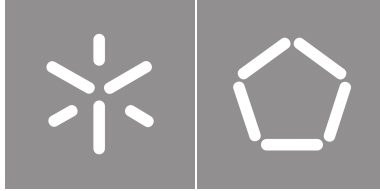


Universidade do Minho
Escola de Engenharia

Tiago Filipe Correia Lobão

Automatização de Processos Operacionais
em E-commerce: Uma Solução Web de Bac-
koffice



Universidade do Minho

Escola de Engenharia

Tiago Filipe Correia Lobão

**Automatização de Processos
Operacionais em E-commerce: Uma
Solução Web de Backoffice**

Dissertação de Mestrado

Mestrado Integrado em Engenharia de Telecomunicações e
Informática

Trabalho efetuado sob a orientação de:

José Manuel Tavares Vieira Cabral

Sérgio Adriano Fernandes Lopes

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositoriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho



Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International
CC BY-NC-SA 4.0

<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.en>

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.

Guimarães

(Local)

31 de outubro 2025

(Data)

(Tiago Filipe Correia Lobão)

Resumo

Automatização de Processos Operacionais em E-commerce: Uma Solução Web de Backoffice

O comércio eletrônico moderno exige operações rápidas, simples e fiáveis. Muitos negócios ainda enfrentam dificuldades por dependerem de processos manuais e ferramentas dispersas, o que conduz a erros, atrasos e perda de controlo operacional. Este projeto apresenta o desenvolvimento de uma plataforma *backoffice* concebida para centralizar a gestão diária de uma loja *online*, reunindo num único sistema o controlo de inventário, o processamento de encomendas, a faturação e a criação de envios.

A solução foi implementada com uma arquitetura *headless*, composta por uma [Application Programming Interface \(API\) Representational State Transfer \(REST\)](#) desenvolvida em *Node.js/Express* e uma base de dados *PostgreSQL* gerida através do *Prisma ORM*. O *frontend* é uma [Single Page Application \(SPA\)](#) em *React* com *Material UI*, assegurando uma experiência de utilização fluida. A autenticação é realizada por [JSON Web Token \(JWT\)](#), suportando diferentes perfis de acesso — *Super-Admin*, *Admin* e *Operador*. Entre as funcionalidades destacam-se a importação e atualização em massa de produtos via ficheiro [Comma-Separated Values \(CSV\)](#) e a integração direta com plataformas de *e-commerce*, como o *WooCommerce*.

O sistema é servido num ambiente *Droplet* com o *Nginx* a atuar como *reverse proxy* (com [Transport Layer Security \(TLS\)](#)) e o *pm2* a gerir o processo da aplicação *Node.js*. A avaliação técnica automatizada (*Google Lighthouse*) evidenciou bom desempenho e práticas sólidas de acessibilidade e segurança. Em conjunto, estes resultados validam a abordagem adotada, demonstrando que é possível combinar leveza tecnológica com robustez funcional, oferecendo às pequenas e médias empresas uma ferramenta fiável, acessível e preparada para evoluções futuras.

Palavras-chave: Operações de E-commerce, Plataforma de Backoffice, Gestão de Inventário, Processamento de Encomendas, CMS Headless Dedicado

Abstract

Automation of Operational Processes in E-commerce: A Web-Based Backoffice Solution

Modern e-commerce demands fast, simple, and reliable operations. Many businesses still face challenges due to manual processes and fragmented tools, leading to errors, delays, and limited operational control. This project presents the development of a *backoffice* platform designed to centralise the daily management of an *online* store, combining in a single system inventory control, order processing, invoicing, and shipment creation.

The solution was implemented using a *headless* architecture, composed of a [Representational State Transfer \(REST\) Application Programming Interface \(API\)](#) built with *Node.js/Express* and a *PostgreSQL* database managed through the *Prisma ORM*. The *frontend* is a [Single Page Application \(SPA\)](#) developed in *React* with *Material UI*, ensuring a smooth and responsive user experience. Authentication relies on [JSON Web Token \(JWT\)](#), supporting multiple access profiles — *Super-Admin*, *Admin*, and *Operator*. Key features include bulk product updates via [Comma-Separated Values \(CSV\)](#) files and direct integration with e-commerce platforms such as *WooCommerce*.

The system is hosted on a *Droplet* environment, with *Nginx* acting as a *reverse proxy* (secured with [Transport Layer Security \(TLS\)](#)) and *pm2* managing the *Node.js* process. Automated technical evaluation using *Google Lighthouse* demonstrated strong performance and solid accessibility and security practices. Overall, the results validate the adopted approach, showing that it is possible to combine lightweight technology with functional robustness, providing small and medium-sized enterprises with a reliable, accessible, and future-ready management tool.

Keywords: E-commerce Operations, Backoffice Platform, Inventory Management, Order Processing, Dedicated Headless CMS

Índice

Índice de Figuras	ix
Índice de Tabelas	xi
Índice de Listagens	xii
Lista de Abreviaturas	xiii
1 Introdução	1
1.1 Motivação	1
1.2 Objetivos	2
1.3 Estrutura do Documento	3
2 Estado da Arte	4
2.1 Tecnologias de Suporte à Gestão no E-commerce	4
2.1.1 Frontoffice no E-commerce: A Vitrine Digital	5
2.1.2 Backoffice no E-commerce: O Motor Invisível da Operação Digital	6
2.2 Trabalhos Relacionados	7
2.2.1 Backoffice para E-commerce: Plataformas Existentes	7
2.2.2 Integração via CSV e API	8
2.2.3 Automação de Envio e Etiquetagem	9
2.2.4 Gestão de Encomendas (Picking, Packing e Shipping)	11
2.2.5 Sincronização Multi-Canal	12
2.2.6 Casos de Estudo	14
2.3 Tecnologias e Arquiteturas	15
2.3.1 Linguagens e Frameworks	16
2.3.2 Arquiteturas Web	16
3 Análise e Conceção	21

