes da aplicação.

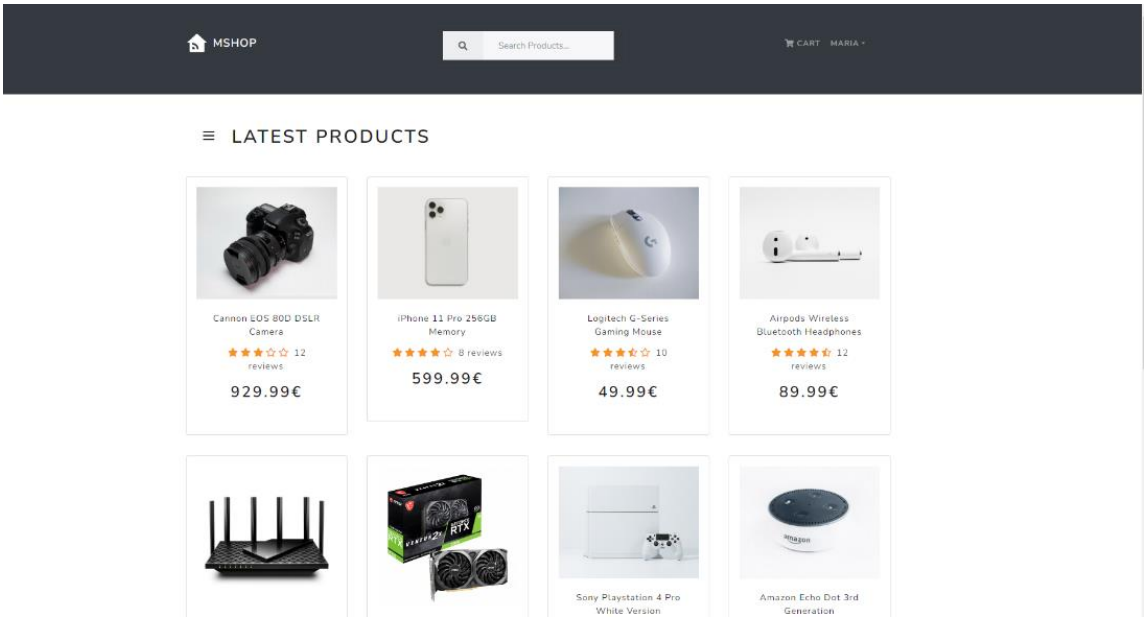

Figura 9 – Página inicial da loja *online*

Por exemplo, a exibição dos produtos é gerida por um componente que pode ser usado em várias áreas da plataforma. A página inicial é constituída por diferentes componentes, cada um responsável por diferentes tarefas, tais como: a listagem dos produtos, a barra da aplicação que é comum a todas as páginas, etc.

O utilizador pode também aplicar filtros, tais como: marca, categoria, preço e tipo de ordenação, sendo que os produtos presentes na base de dados serão filtrados e apresentados ao utilizador conforme ilustrado na Figura 10.

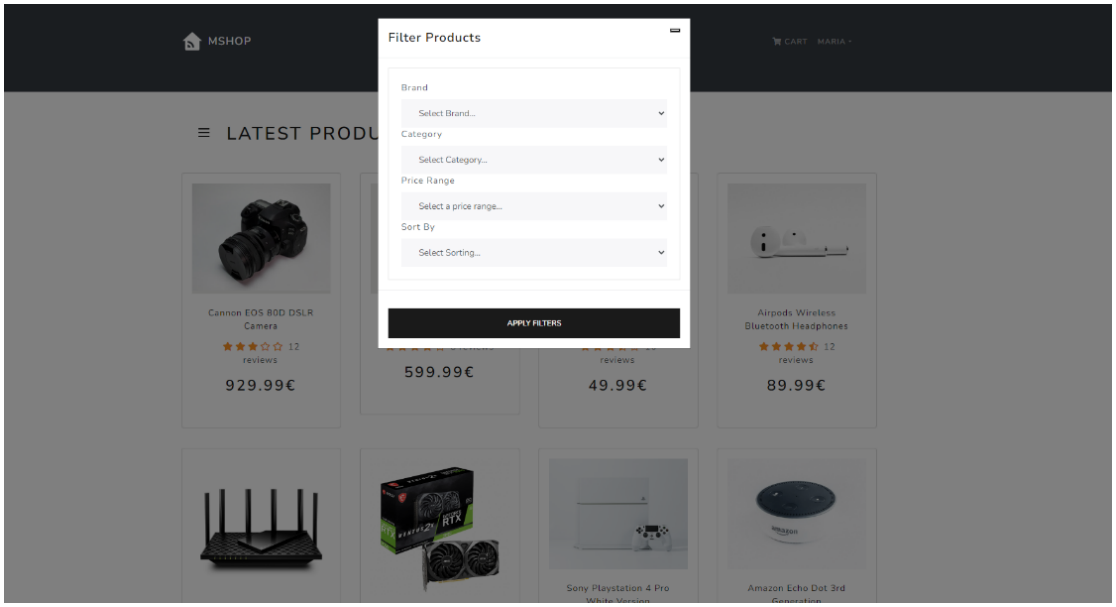

Figura 10 - Filtragem de produtos

A barra principal da aplicação incorpora elementos como o componente de pesquisa de produtos, o acesso direto ao carrinho de compras e perfil do utilizador. Estes recursos essenciais, são acessíveis a partir de qualquer ponto da aplicação, permitindo aos utilizadores explorar produtos de forma eficiente, gerir facilmente as suas compras em andamento e aceder rapidamente ao seu perfil pessoal. O componente do carrinho de compras cujo principal objetivo é proporcionar uma visão clara dos produtos que o utilizador adicionou ao seu carrinho e permitir-lhe avançar com o processo de *checkout,* que é abordado com maior detalhe nas secções seguintes.

De modo a efetuar uma encomenda, o utilizador deve adicionar produtos ao carrinho e iniciar o processo sendo redirecionado para a página de *login*, caso o utilizador ainda não tenha sessão iniciada. Além disso, como parte do processo de *checkout*, após efetuar o *login* com sucesso, o utilizador será redirecionado para a página de informações relativas à nova encomenda onde é necessário preencher alguns detalhes, como o endereço de entrega e selecionar o método de pagamento, sendo que será utilizado o *PayPal* ao longo da presente dissertação.

Após o utilizador criar uma encomenda, será redirecionado para a página com os detalhes da encomenda, conforme ilustrado na Figura 11. Inicialmente, ao carregar a página, os detalhes completos do pedido são extraídos do estado global da aplicação, abrangendo informações essenciais como os produtos selecionados, endereço de entrega e método de pagamento. Como referido anteriormente, na presente dissertação optamos por utilizar a *API* disponibilizada pelo *PayPal*, sendo que possibilita também os pagamentos através de cartão bancário e permite a integração destes métodos de pagamento de forma simples, tendo apenas de efetuar a instalação das bibliotecas inerentes com recurso ao instalador de pacotes *npm*.

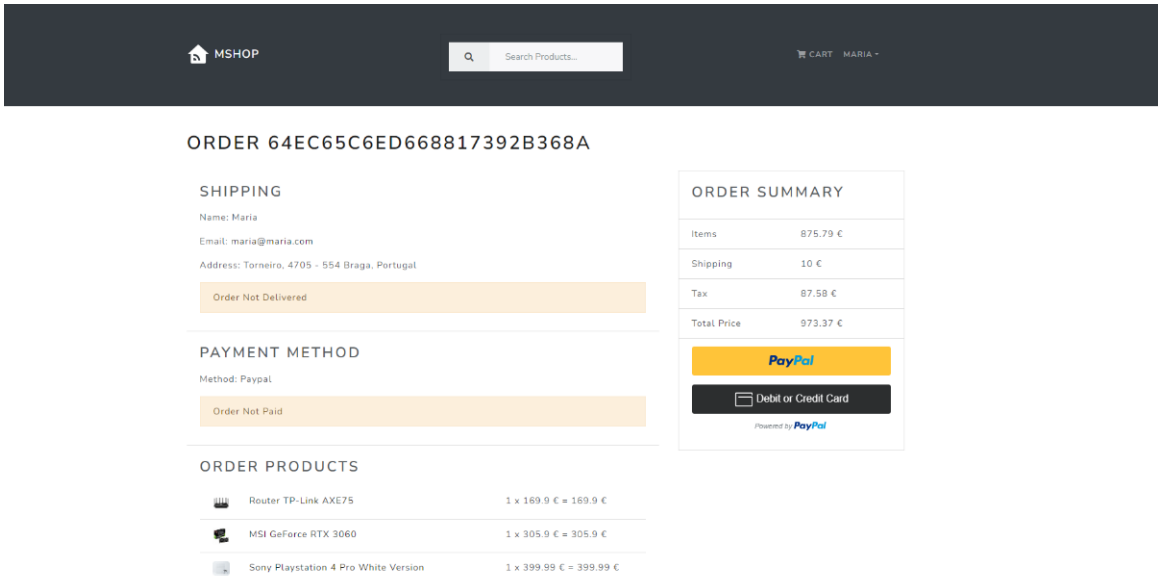

Figura 11 - Detalhes da encomenda

Quando o pagamento é efetuado com sucesso, a ação correspondente no *Redux* é acionada, resultando na atualização imediata do estado do pedido para refletir essa transação. Além disso, uma opção é proporcionada aos administradores do sistema para marcar uma encomenda como entregue.

O utilizador pode consultar o histórico de encomendas realizadas através da página de perfil do utilizador, podendo também consultar os detalhes e efetuar o pagamento caso ainda não o

tenha feito. Esta página apresenta ainda as informações pessoais, que podem ser atualizadas a qualquer momento, conforme representado na Figura 12.

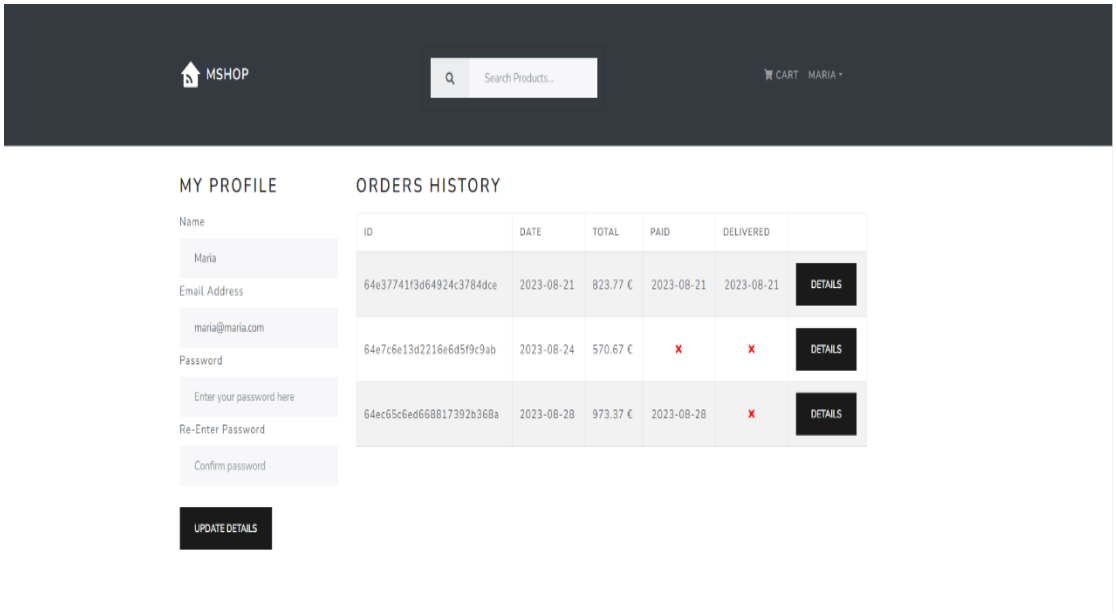

Figura 12 - Perfil do utilizador

3.3.4 Backoffice

Nesta secção, são analisadas as funcionalidades disponibilizadas pela loja *online* do ponto de vista do administrador, analisando como o *backoffice* é estruturado e como as tarefas administrativas são geridas. Como referido anteriormente, o administrador tem acesso a todas as funcionalidades disponibilizadas a um utilizador comum.

Do ponto de vista do administrador, o cabeçalho da aplicação desempenha um papel crucial ao fornecer acesso rápido e eficiente ao menu de gestão. Neste contexto, o cabeçalho oferece opções específicas de administração, como o acesso a listagens de utilizadores, produtos e encomendas.

Além disso, permite a navegação para áreas de criação, edição e eliminação de conteúdo, facilitando a manutenção e manipulação dos dados presentes na base de dados, conforme representado na Figura 13.

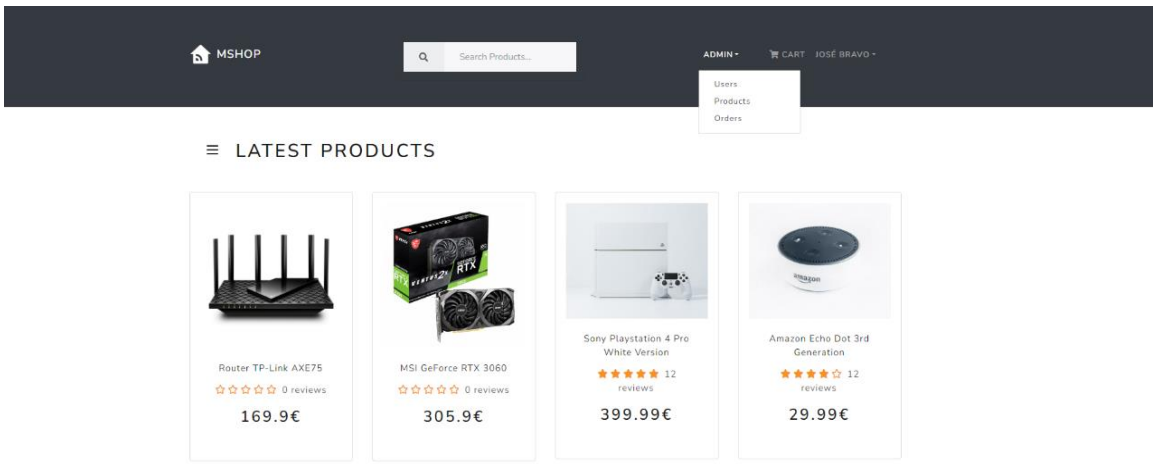

Figura 13 - Menu de administrador

A página administrativa relativa aos utilizadores, representada na Figura 14, desempenha um papel fundamental na gestão de utilizadores da aplicação. Através deste ecrã, os administradores têm a capacidade de visualizar e interagir com a lista completa de utilizadores registados.

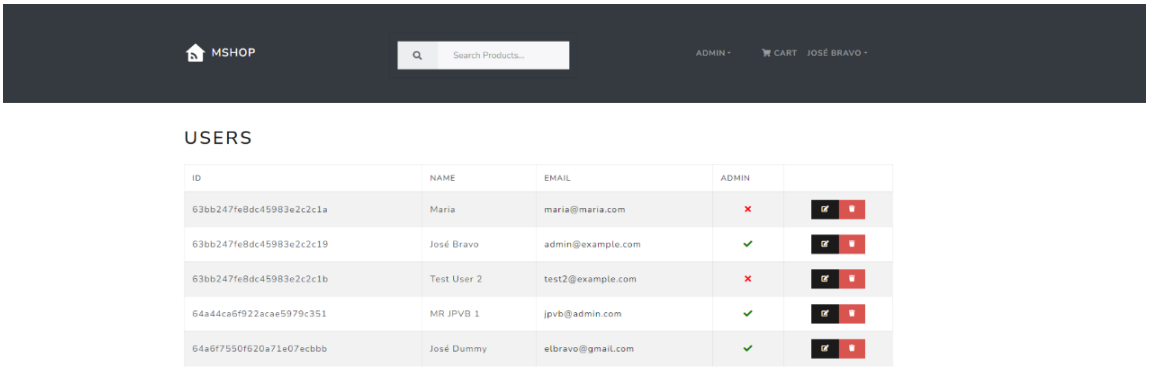

Figura 14 - Gestão de utilizadores

A informação apresentada é organizada numa tabela que inclui os campos de identificação, nome e endereço de *email* dos utilizadores. Além disso, é indicado se um utilizador possui privilégios de administrador. Este ecrã também disponibiliza opções para editar os detalhes de um utilizador ou remover permanentemente um utilizador específico. Em suma, esta página

permite que os administradores mantenham um controlo completo sobre as contas de utilizador na aplicação.

Relativamente à página de gestão dos produtos, os administradores têm a capacidade de visualizar e gerir a lista completa de produtos disponíveis na loja. Os produtos são apresentados numa tabela, onde os campos de identificação, nome, preço, categoria e marca são claramente exibidos. Para simplificar ainda mais a gestão, os administradores têm a opção de criar produtos através de um botão de criação dedicado. Além disso, cada registo contém atalhos que permitem aos administradores ajustar detalhes individuais dos produtos ou remover produtos existentes.

De forma semelhante, a página das encomendas desempenha um papel essencial na gestão das encomendas da loja *online*. Este ecrã é acessível apenas aos administradores e permite visualizar uma lista abrangente de todas as encomendas efetuadas pelos clientes. As encomendas são apresentadas numa tabela que inclui campos como ID, proprietário, data, total, estado de pagamento e estado de entrega. Além disso, os administradores têm a possibilidade de visualizar os detalhes completos de cada encomenda, clicando no botão associado a cada registo. Esta interface simplificada proporciona aos administradores uma visão clara e organizada das encomendas efetuadas, permitindo-lhes gerir eficientemente o ciclo de vida de cada encomenda, conforme ilustrado na Figura 15.

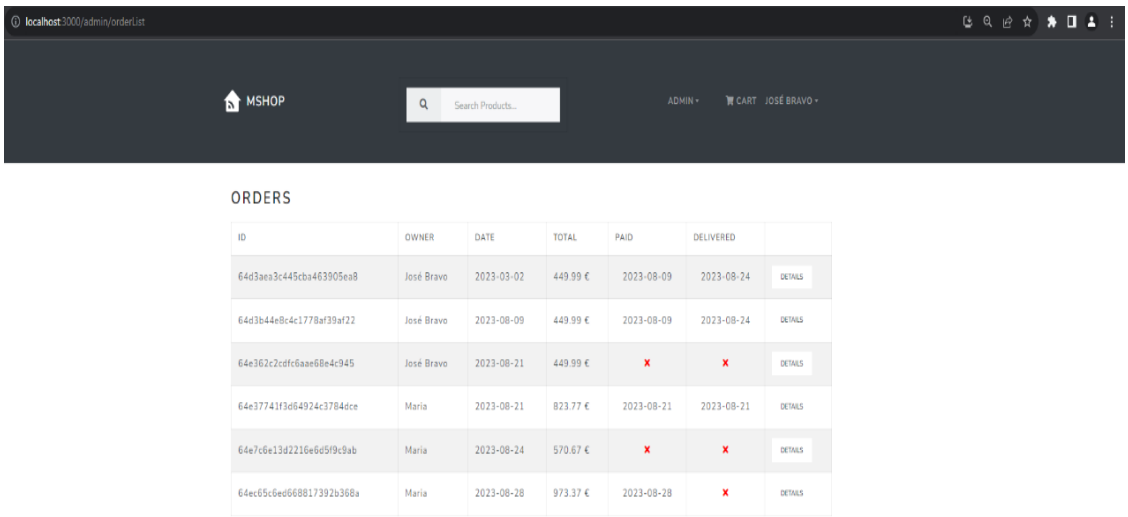

Figura 15 - Gestão de encomendas

4. Plataforma para criação de lojas

Neste capítulo é descrita a plataforma desenvolvida para a criação de lojas *online*, desde a arquitetura da ferramenta até à *API* do *backend* e a estrutura da *framework* que possibilita uma gestão eficiente das funcionalidades de encomendas, carrinho de compras e pagamentos. Além disso, também são abordadas as parametrizações necessárias para personalizar e integrar esses componentes-chave. Este capítulo proporciona uma visão detalhada do funcionamento da solução, fornecendo uma compreensão completa de como a *framework* foi concebida para disponibilizar componentes customizáveis devidamente estruturados, permitindo uma gestão eficaz das funcionalidades de encomendas, carrinho de compras e pagamentos, adaptadas às necessidades exclusivas de cada utilizador. A ferramenta desenvolvida utiliza as mesmas tecnologias desenvolvidas na loja *online* descrita anteriormente, possibilitando assim a integração dos diferentes componentes customizáveis desenvolvidos.

4.1 Funcionalidades Suportadas

A plataforma desenvolvida consiste numa aplicação *MERN*, que disponibiliza ao utilizador um formulário intuitivo onde é possível configurar alguns detalhes da loja *online* a gerar, podendo personalizar as seguintes propriedades:

- Definir nome da loja e detalhes do utilizador;
- Personalizar a loja selecionando um dos temas *bootstrap* disponibilizados;
- Configurar a base de dados fornecida pelos serviços da *MongoDB:*
    - Definir *URI* de conexão à base de dados;
    - Configurar novos campos para as coleções da base de dados (*Orders, Users, Products)*.
- Personalizar as diferentes páginas:
    - Adicionar barra de navegação;

- Adicionar rodapé;
- Alterar cor de botões e alertas;
- Alterar o tipo de letra.

- Definir a página inicial da loja.

O utilizador pode também optar por incluir diferentes funcionalidades na loja com base no objetivo de negócio, sendo que estas funcionalidades serão disponibilizadas através da integração dos componentes-chave inerentes a cada uma. O utilizador pode pré-visualizar e opcionalmente adicionar as seguintes páginas e funcionalidades:

- Registo e autenticação de utilizadores;
- Gestão dos registos armazenados na base de dados:
  - Encomendas, Produtos e Utilizadores.
- Estatísticas relativas aos dados da loja;
- Catálogo e filtragem de produtos;
- Carrinho de compras, encomendas e pagamentos.

Após definir os detalhes e selecionar as funcionalidades desejadas, o utilizador tem a possibilidade de fazer o *download* do código fonte da aplicação gerada num formato *.zip,* conforme ilustrado na Figura 16.

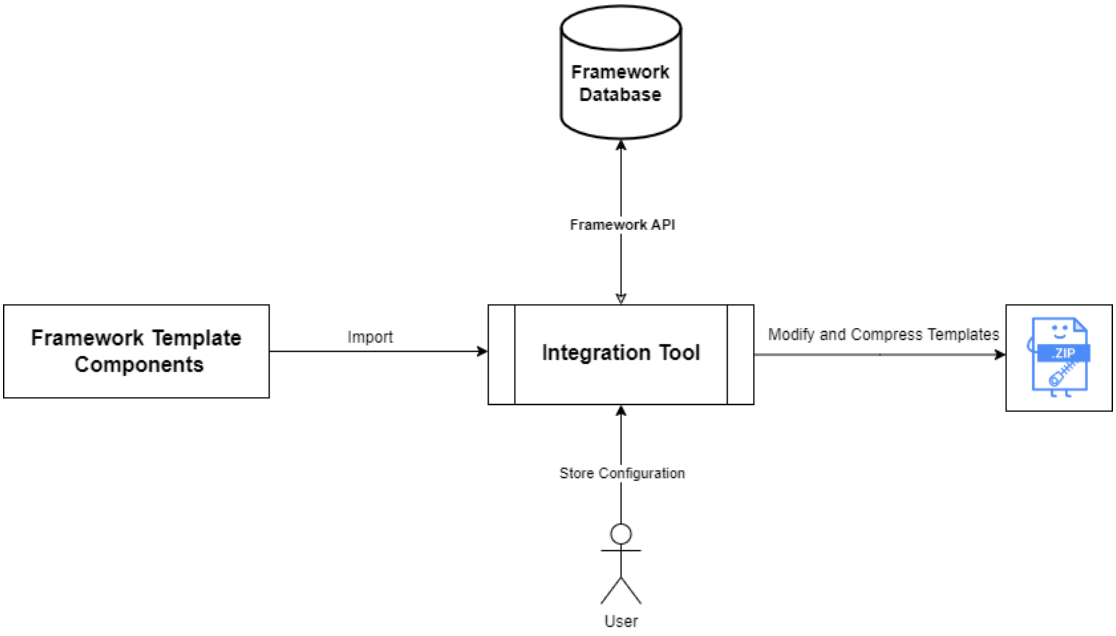

Figura 16 - Arquitetura da plataforma de geração de lojas

A presente dissertação tem como foco os componentes associados às funcionalidades do carrinho de compras, encomendas e pagamentos, designadas por funcionalidades-chave.

A plataforma pode ser conceptualmente divida em duas partes: a *framework* e a ferramenta de geração da loja. A *framework* é responsável por disponibilizar os componentes customizáveis e uma *API* para armazenar e manipular os registos referentes a cada configuração de uma loja. Por outro lado, a ferramenta é responsável por importar e manipular os *templates* disponibilizados pela *framework,* de modo a gerar uma loja *online*, com as propriedades definidas pelo utilizador, em formato *.zip*.

4.2 Estrutura do código

A estrutura do código da plataforma desenvolvida, representada na Figura 17, segue a mesma lógica utilizada no desenvolvimento da loja *online* do capítulo anterior, sendo que utiliza uma arquitetura modular e escalável, facilitando tanto a manutenção como a implementação de novas funcionalidades. O código está dividido em duas partes principais: o *backend*, desenvolvido em *Node.js*, e o *frontend*, construído com recurso à biblioteca *React*. A análise subsequente permitirá compreender a inter-relação entre estes componentes.

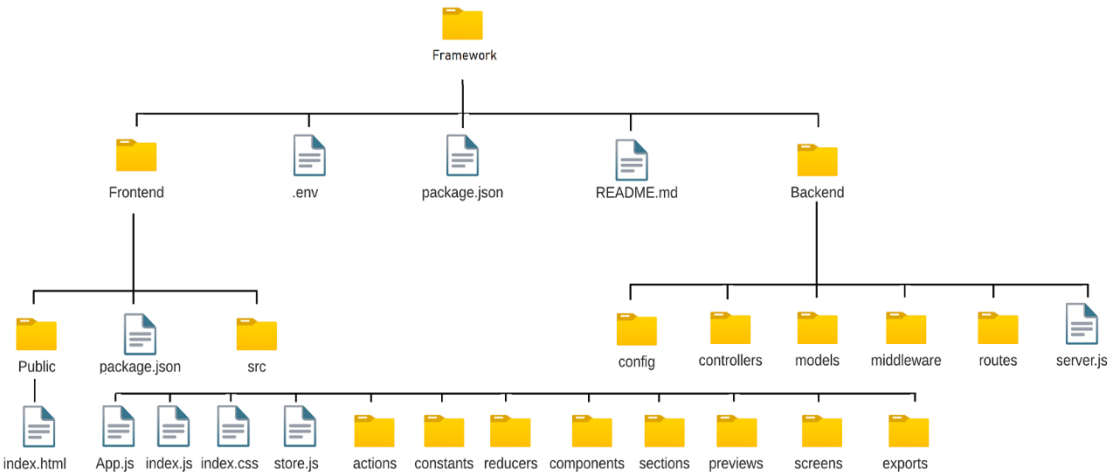

Figura 17 - Estrutura do código da plataforma de geração de lojas

O ficheiro *.env* da diretoria principal, é responsável por armazenar algumas variáveis sensíveis da ferramenta, como o *URI* utilizado para aceder à base de dados e o *PAYPAL_CLIENT_ID* utilizado para integrar os serviços disponibilizados pela *API* do *PayPal*.

Relativamente ao *frontend*, é responsável pela interação direta com o utilizador disponibilizando diferentes páginas, como a página inicial onde é possível configurar uma nova loja, e a página de edição de um registo existente, contidas na diretoria *screens.* As páginas de criação e edição de configurações de uma loja, consistem num formulário onde o utilizador pode optar por incluir as diferentes funcionalidades como também definir os detalhes e tema da loja *online* a gerar. Cada funcionalidade pode ser pré-visualizada, recorrendo para tal, aos ficheiros contidos na diretoria *previews* que utilizam dados estáticos de modo a permitir ao utilizador ter uma noção mais clara do aspeto e propriedades inerentes a cada uma. A diretoria *exports* contém todos os *templates* que irão ser utilizados pela loja *online* gerada, tanto a nível de *frontend* como de *backend*. Estes ficheiros são alterados consoante as escolhas do utilizador, através da utilização de *tags* que são substituídas durante o processo de *download,* conforme representado na Figura 18.

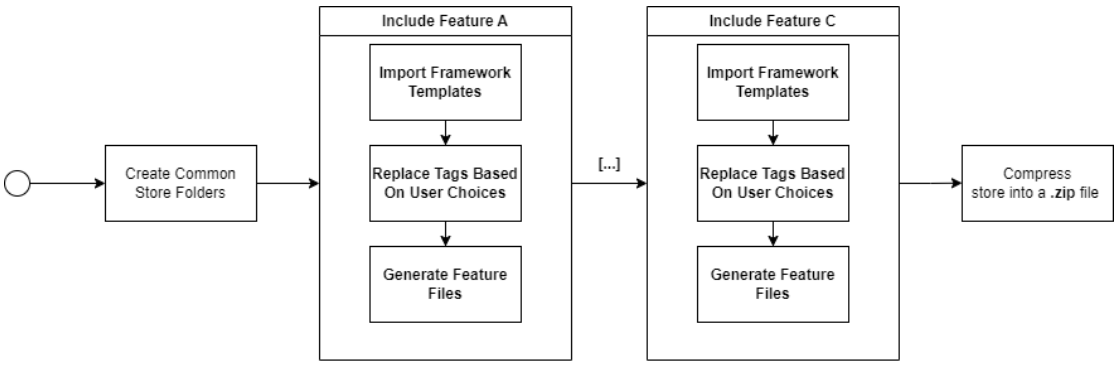

Figura 18 - Processo de *download* do ficheiro *.zip*

No *backend* temos o ficheiro relativo ao modelo da coleção da base de dados da *framework*, as rotas suportadas pelo servidor e os controladores. As rotas são definidas no ficheiro *server.js* e *routes.js*, permitindo a realização das operações *CRUD (Create, Read, Update, Delete)* sobre os registos presentes na base de dados, através da utilização dos controladores respetivos.

Os componentes customizáveis que constituem a *framework* encontram-se na diretoria *exports,* sendo o seu processamento e integração efetuado através do ficheiro *Download.js.* Na secção seguinte é explicado o funcionamento da *framework,* dando ênfase aos componentes customizáveis, associados às funcionalidades-chave, o modelo de dados e a *API* disponibilizada para posterior interação e integração dos componentes através da ferramenta.

4.3 Framework

A estrutura lógica dos componentes específicos da loja *online* é fundamental para o bom funcionamento das funcionalidades de encomendas, carrinho de compras e pagamentos. Nesta secção são descritos os componentes fundamentais, o modelo de dados e a *API* que constituem a *framework* para a implementação destas funcionalidades, sendo que as parametrizações necessárias para a sua integração serão abordadas na secção seguinte relativa à ferramenta.

4.3.1 Componentes customizáveis

Cada componente customizável é constituído por um conjunto de ficheiros *JavaScript,* que funcionam como *templates* através dos quais é possível desenvolver a base de uma loja *online*. Os componentes do carrinho de compras, encomendas e pagamentos são constituídos por diferentes tipos de ficheiros:

- **Modelos -** representam o modelo dos dados inerentes.
- **Rotas -** definem as rotas a incluir no servidor da loja.
- **Controladores** – execução das operações e lógica associada a cada rota suportada pelo servidor da loja.
- **Ações –** definem as ações *Redux* a utilizar no *frontend* da aplicação.
- **Redutores –** definem os redutores a incluir no ficheiro *store.js* para o controlo e gestão do estado global da loja
- **Páginas –** os *screens* ou páginas definidas e acessíveis através do *frontend* da loja *online*.

A Figura 19 visa ilustrar as dependências, a interligação com o servidor e a manipulação dos estados da loja, através de um diagrama que representa os diferentes ficheiros que compõem cada componente customizável. Os componentes disponibilizados pela *framework* contêm os ficheiros base relativos ao *frontend e backend* da loja *online* a gerar.

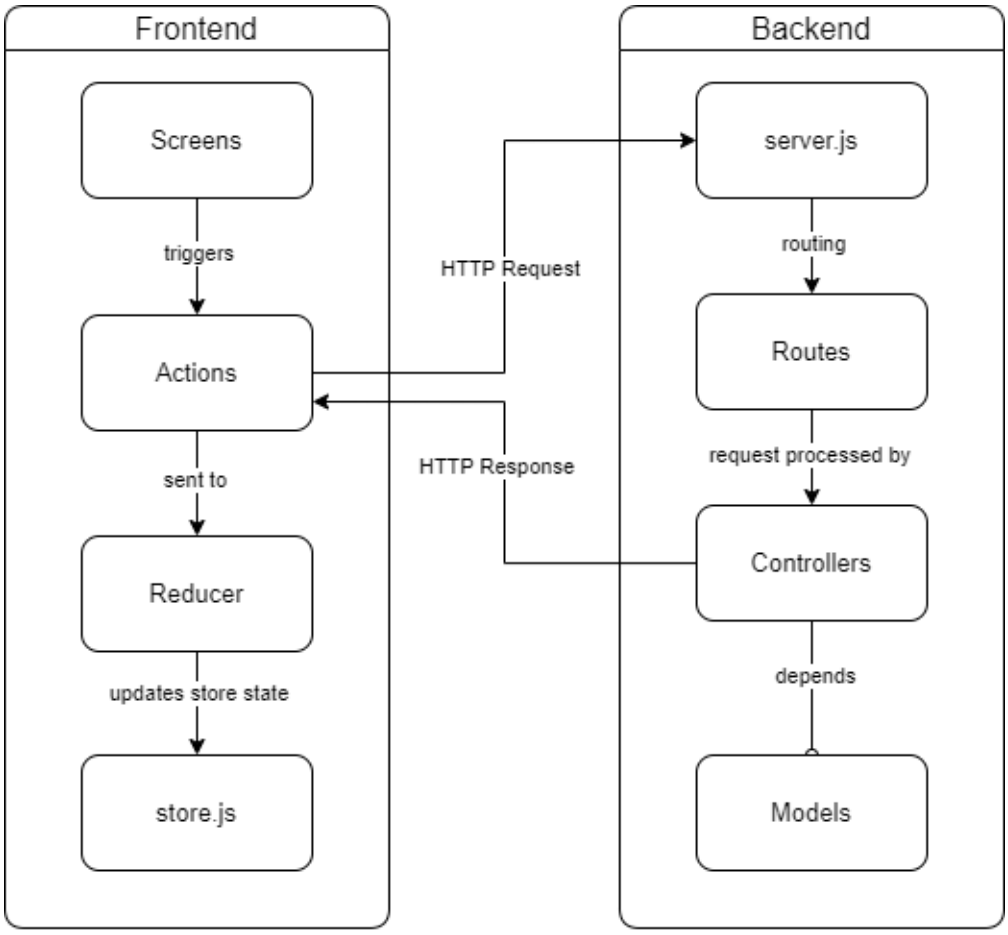

Figura 19 - Arquitetura dos componentes

O*s* ficheiros de modelos, rotas e controladores constituem o *backend*. Quando o servidor recebe um pedido *HTTP*, verifica se a rota está definida e redireciona para o ficheiro respetivo. No ficheiro das rotas é aplicado o *middleware* e o controlador é invocado consoante o pedido recebido. As ações, redutores e páginas constituem o *frontend* inerente a cada uma das funcionalidades.

Os ficheiros relativos às funcionalidades foco da presente dissertação estão representados na Tabela 6.

Tabela 6 - Componentes fundamentais para funcionalidades-chave

Carrinho de compras	*cartModel.js*
	cartController.js
	cartRoutes.js
	cartReducers.js
	cartActions.js
	CartScreen.js
Encomendas e pagamentos	*orderModel.js*
	orderController.js
	orderRoutes.js
	orderReducers.js
	orderActions.js
	paymentMiddleware.js
	paymentRoutes.js
	paymentControllers.js
	PlaceOrderScreen.js
	ShippingScreen.js
	SelectPaymentScreen.js
	OrderScreen.js

A estrutura lógica destes elementos específicos da loja *online* desempenha um papel vital na generalização das funcionalidades de carrinho de compras, encomendas e pagamentos. De modo a interligar estas funcionalidades, deve-se garantir que cada componente recebe os parâmetros necessários para realizar as operações respetivas.

O componente do carrinho de compras recebe um conjunto de objetos como parâmetro, denominado *cartItems*, que contém os detalhes dos produtos adicionados. A Tabela 7 representa as propriedades de cada elemento da variável *cartItems.*

Tabela 7 - Parâmetros do componente carrinho de compras

Campo	*Tipo*	*Descrição*
product	ObjectId	Id do produto
name	String	Nome do produto
image	String	Caminho para a imagem do produto
price	Double	Preço unitário do produto
stockCount	Integer	Quantidade disponível em *stock*
selectedQuantity	Integer	Quantidade selecionada

De forma semelhante, o componente relativo a uma nova encomenda recebe como parâmetro um objeto constituído por: uma lista de produtos (apresentada na Tabela 7), um objeto com os detalhes do endereço de entrega, um campo para o método de pagamento selecionado e os campos relativos aos preços de entrega e total, conforme representado na Tabela 8.

Tabela 8 - Parâmetros do componente de criação de encomendas

Campo	*Tipo*	*Descrição*
cartItems	Array	Lista de produtos no carrinho
shippingAddress	Object	Detalhes do endereço de entrega
taxPrice	Double	Preço de taxas e custo de entrega
totalprice	Double	Preço total da encomenda
paymentMethod	String	Método de pagamento selecionado

No processo de encomendas *standard*, disponibilizado pelos *templates* da *framework,* o utilizador é inicialmente redirecionado para a página relativa ao endereço de entrega da encomenda, sendo posteriormente apresentada a página relativa à seleção do método de

pagamento. Após definir estes detalhes da encomenda, o utilizador é redirecionado para a página onde pode efetivamente criar a encomenda, onde são apresentadas as informações introduzidas anteriormente, a lista de compras e o preços totais e parciais da encomenda. Este processo está representado de uma forma geral na Figura 20 efetuando a abstração da interação com a base de dados e o processamento do pagamento.

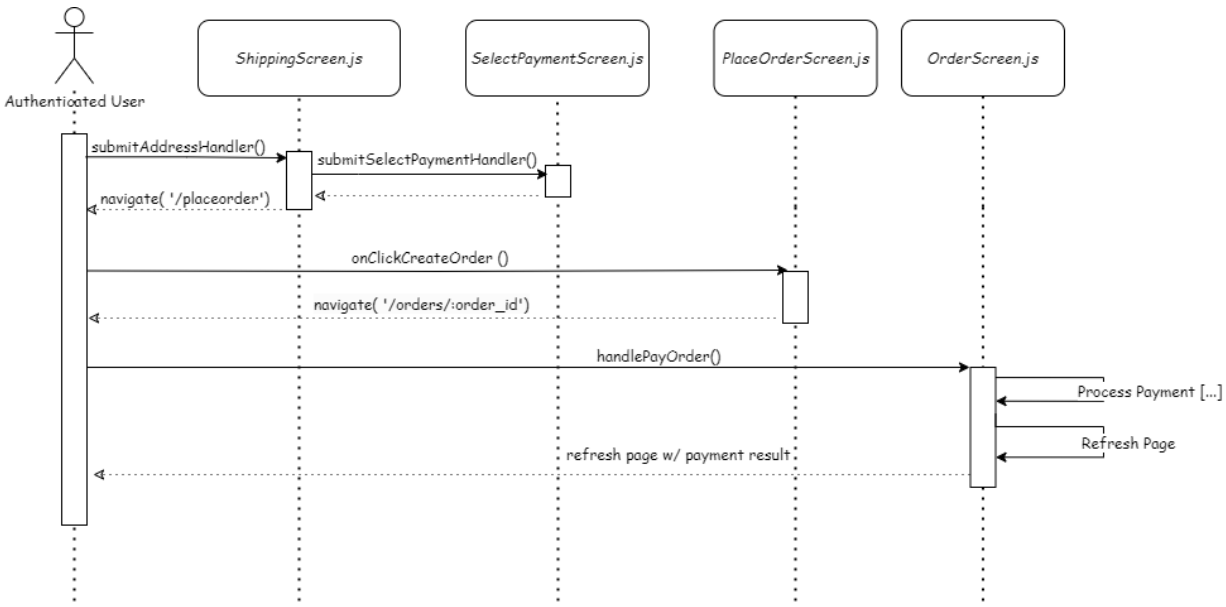

Figura 20 - Diagrama de sequência para criação de encomendas

De modo a integrar os serviços disponibilizados pela *API* do *PayPal*, foi necessário efetuar o registo como desenvolvedor na plataforma do *PayPal e* criar uma aplicação de forma a ser possível obter o *PAYPAL_CLIENT_ID* utilizado nos pedidos efetuados à *API* disponibilizada. Este processo requer a instalação da biblioteca *react-paypal-js.* O ficheiro *OrderScreen.js* utiliza um componente *React* disponibilizado pela biblioteca referida, denominado *PayPalButton.* Quando um utilizador pressiona este botão, um *modal* é apresentado de forma a permitir ao utilizador definir os detalhes do pagamento, iniciando sessão na sua conta de *PayPal* ou fornecendo os detalhes do cartão bancário. O *PayPal* lida com o processamento da transação, verificando a segurança e os detalhes do pagamento, conforme ilustrado na Figura 21.

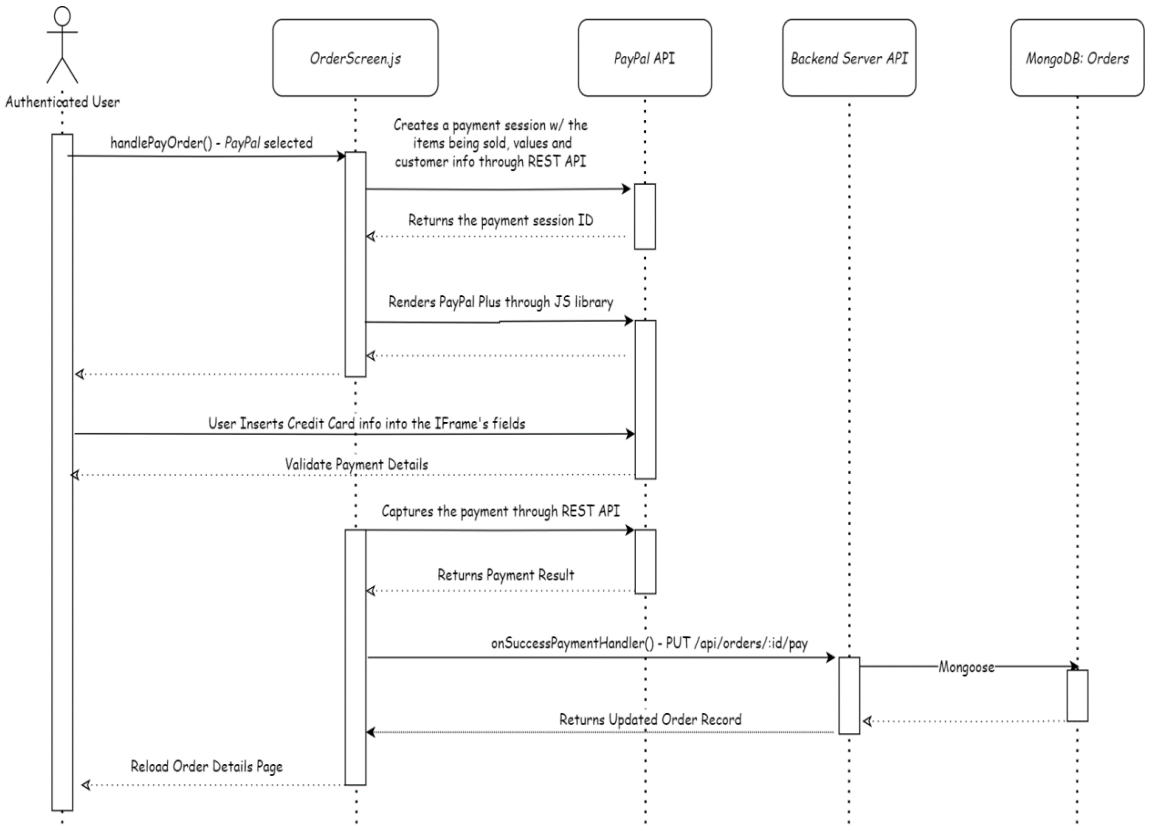

Figura 21 - Diagrama de sequência para pagamento com *PayPal*

Relativamente aos aspetos visuais dos componentes disponibilizados pela *framework*, foram definidos alguns aspetos e elementos que podem ser renderizados condicionalmente, consoante as escolhas do utilizador, conforme ilustrado na Figura 22.

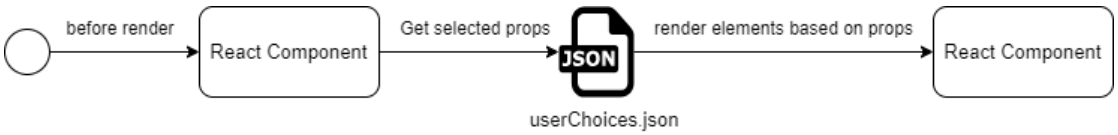

Figura 22 - Renderização condicional baseada no ficheiro *userChoices.json*

A personalização da página do carrinho de compras engloba alguns aspetos visuais como a inclusão da barra de navegação, a definição de mensagens customizadas para o caso de erro e ainda a inclusão de um atalho que permita aceder facilmente a esta página a partir de qualquer ponto da aplicação. Estas personalizações podem ser definidas no formulário disponibilizado pela ferramenta de geração de lojas ou, opcionalmente, através da edição direta do ficheiro *userChoices.*json incluído no ficheiro *.zip* gerado. A Figura 23 representa a renderização condicional relativa aos elementos que constituem a página do carrinho de compras.

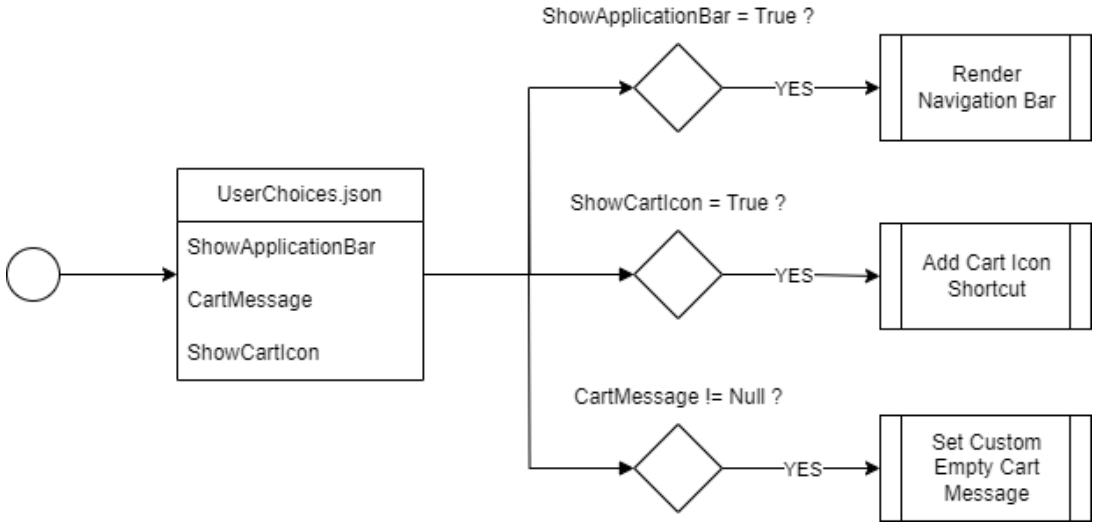

Figura 23 - Renderização condicional do *template CartScreen.js*

De forma semelhante, as diferentes páginas que constituem o processo de encomendas também podem ser personalizadas, permitindo assim a inclusão e customização dos diferentes elementos que constituem a *UI/UX* das páginas inerentes. Os componentes disponibilizados funcionam como *templates* pelo que, o código fonte pode ser editado, posteriormente, de modo a ser possível adicionar outros elementos a renderizar condicionalmente. Na secção seguinte é descrito o modelo dos dados utilizados pela *framework.*

4.3.2 Modelo de dados

A *framework* desenvolvida utiliza uma base de dados disponibilizada pelos serviços da *MongoDB.* Esta base de dados é constituída por cinco coleções distintas: *Orders, Products, Users, Pages* e *Configurations.*

A estrutura dos dados de utilizadores, encomendas e produtos é baseada nos ficheiros modelo (*userModel.js*, *orderModel.js e productModel.js)* sendo constituídos por todos os campos *standard* representados na Figura 5 - Modelo de dados da loja *online* e adicionalmente por novos campos definidos pelo utilizador durante a criação de uma nova loja, conforme ilustrado na Figura 24.

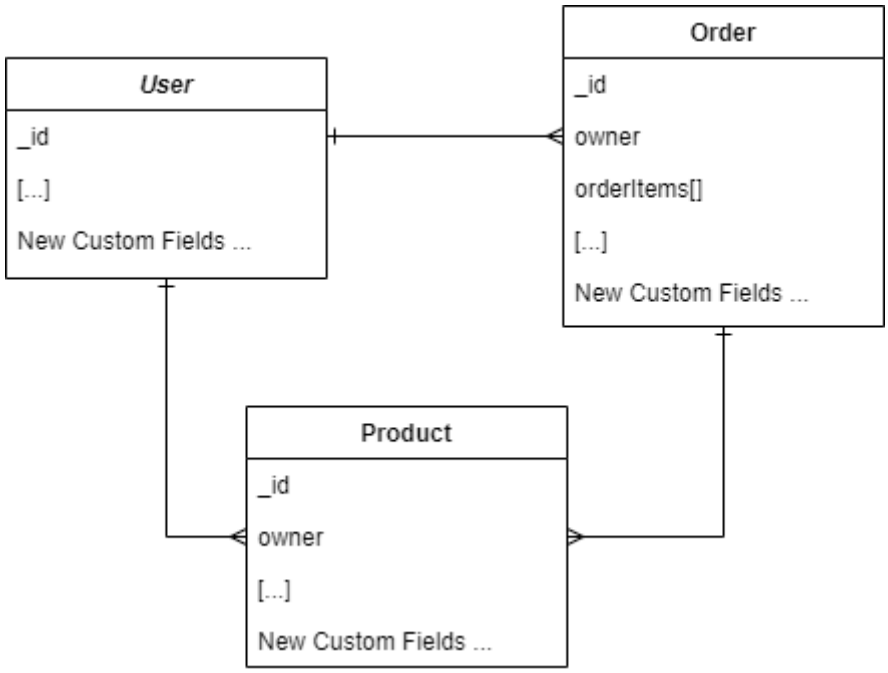

Figura 24 - Modelo de dados da *framework – Order, product e user*

Relativamente às coleções *Configurations e Pages,* têm como objetivo definir as propriedades inerentes à configuração de uma nova loja e a respetiva personalização do utilizador relativa a cada uma das funcionalidades e páginas que compõem a loja a gerar, conforme ilustrado na Figura 25. Cada loja que se pretenda gerar é constituída pelos campos relativos aos detalhes da loja (nome, tema, etc.) e pelas funcionalidades ou páginas selecionadas

pelo utilizador. Cada funcionalidade ou página é caraterizada pelas propriedades definidas pelo utilizador (incluir barra de navegação, customização de mensagens, etc.) durante o processo de criação da loja.

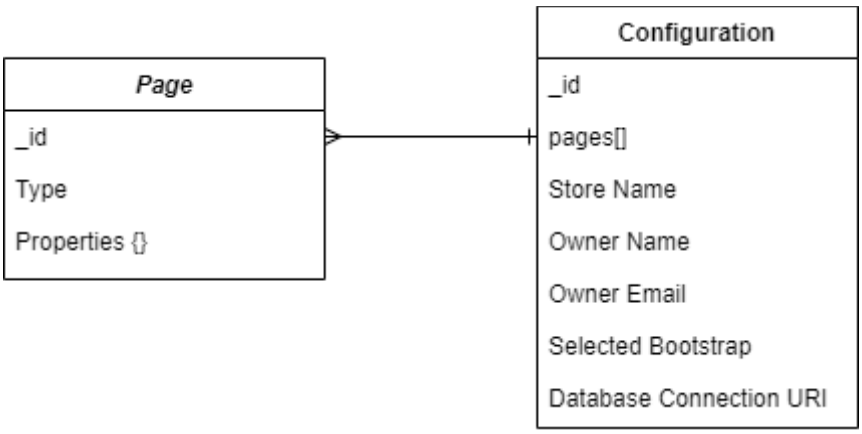

Figura 25 - Modelo de dados da framework – *Configuration e page*

O campo *Type* define a funcionalidade a adicionar à loja, pelo que, um registo da coleção *Configuration* pode ser constituído por vários registos da coleção *Page* desde que tenham diferentes valores para este campo. A inclusão de novas funcionalidades é facilitada pelo facto de se tratar de uma base de dados não relacional, o que permite uma maior versatilidade dos dados e dos registos inerentes a cada uma das coleções.

Cada documento da coleção *Configurations,* representa as configurações e escolhas do utilizador relativas à personalização de uma loja *online* e contém campos como o nome do proprietário, o email, o título da loja, as funcionalidades a incluir, entre outros. Cada funcionalidade a incluir é caraterizada por um objeto, referente a um registo da coleção *Pages*.

A Tabela 9 representa todos os campos que constituem o modelo de dados da *framework* desenvolvida.

Tabela 9 - Modelo de dados da *framework*

Campo	*Tipo*	*Descrição*
ownerName	String	Nome do utilizador
ownerEmail	String	Email do utilizador
Title	String	Nome da loja
landingPage	String	Página inicial da loja
selectedBootstrap	String	Nome do tema *bootstrap* selecionado
mongoURI	String	URI encriptado da base de dados da loja
includeProductListScreen	Boolean	Incluir a página de gestão de produtos
includeOrderListScreen	Boolean	Incluir a página de gestão de encomendas
includeUserListScreen	Boolean	Incluir a página de gestão de utilizadores
includeStatisticsScreen	Boolean	Incluir a página de estatísticas da loja
includeAuthPages	Boolean	Incluir as páginas de autenticação
includeHomeScreen	Boolean	Incluir página com catálogo e filtragem
includeShopCart	Boolean	Incluir página de carrinho de compras
includePlaceOrder	Boolean	Incluir criação e pagamento de encomendas.
productListScreen	Object	Propriedades - página de gestão de produtos
orderListScreen	Object	Propriedades - página de gestão de encomendas
userListScreen	Object	Propriedades – página de gestão de utilizadores.
statisticsScreen	Object	Propriedades da página de estatísticas da loja
authPages	Object	Propriedades das páginas de autenticação
homeScreen	Object	Propriedades da página de catálogo e filtragem de
cartScreen	Object	Propriedades da página de carrinho de compras
placeOrderScreens	Object	Propriedades das páginas de criação e pagamento de encomendas

A coleção foi desenvolvida de modo a permitir integrar novas funcionalidades. A inclusão de cada funcionalidade é definida por um campo do tipo booleano, sendo que as configurações e propriedades inerentes a cada página ou funcionalidade são encapsuladas num objeto, de modo a ter um registo referente a cada página selecionada pelo utilizador.

4.3.3 API do servidor

No servidor da plataforma, foi implementada uma *API* de modo a suportar os pedidos efetuados pelo *frontend,* de modo similar à *API* desenvolvida para a loja *online* descrita no capítulo anterior. A *API* do servidor é responsável por responder a todos os pedidos relativos às operações *CRUD* sobre os registos na coleção das *Configurations* da base de dados, conforme representado na Tabela 10.

Tabela 10 - Rotas da ferramenta de geração de lojas

Funcionalidade	Rota	HTTP
Obter todas as configurações de lojas	/api/store/	GET
Obter configuração de loja por ID	/api/store/:id	GET
Atualizar configuração de loja	/api/store/:id	PUT
Apagar uma configuração de loja	/api/store/:id	DELETE

A *API* é ainda reforçada pela utilização de *middlewares*, responsáveis pela captura e tratamento de erros, sendo que, no contexto da *framework*, não foram implementados *middlewares* de autenticação/autorização.

4.4 Ferramenta para integração de componentes

A ferramenta de geração de lojas tem como objetivo modificar os componentes customizáveis disponibilizados pela *framework* de forma a gerar a loja *online* com as propriedades definidas pelo utilizador, conforme ilustrado na Figura 26. Quando o utilizador opta por incluir uma funcionalidade, várias ações são desencadeadas no código para adicionar os componentes necessários, modificando os *templates* definidos na diretoria *exports*.

. A ferramenta desenvolvida permite ao utilizador escolher as funcionalidades que pretende adicionar à sua loja e ainda personalizar as diferentes páginas da aplicação. O ficheiro *.zip* gerado, contém ainda um ficheiro *userChoices.json* com as propriedades definidas pelo utilizador, sendo que estes valores são utilizados pelos componentes *React* de forma a renderizar certos elementos de forma condicional, baseado nas opções do utilizador, conforme ilustrado na Figura 22 apresentada anteriormente.

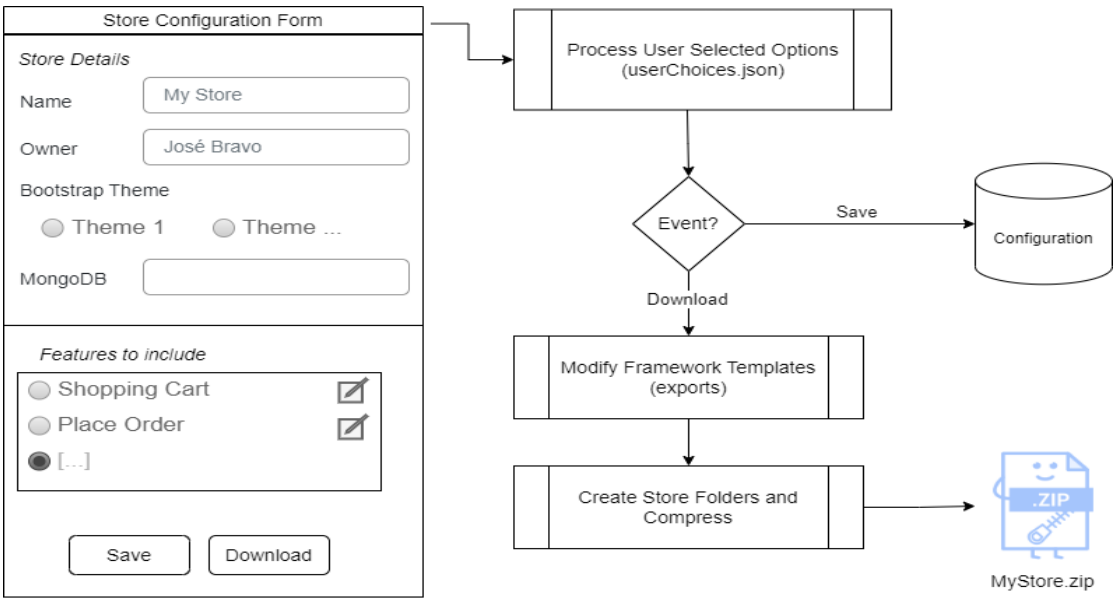

Figura 26 - Funcionamento da ferramenta

Na próxima secção são descritos os passos necessários para a integração dos diferentes componentes e respetivas funcionalidades, abordando as parametrizações efetuadas para a utilização de cada componente-chave, tendo como foco os componentes inerentes ao carrinho de compras, encomendas e pagamentos. A ferramenta desenvolvida utiliza os componentes customizáveis e a *API* disponibilizados pela *framework*.

4.4.1 Registo e autenticação

A funcionalidade de registo e autenticação disponibiliza as páginas e processos de registo e início de sessão, sendo que o utilizador pode personalizar alguns aspetos visuais de cada uma destas páginas. Quando esta funcionalidade é incluída pelo utilizador, são desencadeadas as seguintes ações:

1. Os *templates LoginScreen.js* e *RegisterScreen.js* são adicionados e as rotas de navegação, representadas na Tabela 11, relativas ao *frontend*, são importadas no ficheiro *App.js:*

Tabela 11 - Rotas de *frontend* para registo e autenticação

Funcionalidade	Rota
Navegar para a página de início de sessão	/login
Navegar para a página de registo	/register

2. Os *templates userActions.js*, *userConstants.js* e *userReducers.js* são adicionados ao *frontend* da aplicação, e no ficheiro *store.js* adiciona-se os *reducers* respetivos.
3. Relativamente ao *backend,* os *templates userRoutes.js* e *userControllers.js* são adicionados e, no ficheiro *server.js* são adicionadas as rotas da *API* do servidor relativas à gestão de utilizadores.

4.4.2 Gestão das coleções e estatísticas da loja

De modo a incluir as páginas de gestão dos registos da base de dados, é seguida a mesma lógica descrita para a funcionalidade de registo e autenticação, utilizando os ficheiros, rotas e *reducers* inerentes a cada funcionalidade.

1. Os *templates ProductListScreen.js, OrderListScreen.js, UserListScreen.js, StatisticsScreen.js, ProductScreen.js, OrderScreen.js, UserScreen.js* são adicionados e as rotas de navegação relativas ao *frontend,* representadas na Tabela 12, são importadas no ficheiro *App.js:*

Tabela 12 - Rotas de *frontend* para gestão de coleções e estatísticas da loja

Funcionalidade	Rota
Navegar para a página de gestão de produtos	/admin/productlist
Navegar para a página de gestão de encomendas	/admin/orderlist
Navegar para a página de gestão de utilizadores	/admin/userlist
Navegar para a página administrativa de detalhes de produto	/admin/productlist/:id
Navegar para a página administrativa de detalhes de encomenda	/admin/orderlist/:id
Navegar para a página administrativa de detalhes de utilizador	/admin/userlist/:id
Navegar para a página de estatísticas da loja	/admin/statistics

2. Os *templates userActions.js*, *userReducers.js, orderActions.js*, *orderReducers.js, productActions.js* e *productReducers.js* são adicionados ao *frontend* da aplicação, e no ficheiro *store.js* adiciona-se os *reducers* inerentes.
3. Relativamente ao *backend,* os *templates* relativos aos controladores e rotas são adicionados e, no ficheiro *server.js* são importadas as rotas da *API* do servidor relativas à gestão de utilizadores, encomendas e produtos.

4.4.3 Catálogo e filtragem de produtos

A funcionalidade referente ao catálogo e filtragem de produtos inclui uma página com a lista de produtos disponíveis para compra e uma página de detalhes de produto. Para a integração destas funcionalidades foram efetuados os seguintes passos:

1. Os *templates HomeScreen.js* e *ProductScreen.js* são adicionados e as rotas de navegação relativas ao *frontend* representadas na Tabela 13, são importadas no ficheiro *App.js:*

Tabela 13 - Rotas de *frontend* para catálogo e filtragem de produtos da loja

Funcionalidade	Rota
Navegar para a página de gestão de produtos	/home
Navegar para a página de detalhes do produto	/product/:id

2. Os *templates productActions.js* e *productReducers.js* são adicionados ao *frontend* da aplicação, e no ficheiro *store.js* adiciona-se os *reducers* inerentes.
3. Relativamente ao *backend,* os *templates* relativos aos controladores e rotas são adicionados e, no ficheiro *server.js* são importadas as rotas da API do servidor relativas à listagem e filtragem de produtos.

4.4.4 Carrinho de compras

O componente do carrinho de compras não requer autenticação nem registo. Quando o utilizador opta por incluir o carrinho de compras, os *templates* respetivos são importados e modificados pela ferramenta de modo a adicionar as rotas e *reducers* à aplicação, conforme ilustrado na Figura 27.

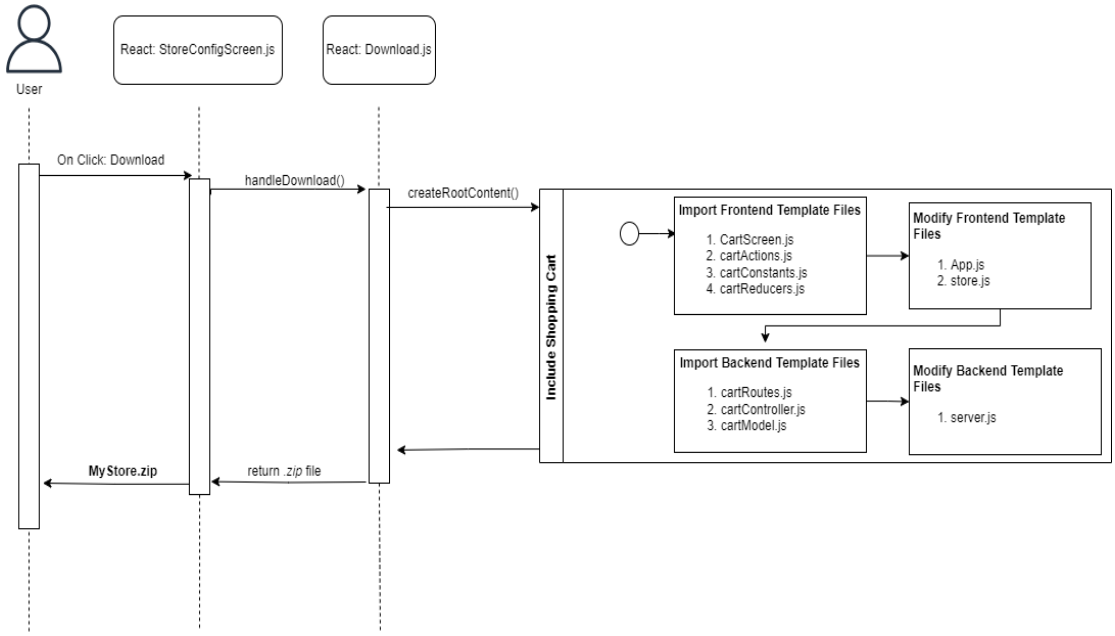

Figura 27 - Processo de download - Inclusão do carrinho de compras

O componente do carrinho de compras recebe como parâmetro ou através do estado global, uma lista de objetos que define os produtos e respetivas quantidades no carrinho de compras, sendo posteriormente calculado o preço total. A Tabela 7 apresentada na secção anterior, representa a estrutura de cada elemento da lista de produtos no carrinho de compras – *cartItems* - recebida como parâmetro ou através do estado global da aplicação, no componente do carrinho de compras.

Para a integração do carrinho de compras foram efetuados os seguintes passos:

1. O *template CartScreen.js* é adicionado e a rota de navegação relativa ao *frontend,* representada na Tabela 14, é importada no ficheiro *App.js:*

Tabela 14 - Rota de *frontend* para o carrinho de compras

Funcionalidade	Rota
Navegar para a página de carrinho de compras	/cart

2. Os *templates cartActions.js*, *cartContants.js* e *cartReducers.js* são adicionados ao *frontend* da aplicação, e no ficheiro *store.js* adiciona-se os *reducers* respetivos.
3. Relativamente ao *backend,* os *templates* relativos aos controladores e rotas são adicionados e, no ficheiro *server.js* são importadas as rotas da API do servidor relativas à gestão dos produtos no carrinho de compras

4.4.5 Encomendas e pagamentos

A funcionalidade de encomendas disponibiliza as seguintes páginas: criação de encomenda, detalhes de entrega e seleção do método de pagamento (neste caso será o *PayPal*) e ainda a página de detalhes da encomenda. O processo de criação e pagamento de encomendas requer que o utilizador tenha sessão iniciada, através da utilização de *middlewares* onde se verifica a validade do *Token JWT* inserido no cabeçalho de cada pedido efetuado ao servidor. De modo a incluir este processo foram efetuados os seguintes passos:

1. Os *templates PlaceOrderScreen.js, ShippingScreen.js, SelectPaymentScreen.js e OrderScreen.js são* adicionados e as rotas de navegação relativa ao *frontend*, representadas na Tabela 15 são importadas no ficheiro *App.js:*

Tabela 15 - Rotas de criação e pagamento de encomendas

Funcionalidade	Rota
Navegar para a página de criação de uma encomenda	/placeorder
Navegar para a página de detalhes de entrega de uma encomenda	/shipping
Navegar para a página de seleção do método de pagamento de uma encomenda	/payment
Navegar para a página de detalhes de uma encomenda	/orders/:id

2. Os *templates orderActions.js*, *orderContants.js* e *orderReducers.js* são adicionados ao *frontend* da aplicação, e no ficheiro *store.js* adiciona-se os *reducers* respetivos.
3. Relativamente ao *backend,* os *templates* relativos aos controladores e rotas são adicionados e, no ficheiro *server.js* são importadas as rotas da *API* do servidor relativas à criação e pagamento de encomendas através do ficheiro *orderRoutes.js*.

O componente relativo à criação de uma encomenda recebe um objeto como parâmetro, como referido anteriormente. Este objeto é constituído por: uma lista de produtos no carrinho, um objeto com os detalhes do endereço de entrega, um campo para o método de pagamento selecionado e os campos relativos aos preços de entrega e total, conforme representado na Tabela 8 - Parâmetros do componente de criação de encomendas.

De modo a integrar os serviços disponibilizados pela *API* do *PayPal*, foi necessário efetuar o registo como desenvolvedor na plataforma do *PayPal e* criar uma aplicação de forma a ser possível obter o *PAYPAL_CLIENT_ID* utilizado nos pedidos efetuados á *API* disponibilizada, conforme descrito na Figura 21 - Diagrama de sequência para pagamento com *PayPal*.

O processo inerente à criação e pagamento de encomendas pressupõe a existência de produtos no carrinho de compras. A utilização das funcionalidades pode ser conseguida de forma independente, isto é, o utilizador pode optar por incluir o carrinho de compras e não incluir o processo de encomenda, ou vice-versa. No entanto, caso o utilizador pretenda incluir o processo de criação de encomendas sem o carrinho de compras, deve garantir que é fornecido um objeto idêntico ao carrinho de compras, como parâmetro para a página de criação de

encomendas, ou alternativamente através do estado global da aplicação definido no ficheiro *store.js*.

No seguinte capítulo é efetuada a demonstração de utilização da plataforma de geração de lojas, ilustrando o processo de configuração de uma loja através do preenchimento do formulário disponibilizado pela ferramenta. Posteriormente, é efetuada a demonstração da loja online gerada tendo como base as preferências do utilizador.

5. Demonstração de utilização

Neste capítulo, serão apresentados cenários de uso típicos que um utilizador final pode encontrar ao interagir com a ferramenta. Através de capturas de ecrã, diagramas de fluxo e descrições passo-a-passo, é fornecida uma visão abrangente que permita avaliar tanto a usabilidade como a eficácia das funcionalidades implementadas.

5.1 Configuração e personalização de uma loja

A ferramenta de geração de lojas consiste em três componentes fundamentais responsáveis por: criar, editar e listar os registos de configurações existentes na base de dados. A Figura 28 representa a página de configuração de uma nova loja.

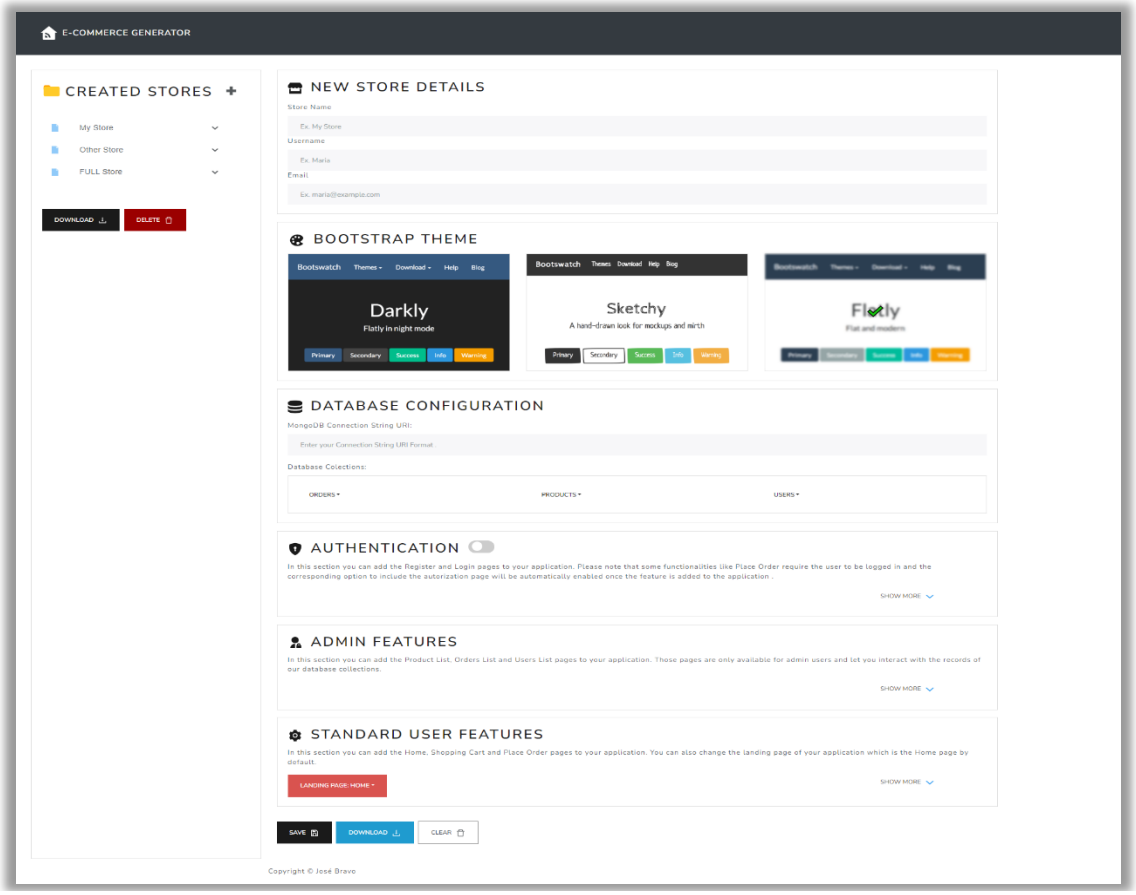

Figura 28 - Configurar nova loja

A página de criação e edição de registos são idênticas, sendo que disponibilizam a lista de configurações existentes e um formulário onde o utilizador pode personalizar e configurar a loja *online* que pretende gerar. No processo de edição os valores do formulário são populados automaticamente através dos dados armazenados na base de dados. O utilizador deve preencher os detalhes da loja e selecionar as opções e funcionalidades pretendidas, sendo que pode personalizar alguns aspetos e pré-visualizar cada uma das páginas.

Na secção de configuração da base de dados, representada na Figura 29, o utilizador pode fornecer a chave de conexão à base de dados da loja a gerar, caso contrário, após o *download* ser efetuado deve preencher o parâmetro *MONGO_URI* definido no ficheiro *.env* da loja *online* gerada.

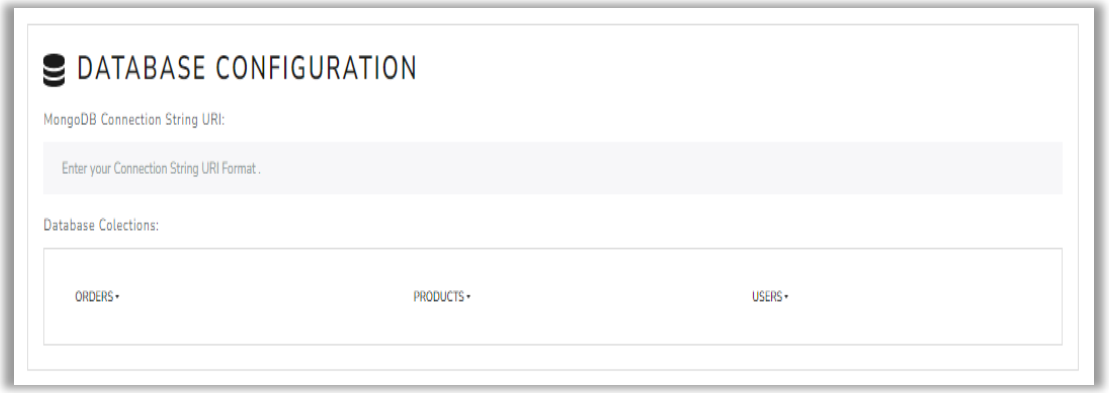

Figura 29 - Configuração da base de dados da loja

O utilizador pode também adicionar novos campos aos modelos inerentes às coleções utilizadas pela loja *online* a gerar.

Na secção de autenticação, o utilizador pode optar por incluir as funcionalidades de início de sessão e registo, conforme ilustrado na Figura 30, sendo que pode pré-visualizar ambas as páginas e incluir uma barra de navegação.

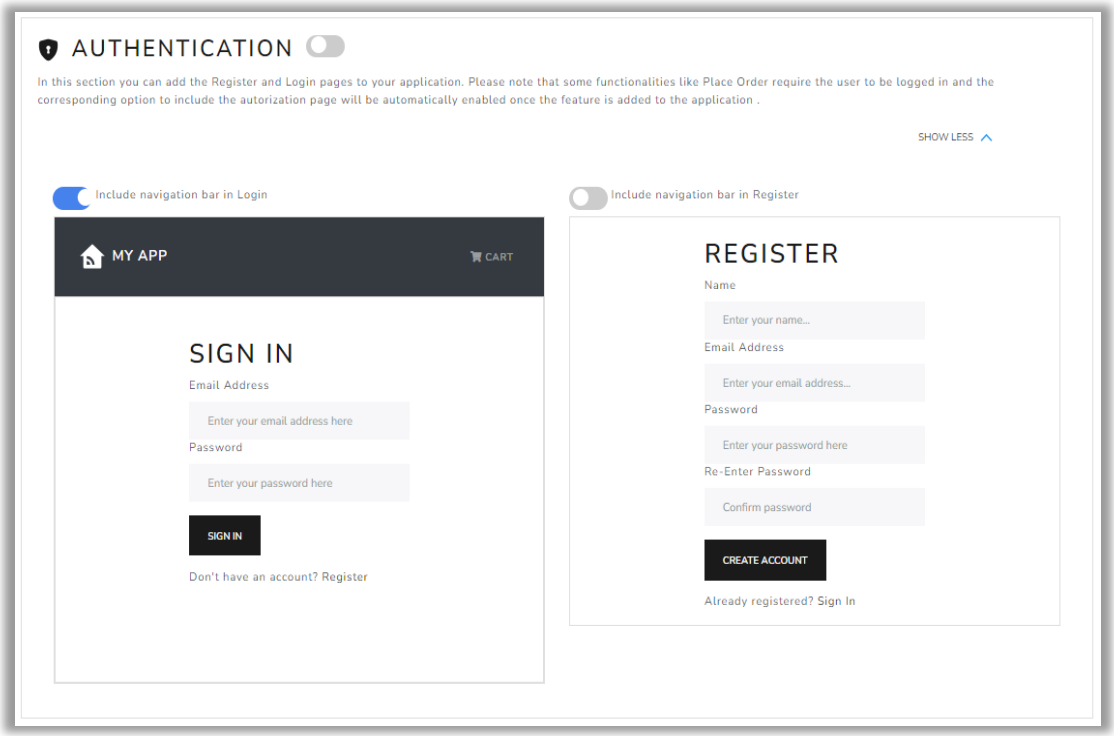

Figura 30 - Incluir funcionalidade de autenticação

Na secção das funcionalidades administrativas, representada na Figura 31, o utilizador pode incluir as páginas de gestão dos registos de produtos, utilizadores e encomendas presentes na base de dados da loja, como também a página relativa às estatísticas, ilustrada na Figura 32.

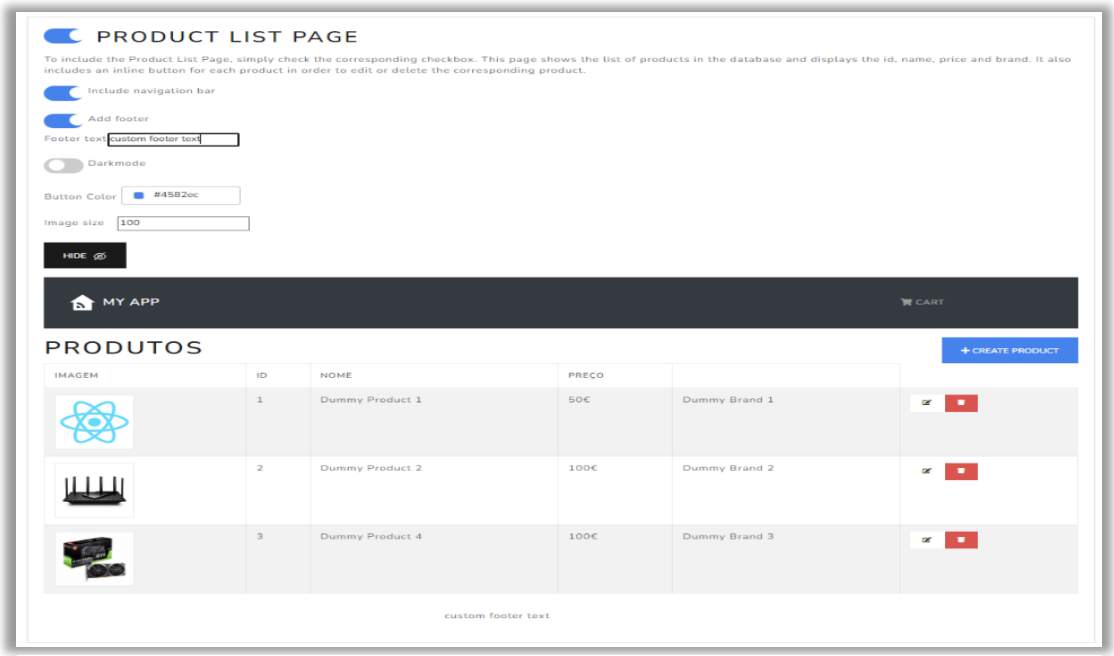

Figura 31 - Incluir página de gestão de produtos

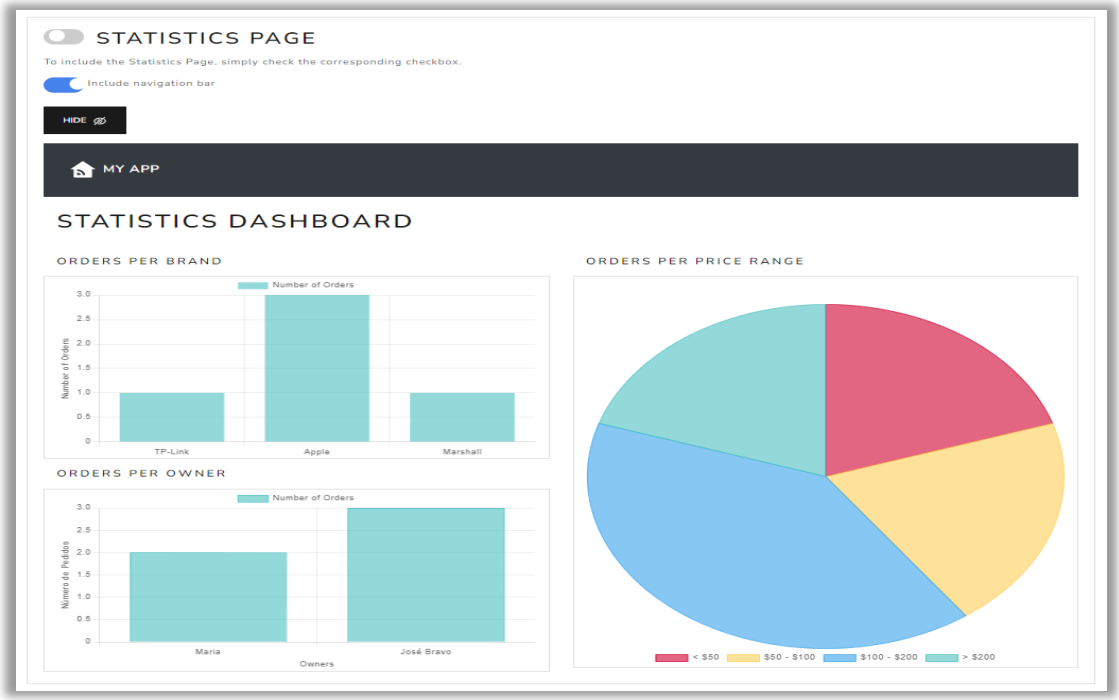

Figura 32 - Incluir página de estatísticas da loja

Na secção das funcionalidades do utilizador comum, Figura 33, é possível definir a página inicial da loja e incluir as seguintes páginas e funcionalidades: catálogo e filtragem de produtos, carrinho de compras, encomendas e pagamentos.

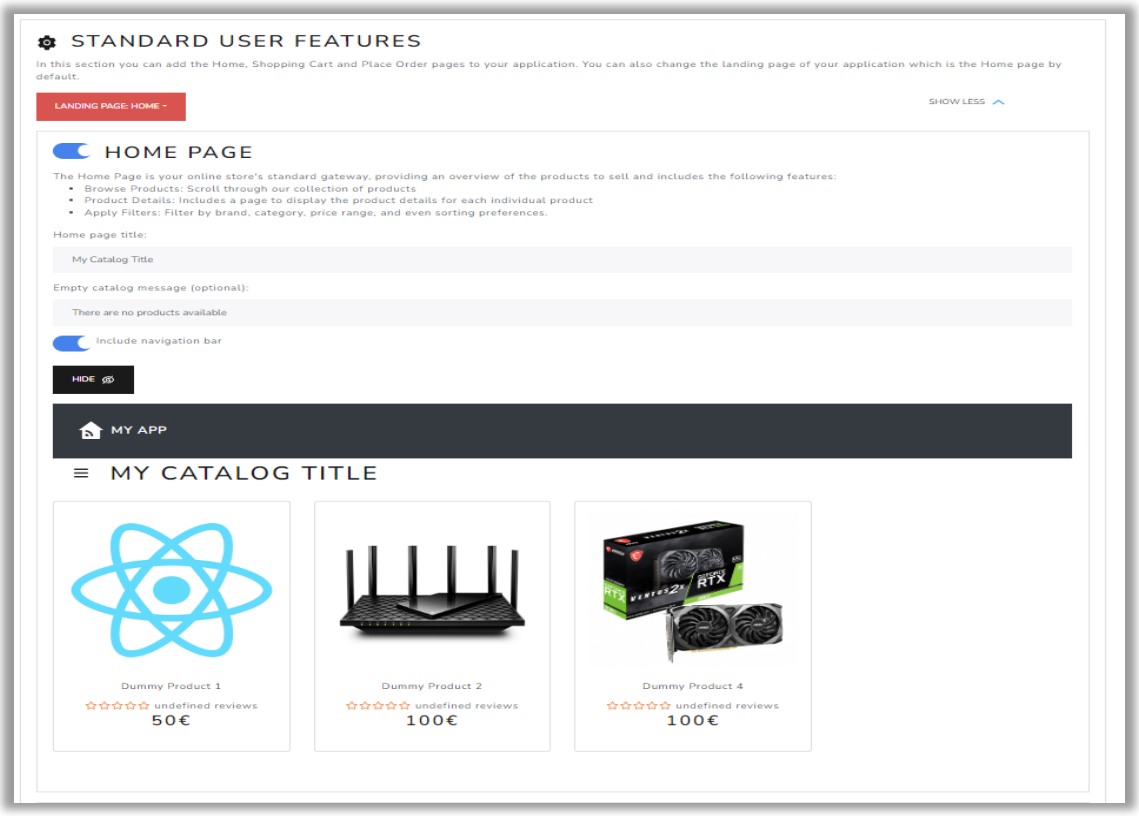

Figura 33 - Incluir catálogo de produtos

Relativamente à página do catálogo de produtos, o utilizador pode definir um título para a página, incluir a barra de navegação e definir a mensagem a mostrar quando não existem produtos disponíveis.

Na página de carrinho de compras, ilustrada na Figura 34, o utilizador pode optar por incluir a barra de navegação, adicionar um ícone à barra de navegação que permite aceder diretamente à página do carrinho de compras e definir a mensagem a mostrar quando não existem produtos no carrinho.

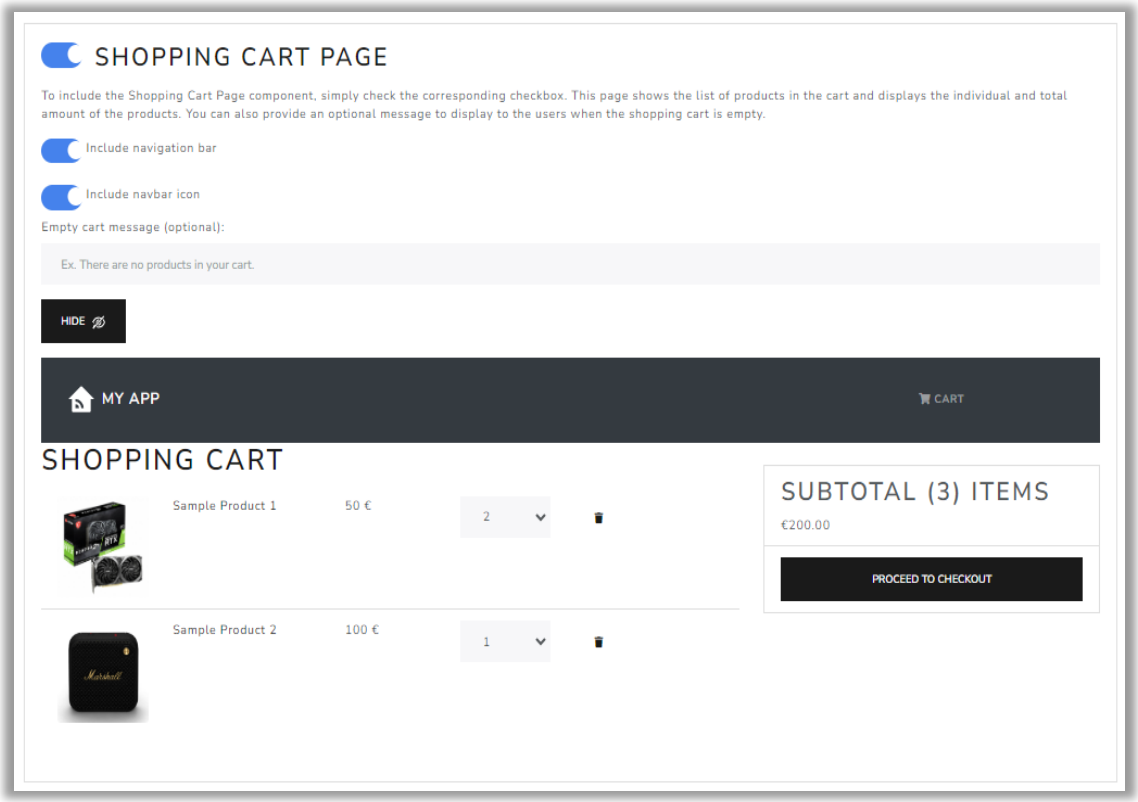

Figura 34 - Incluir carrinho de compras

Em relação ao processo de encomenda e pagamento, o utilizador pode incluir esta funcionalidade e incluir a barra de navegação em cada uma das páginas que constituem este processo, como as páginas de criação de encomendas, definição do endereço e seleção do método de pagamento, conforme ilustrado na Figura 35.

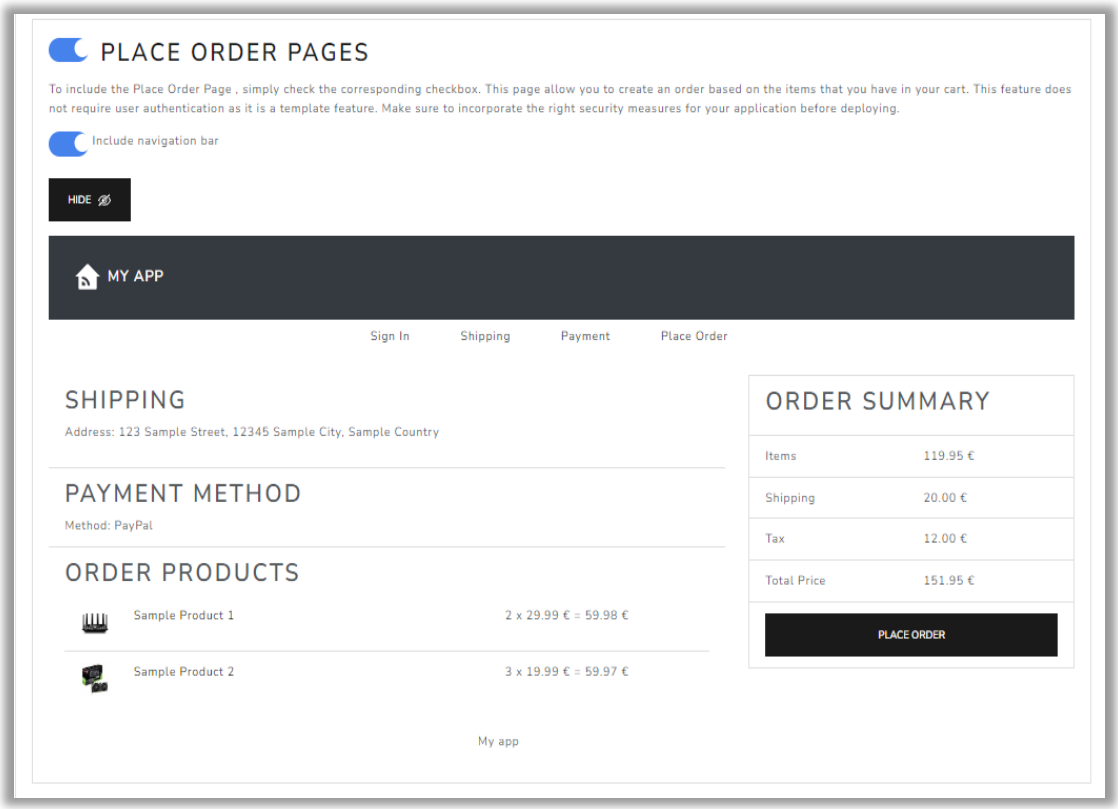

Figura 35 - Incluir processo de encomendas e pagamentos

Após preencher os detalhes e selecionar as funcionalidades desejadas da loja, o utilizador pode descarregar diretamente o código fonte da aplicação, em formato *.zip*, ou caso pretenda terminar a configuração mais tarde, pode optar por guardar estas configurações na base de dados através do componente *Save.js*.

Quando o utilizador carrega no botão para descarregar o código fonte da loja, o componente *Download.js* verifica as opções do utilizador e exporta os ficheiros respetivos, sendo que cria também um ficheiro *userChoices.json* que é utilizado pelos componentes *react* da aplicação gerada, de forma a renderizar dinamicamente os elementos com base nas configurações da loja. A Figura 36 representa a página de edição de um registo.

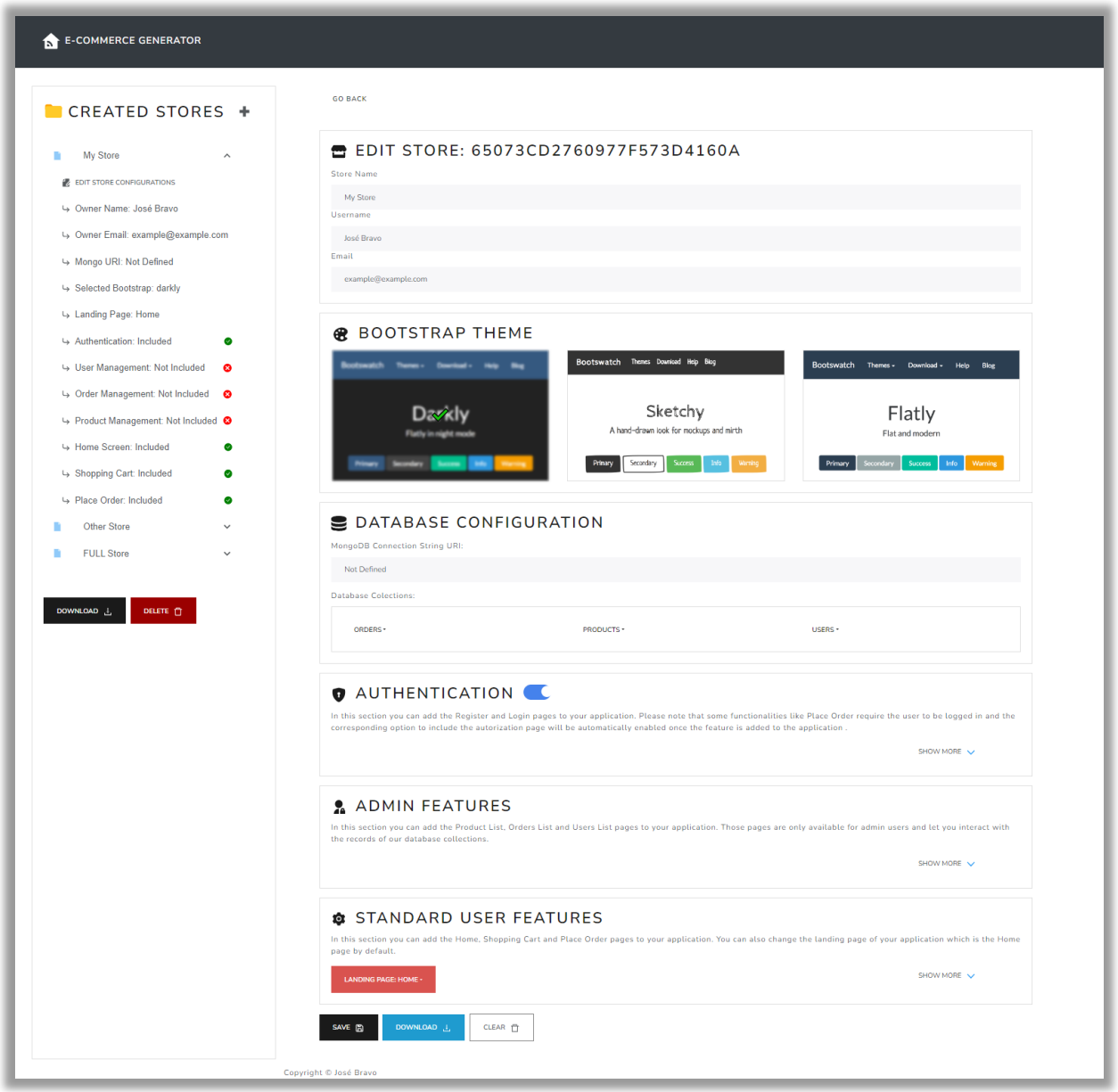

Figura 36 - Edição de configurações de uma loja

Na seguinte secção é efetuada a demonstração de utilização da loja gerada após efetuar o *download* do ficheiro *.zip* com o código fonte da aplicação.

5.2 Loja parcial gerada

O ficheiro *.zip* transferido contém as diretorias *frontend* e *backend*, um ficheiro *package.json*, que contém a lista de dependências e os *scripts* de inicialização concorrente do servidor e da aplicação *react*, e um ficheiro *README.md* que contém as seguintes instruções para iniciar a aplicação da loja:

1. Extrair a diretoria para uma localização à escolha do utilizador.
2. Abrir os terminais das diretorias *frontend e backend* e executar o seguinte comando:
   a. $ npm install
3. Abrir o terminal na diretoria principal e executar o comando:
   a. $ npm run dev

5.2.1 Ficheiros e configurações

Após executar estas instruções, a loja gerada irá iniciar o *frontend* da aplicação na porta 3000, e o servidor na porta 5000, por omissão. A Figura 37 ilustra as propriedades do ficheiro *userChoices.json* criado para a loja *"My Store"* representada anteriormente, na Figura 36.

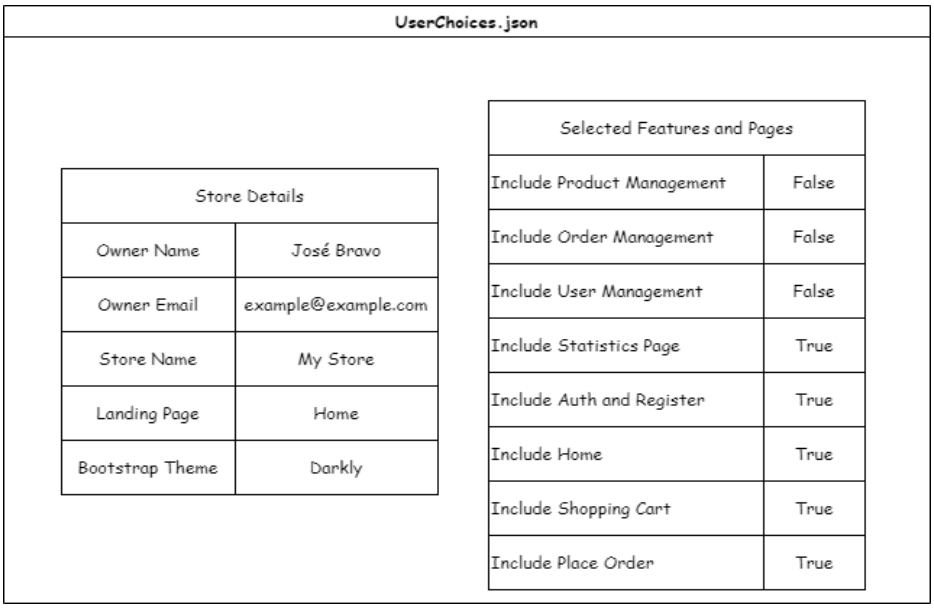

Figura 37 - Ficheiro userChoices.josn

A página inicial é definida com base nas escolhas do utilizador, sendo que a renderização de certos elementos em cada página é controlada pelo ficheiro auxiliar *userChoices.json* gerado durante o processo de *download.* Com base nestas configurações, é definida a lista de ficheiros a serem exportados e incluídos na loja gerada, durante o processo de *download,* disponibilizado pela ferramenta de geração de lojas. A Tabela 16 lista todos os ficheiros e respetivas condições para a sua inclusão.

Tabela 16 - Ficheiros da loja gerada

Ficheiro	Condição para ser incluído	Descrição
index.js	Sempre	Ponto de entrada para o *frontend*.
server.js	Sempre	Ponto de entrada para o *backend*.
package.json	Sempre	Ficheiro de configuração do projeto.
userController.js	includeAuthPages	Controlador para a autenticação de utilizadores.
orderController.js	includePlaceOrder	Controlador para a gestão de encomendas.
userActions.js	includeAuthPages	Ações Redux para a autenticação de utilizadores.
orderActions.js	includePlaceOrder	Ações Redux para a gestão de encomendas.
userReducers.js	includeAuthPages	Redutores Redux para a gestão e autenticação de utilizadores.
orderReducers.js	includePlaceOrder	Redutores Redux para a gestão de encomendas.
userConstants.js	includeAuthPages	Constantes Redux para a gestão e autenticação de utilizadores.
orderConstants.js	includePlaceOrder	Constantes Redux para a gestão de encomendas.
ProductEditScreen.js	includeProductListScreen	Ecrã para a edição de produtos.
ProductListScreen.js	includeProductListScreen	Ecrã para a listagem de produtos.
HomeScreen.js	includeHomeScreen	Ecrã inicial da aplicação.
ProductScreen.js	includeHomeScreen	Ecrã para a visualização detalhada de um produto.
CartScreen.js	includeShopCart	Ecrã para o carrinho de compras.
LoginScreen.js	includeAuthPages	Ecrã para o login de utilizadores.
RegisterScreen.js	includeAuthPages	Ecrã para o registo de novos utilizadores.
PlaceOrderScreen.js	includePlaceOrder	Ecrã para a finalização da encomenda.
ShippingScreen.js	includePlaceOrder	Ecrã para a seleção do método de envio.
SelectPaymentScreen.js	includePlaceOrder	Ecrã para a seleção do método de pagamento.
OrderScreen.js	includePlaceOrder	Ecrã para a visualização detalhada de uma encomenda.
EditOrderScreen.js	includePlaceOrder	Ecrã para a edição de encomendas.
OrderListScreen.js	includePlaceOrder	Ecrã para a listagem de encomendas.

5.2.2 Utilização da loja e respetivas funcionalidades

Após analisar o ficheiro, concluímos quais são os detalhes da loja, as funcionalidades que foram incluídas e as propriedades inerentes a cada uma. Neste caso a loja gerada tem como página inicial a página do catálogo com filtragem de produtos, Figura 38, designada por *Home* e inclui as funcionalidades de estatísticas, autenticação, carrinho de compras e encomendas. Cada funcionalidade é personalizada através de um objeto que define as suas propriedades de renderização, como a inclusão da barra de navegação, as cores dos botões, as mensagens a apresentar ao utilizador, entre outras.

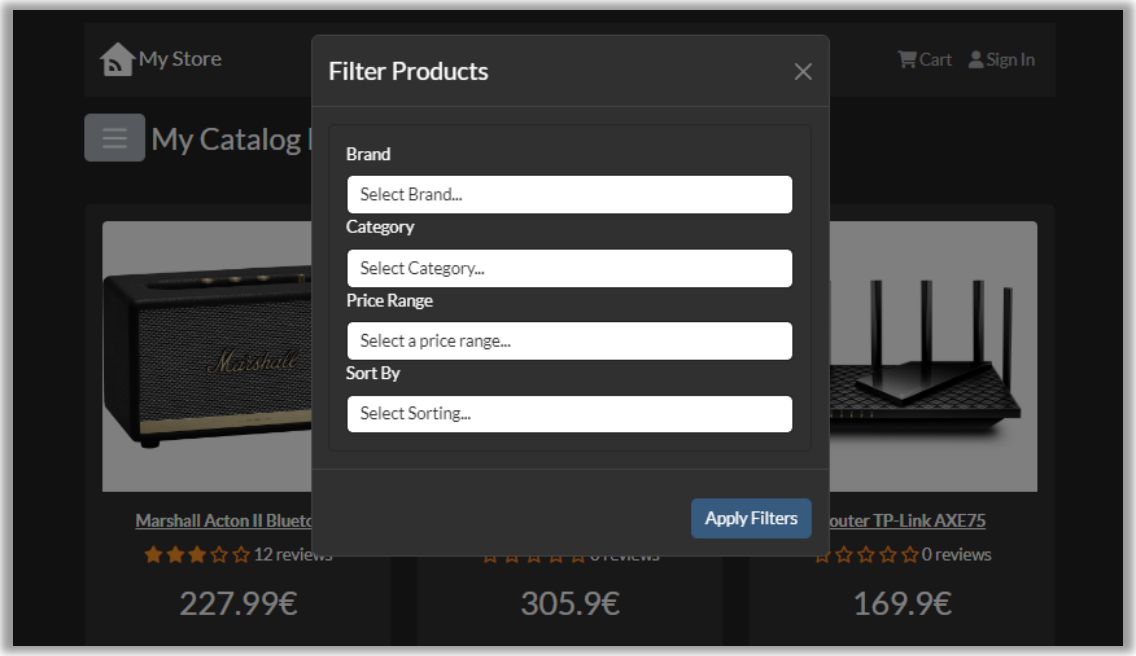

Figura 38 - Página inicial da loja gerada

De forma semelhante, temos a página do carrinho de compras que foi definida com as propriedades representadas na Figura 39.

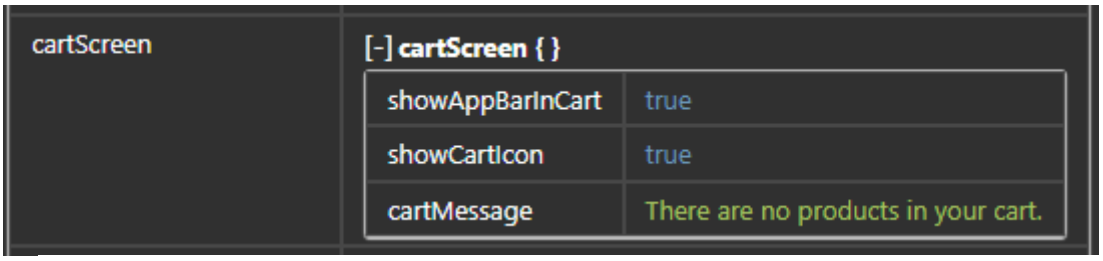

Figura 39 – Propriedades de renderização da página de carrinho de compras

Com base nestas propriedades, a barra de navegação é incluída sendo que também inclui o ícone que permite ao utilizador aceder diretamente ao seu carrinho de compras, conforme ilustrado na Figura 40.

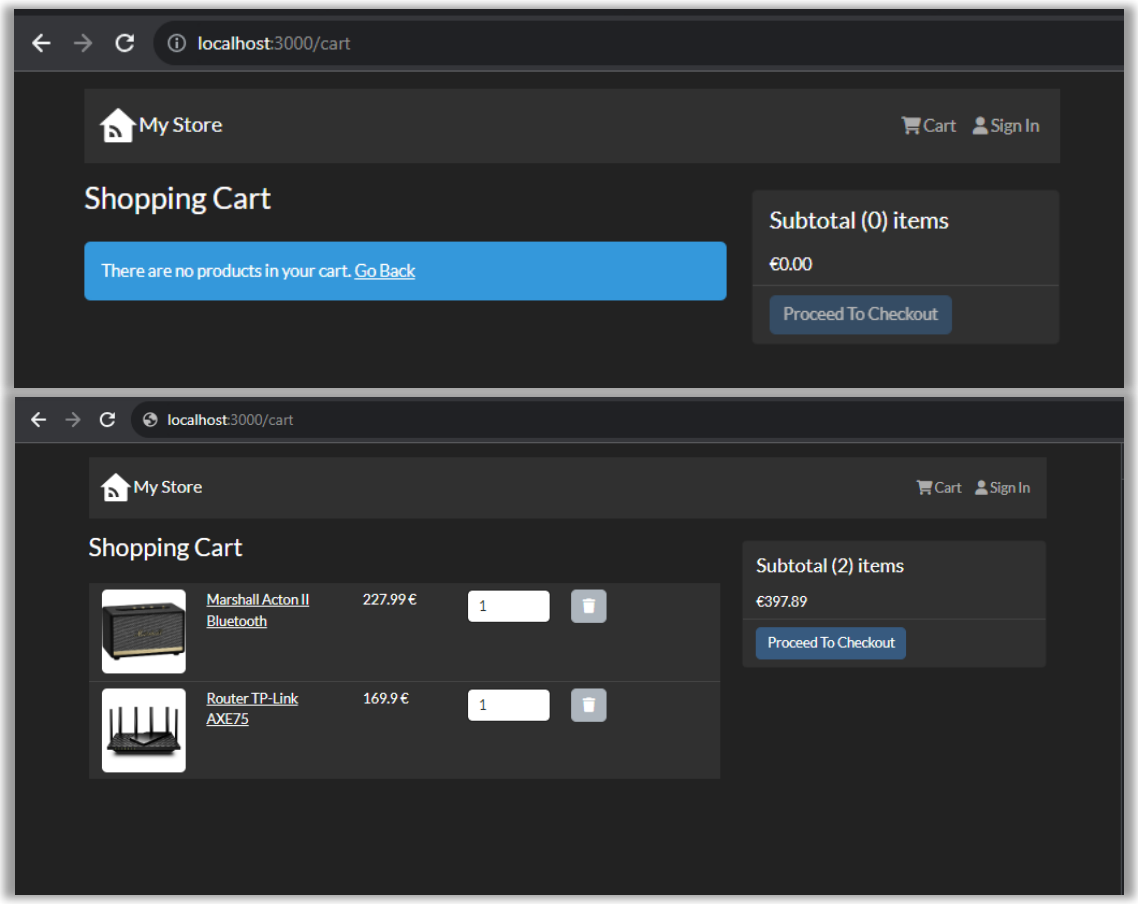

Figura 40 – Página do carrinho de compras

Relativamente ao processo de encomendas, segue a mesma lógica das funcionalidades descritas anteriormente, sendo que a única diferença é o facto de os utilizadores necessitarem autenticação para criar uma encomenda de modo a ser possível enviar o *json web token* nos pedidos efetuados ao servidor. Com base nas configurações definidas neste exemplo, as páginas inerentes a este processo incluem a barra de navegação da aplicação, algo que pode ser alterado diretamente no ficheiro *userChoices.json* da loja.

Após iniciar sessão e carregar no botão *Proceed to checkout,* o utilizador inicia o processo de criação de uma nova encomenda sendo redirecionado para a página referente aos detalhes de entrega. O utilizador deste exemplo é um administrador da aplicação, e como tal, é adicionado um separador no cabeçalho com as funcionalidades administrativas de gestão da base de dados que foram incluídas. A Figura 41 e Figura 42 representam este processo.

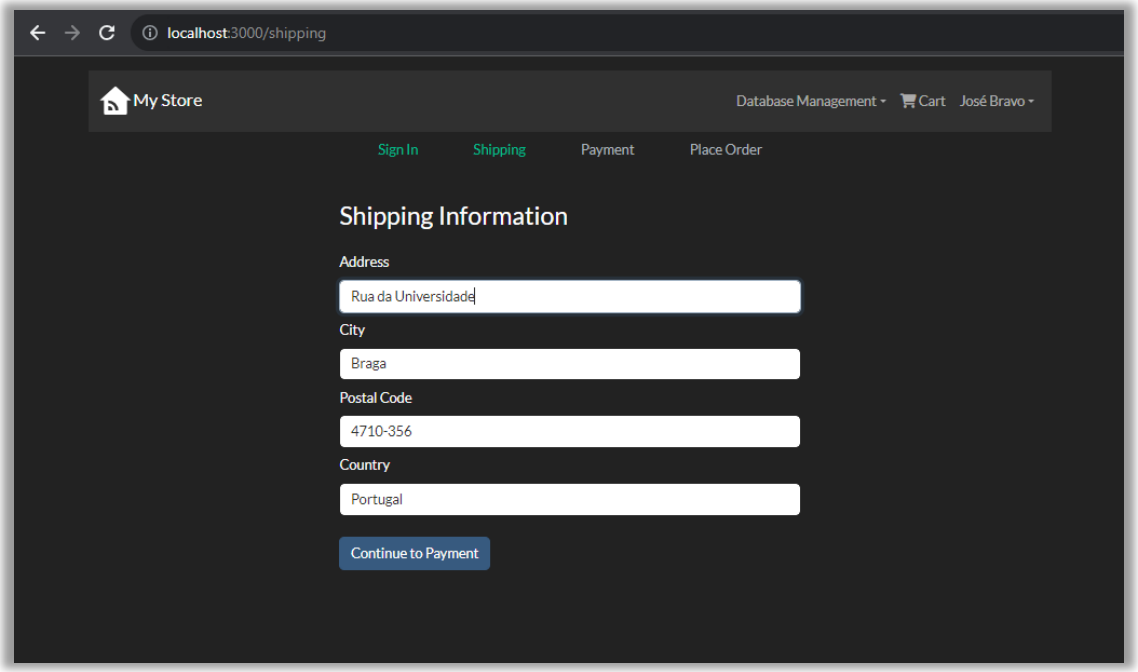

Figura 41 - Processo de encomenda - Detalhes de entrega

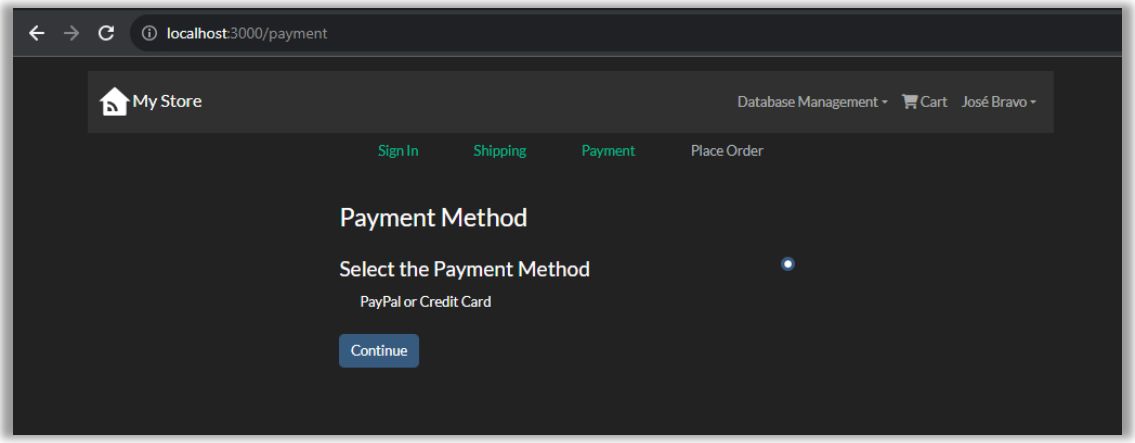

Figura 42 - Processo de encomenda - Método de Pagamento

Após preencher todos os detalhes da encomenda, o utilizador é redirecionado para o último passo deste processo, onde pode efetivamente criar o registo carregando no botão *Place Order,* sendo redirecionado para a página da encomenda criada, onde pode efetuar o pagamento, conforme ilustrado na Figura 43 e Figura 44.

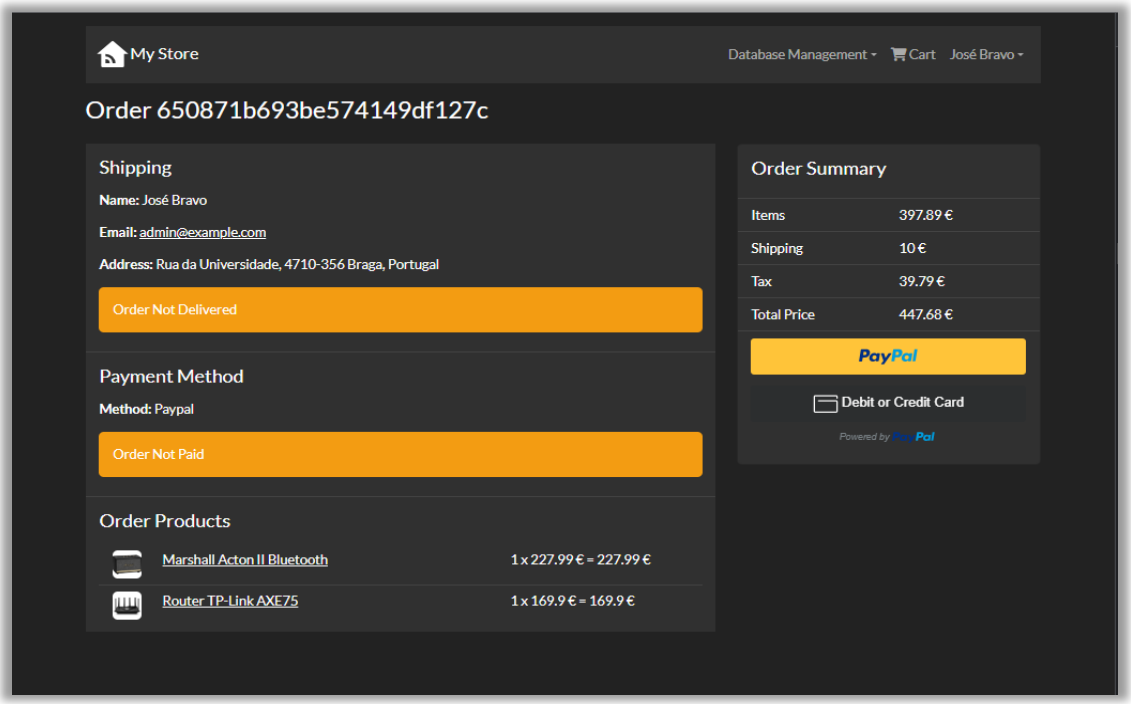

Figura 43 - Página de detalhes da encomenda – Pagamento Pendente

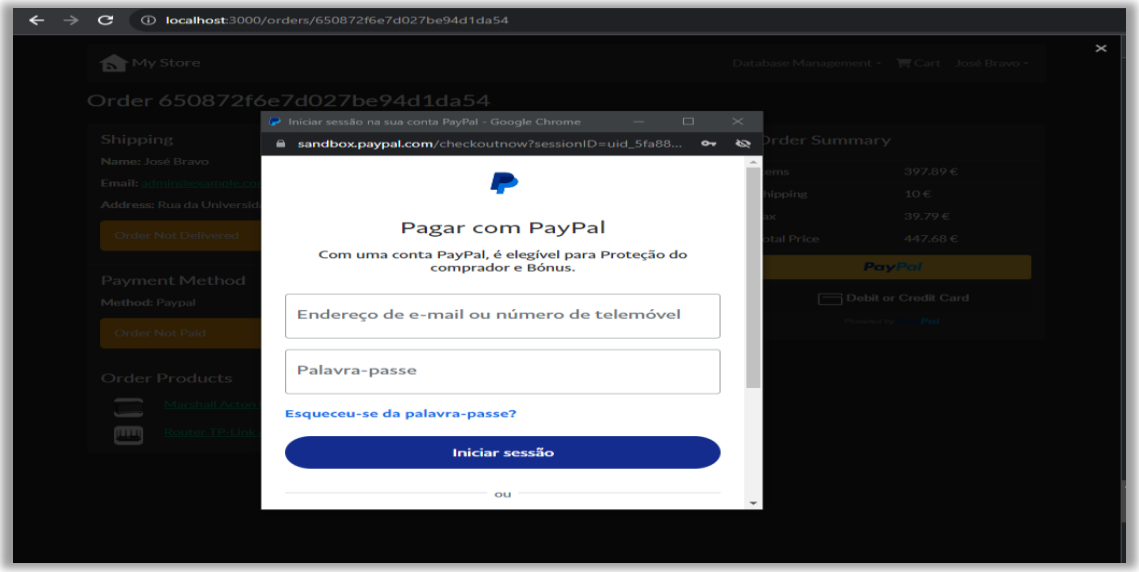

Figura 44 - Efetuar pagamento com *PayPal*

Após efetuar o pagamento, a página é atualizada de acordo sendo que, caso seja um administrador, é adicionado um botão que permite atualizar o estado de entrega. Do ponto de vista de um utilizador sem privilégios de administrador, a página é atualizada apenas com a data do pagamento, conforme ilustrado na Figura 45.

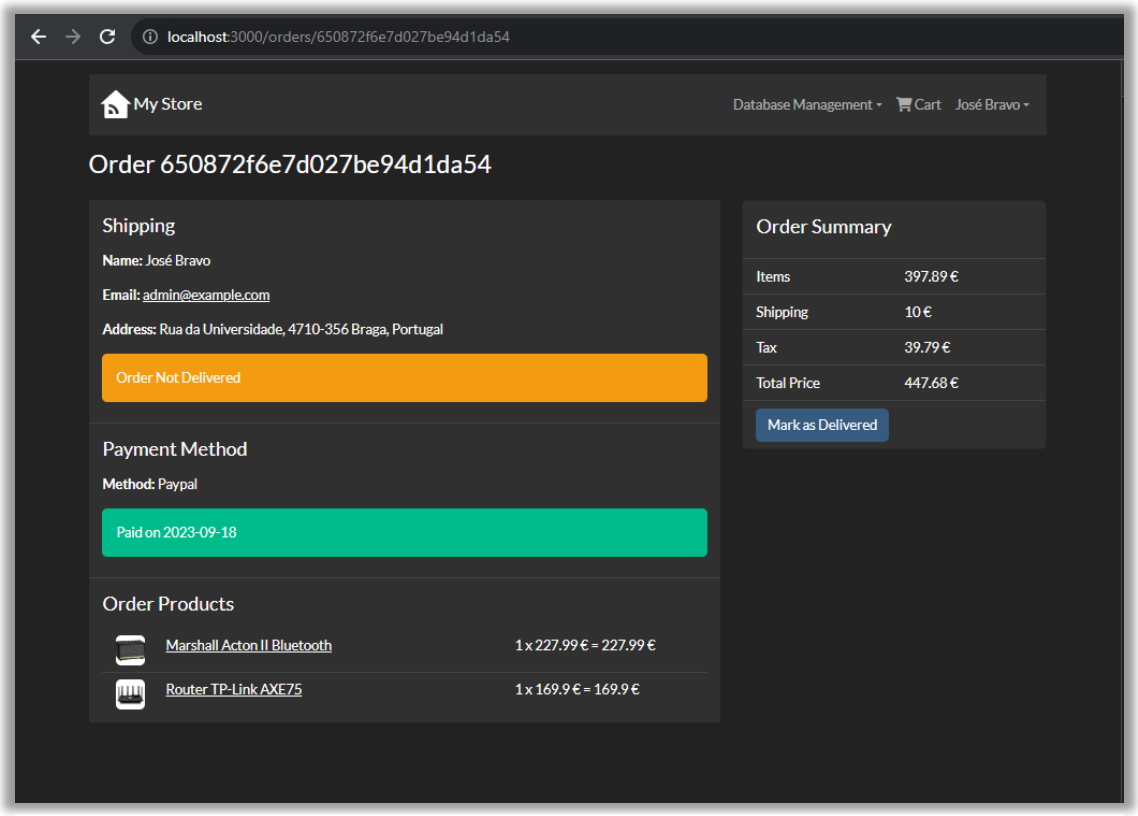

Figura 45 - Página de detalhes de encomenda - Pagamento Efetuado

Todos estes parâmetros inerentes a cada página incluída, como a adição da barra de navegação, as mensagens a apresentar, etc., podem ser atualizados diretamente no ficheiro *userChoices.json* da loja. Por outro lado, a inclusão de novas funcionalidades terá de ser efetuada na ferramenta de geração de lojas ou integradas através da modificação do código fonte gerado

6. Conclusão

O presente capítulo sintetiza o trabalho desenvolvido, permitindo uma reflexão crítica sobre os resultados alcançados, bem como uma projeção para possíveis trabalhos futuros.

A pesquisa relativa às plataformas existentes, pressupôs a análise e comparação das vantagens inerentes a cada uma. A revisão da literatura incidiu na análise de soluções propostas em alguns artigos científicos, contribuindo para uma visão mais abrangente sobre as alternativas inerentes ao processo de geração de lojas *online*.

O desenvolvimento da loja online permitiu compreender e analisar os requisitos de modo a generalizar e projetar os componentes customizáveis utilizados pela *framework* e a familiarização com as tecnologias *MERN*.

No desenvolvimento da plataforma de geração de lojas, definiram-se os parâmetros associados a cada componente customizável, disponibilizado pela *framework,* de forma a ser possível implementar e definir o funcionamento da ferramenta que permite a personalização e modificação dos *templates* de acordo com as preferências do utilizador, através da integração dos diferentes componentes.

A fase de validação permitiu demonstrar a versatilidade da utilização da plataforma através da geração de uma loja *online* com as funcionalidades foco e, opcionalmente, todas as outras funcionalidades descritas.

Desta forma, a estruturação desta dissertação não apenas explora os pormenores intrínsecos das encomendas e do carrinho de compras, mas também posiciona essas funcionalidades no contexto mais amplo da *framework* em desenvolvimento.

6.1 Discussão e análise de resultados

A plataforma de geração de lojas desenvolvida permite a criação rápida e intuitiva de lojas *online* completas ou parciais, dependendo das preferências do utilizador. A plataforma permite

gerar uma loja que inclua todas as funcionalidades descritas anteriormente ou simplesmente gerar a loja com determinadas funcionalidades.

Os modelos de dados referentes às coleções de utilizadores, produtos e encomendas também podem ser modificados através da adição de novos campos o que oferece uma maior versatilidade e capacidade de customização da base de dados. O aspeto e estilo da loja *online* a gerar também podem ser alterados consoante as preferências do utilizador, através da utilização de diferentes temas *bootstrap* disponibilizados pela plataforma, sendo que este ficheiro também pode ser alterado após descarregar o código-fonte da loja gerada. Além disso, é possível também incluir e customizar diferentes componentes *React (*como a barra de navegação, o menu de administrador, mensagens de erro etc.) utilizados nas diferentes páginas da loja.

Os componentes customizáveis que constituem a *framework* foram desenvolvidos de forma a serem o mais abstratos possível, de forma a facilitar a sua integração em futuras soluções, sendo que os componentes inerentes às funcionalidades do carrinho de compras, encomendas e pagamentos são o foco principal e como tal encontram-se devidamente documentados, incluindo os parâmetros e os passos necessários para a sua integração. Os restantes componentes foram desenvolvidos e/ou integrados seguindo a mesma metodologia de generalização e abstração.

A ferramenta de geração de lojas disponibiliza um formulário ao utilizador onde é possível configurar, personalizar e pré-visualizar a loja *online* a gerar. Estas configurações podem ser guardadas na base de dados caso o utilizador pretenda continuar a configuração mais tarde. Caso contrário, o utilizador pode efetuar o *download* do ficheiro *.zip* que contém o código-fonte da loja gerada. Este processo consiste na importação e manipulação dos *templates* fornecidos pela *framework*, consoante as escolhas do utilizador.

Após descarregar o ficheiro e executar as instruções de configuração e instalação das dependências, a loja será inicializada no *browser* com as funcionalidades definidas durante o processo de configuração. Os aspetos visuais como a inclusão da barra de navegação, os títulos das diferentes páginas, as mensagens a mostrar em caso de erro ou sucesso e a inclusão de botões pré-definidos podem ser alterados de forma simples através da edição do ficheiro

userChoices.json, após efetuar o *download*. Por outro lado, a inclusão/remoção de novas funcionalidades terá de ser efetuada no formulário, durante o processo de configuração ou através da edição manual do código-fonte da aplicação.

6.2 Sugestões de trabalho futuro

A solução desenvolvida responde aos objetivos propostos na sua totalidade, no entanto, existem alguns aspetos passíveis de melhoria e/ou otimização. A presente secção visa propor algumas sugestões de trabalho futuro destacando áreas de potencial crescimento e desenvolvimento.

A segurança é uma preocupação constante no ambiente digital, e a proteção dos dados dos utilizadores é uma prioridade. Nesse sentido, é recomendável avaliar a implementação de medidas adicionais de segurança, uma vez que não é o foco da presente dissertação. Este processo pode incluir a adoção de métodos de autenticação de dois fatores para contas de utilizadores, bem como a criptografia avançada de informações sensíveis armazenadas na plataforma de modo a mitigar potenciais vulnerabilidades.

De modo a garantir que os utilizadores possam tirar o máximo proveito da plataforma e superar eventuais obstáculos, a inclusão de um sistema de suporte técnico é uma consideração valiosa. O suporte pode abranger uma ampla variedade de recursos, como documentação detalhada, tutoriais em vídeo, um *chat* de suporte *online* em tempo real, etc. facilitando assim o processo de configuração de uma loja.

A plataforma atualmente suporta métodos de pagamento convencionais, como cartões de crédito e *PayPal*. No entanto, considerando a diversidade de preferências dos utilizadores e as variações regionais, a expansão para a inclusão de outros métodos de pagamento pode ser uma mais-valia, possibilitando a configuração de lojas *online* com base nas preferências e requisitos específicos do público-alvo de diferentes regiões e países.

Relativamente à base de dados da loja *online*, a plataforma permite que o utilizador forneça a chave de conexão de modo a conectar a loja aos serviços fornecidos pela *MongoDB.* Caso o utilizador não forneça esta chave de conexão, a base de dados da loja *online* gerada será

composta por uma base de dados pré-definida. Apesar do utilizador ter a possibilidade de acrescentar novos campos a cada uma das coleções, este processo poderia ser otimizado fornecendo ao utilizador uma interface que permita criar e configurar a base de dados e as respetivas coleções através da ferramenta de geração de lojas.

A otimização contínua do código-fonte da ferramenta de geração de lojas é um processo essencial para garantir que a plataforma funcione de maneira eficaz, com desempenho otimizado e facilidade de manutenção. Isso envolve a revisão do código existente de modo a mitigar redundâncias ou ineficiências, bem como a atualização das bibliotecas utilizadas. Manter um código limpo e otimizado contribui para um desempenho mais rápido, maior estabilidade e facilita futuras atualizações e integrações. A implementação de testes automatizados também seria uma adição valiosa à ferramenta, uma vez, que permitiria a identificação e tratamento de erros de uma forma simples, facilitando a integração de novas funcionalidades e componentes.

Além das sugestões anteriores, existem outras áreas de expansão que podem ser exploradas, como estratégias de *marketing* digital, análise de dados de utilizadores para personalização avançada de *templates* e funcionalidades adicionais como a integração de modelos de inteligência artificial que atendam a necessidades específicas do comércio eletrónico.

Neste sentido, pode-se concluir que os objetivos delineados foram atingidos com sucesso. A plataforma desenvolvida oferece uma solução robusta e funcional que permite a geração de lojas *online* de forma simples, rápida e intuitiva.

Referências

[1] Tecnologias MERN, “MERN Stack Explained” [Online]. Available: https://www.mongodb.com/mern-stack

[2] Shopify, “What Is Shopify and How Does It Work?” [Online]. Available: https://www.shopify.com/blog/what-is-shopify.

[3] Wix, “Content Management System (CMS)” [Online]. Available: https://www.wix.com/encyclopedia/definition/content-management-system-cms

[4] WordPress [Online]. Availabe: https://pt.wordpress.org/

[5] Magento [Online]. Availabe: https://www.treinaweb.com.br/blog/o-que-e-magento

[6] BigCommerce, [Online] Available: https://ecommerce-platforms.com/pt/compare/bigcommerce-vs-shopify

[7] Redux, [Online] Available: https://javascript.plainenglish.io/state-management-in-react-with-redux-toolkit-ff290d3134c3

[8] Institute of Electrical and Electronics Engineers Inc, “Analysis and Development of E-Commerce Web Application”, 2022.

[9] Bootstrap, [Online] Available: https://www.hostinger.com.br/tutoriais/o-que-e-bootstrap

[10] Django, [Online] Available: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Django/Introduction

[11] PayPal, [Online] Available: https://developer.paypal.com/api/rest/

[12] Association for Computing Machinery, “Design and Implementation of B2B E-commerce Platform Based on Microservices Architecture”, 2019

[13] Docker, [Online] Available: https://www.ibm.com/br-pt/topics/docker

[14] Kubernetes, [Online] Available: https://kubecampus.io/kubernetes/

[15] Spring Cloud e microservices, [Online] Available: https://dev.to/womakerscode/microsservicos-com-spring-cloud-introducao-3cn6

[16] IJRASET, “Microservices Enabled E-Commerce Web Application”, 2022

[17] NATS, [Online] Available: https://docs.nats.io/nats-concepts/what-is-nats

[18] Ilustração sobre MERN, [Online] Available: https://www.geeksforgeeks.org/mern-stack/

[19] Ilustração sobre Redux, [Online] Available: https://www.javatpoint.com/react-redux