3.1	Requisitos do Sistema	21
3.1.1	Requisitos Funcionais	21
3.1.2	Requisitos Não Funcionais	22
3.2	Visão Geral dos Casos de Utilização	23
3.3	Conceção	24
3.3.1	Arquitetura	24
3.3.2	Modelo de Dados	25
3.4	Arquitetura Final	26
3.4.1	Backend	27
3.4.2	Frontend	28
4	Implementação	30
4.1	Backend	30
4.1.1	Setup do Backend	30
4.1.2	Autenticação e Papéis	32
4.1.3	Base de Dados	34
4.1.4	Métodos Personalizados	38
4.2	Frontend	42
4.2.1	Setup do Frontend (React)	43
4.2.2	Frontend Sitemap	45
4.2.3	Estrutura Base da Interface	45
4.2.4	Componentes Reutilizáveis	46
4.2.5	Funções e Métodos	47
4.3	Deploy e Operação	49
4.3.1	Infraestrutura e Sistema Operativo	49
4.3.2	Topologia e Domínio	49
4.3.3	Backend em Produção	49
4.3.4	Frontend em Produção	49
4.3.5	Reverse Proxy e TLS	49
4.3.6	Base de Dados	50
4.3.7	Operação	50
4.3.8	Segurança	51
4.3.9	Notas e Melhorias Futuras	51
5	Análise e Testes	52
5.1	Análise de Funcionalidades	52
5.1.1	Login	52
5.1.2	Home	53
5.1.3	Encomendas	54

5.1.4	Produtos	54
5.1.5	Fornecedores	56
5.1.6	Receções (Inventário/Compras)	57
5.1.7	Clientes	58
5.1.8	Administração	59
5.1.9	Perfil	60
5.2	Análise Técnica Automatizada	62
5.3	Testes de Usabilidade (plano e critérios)	63
5.3.1	Amostra e Ambiente	63
5.3.2	Tarefas (Cenários)	63
5.3.3	Métricas a Recolher	64
5.3.4	Orçamentos de Tempo (Referência)	64
5.3.5	Critérios de decisão	64
6	Conclusão	65
6.1	Conquistas	65
6.2	Trabalho futuro	66
	Bibliografia	68

Índice de Figuras

1	Importância estratégica de um sistema de <i>backoffice</i> no e-commerce: otimização de processos, redução de erros e apoio à análise de dados.	6
2	Fluxo de automação do envio e etiquetagem em plataformas integradas com sistemas de e-commerce	10
3	<i>Workflow</i> automatizado do processo de <i>picking</i> e <i>packing</i> com validação por código de barras	11
4	Modelo de sincronização de inventário e preços com múltiplos canais de venda através de uma plataforma central	13
5	Página principal da loja <i>online</i> da Zippo	14
6	Página principal da loja <i>online</i> da 3DXTECH	15
7	Diagrama de uma arquitetura monolítica	16
8	Diagrama de uma arquitetura em camadas	17
9	Diagrama de uma arquitetura <i>space-based</i>	17
10	Diagrama de uma arquitetura de microserviços	18
11	Diagrama de uma arquitetura orientada a serviços (Service-Oriented Architecture (SOA)) .	18
12	Diagrama de uma arquitetura orientada a eventos (<i>Event-Driven</i>)	19
13	Diagrama de uma arquitetura <i>headless</i>	20
14	Esquema da arquitetura	25
15	Diagrama entidade-relacionamento simplificado (chaves primárias e estrangeiras). . . .	26
16	Arquitetura geral do sistema e integração entre componentes.	27
17	Fluxo de execução dos <i>middlewares</i> globais e de segurança no servidor <i>Express</i>	31
18	Estrutura de pastas do <i>backend</i>	32
19	Coleção de tabelas — Parte 1 (Utilizadores, canais de venda, clientes e encomendas). . .	35
20	Coleção de tabelas — Parte 2 (Aprovisionamento, faturas, fornecedores e movimentos de <i>stock</i>).	36
21	Relação entre controladores e entidades da base de dados.	38
22	Estrutura de pastas do <i>frontend</i>	43

23	Mapa de rotas do <i>frontend</i> e distribuição de acessos.	44
24	Arquitetura funcional do <i>frontend</i> com fluxo de componentes, rotas e elementos reutilizáveis.	44
25	Esquema simplificado do componente <i>Layout</i> e da sua integração com a <i>Navbar</i> e as páginas autenticadas.	46
26	Ecrã de <i>login</i>	53
27	<i>Home</i> do <i>backoffice</i> com cartões de acesso: <i>Orders, Products, Clients, Receptions, Suppliers</i> e <i>Administration</i>	53
28	Lista de encomendas com pesquisa e filtros.	54
29	Detalhe da encomenda: informação consolidada e ações (fatura, expedição, estado).	55
30	Lista de produtos com pesquisa e filtros.	55
31	Detalhe do produto com edição dos campos principais.	56
32	Tabela de fornecedores.	56
33	<i>Popup</i> de edição de fornecedor.	57
34	Lista de receções (faturas de fornecedor) com pesquisa e filtros.	57
35	Criação/edição de receção: seleção de fornecedor e linhas de artigos.	58
36	Lista de clientes com pesquisa.	58
37	Detalhe do cliente com contactos e moradas	59
38	<i>Admin</i> — gestão de utilizadores (tabela e <i>popup</i> de edição).	60
39	<i>Admin</i> — gestão de sites.	61
40	Importação CSV por <i>site</i> : atualização em massa de preços e quantidades.	61
41	Página de Perfil — consulta e edição de dados básicos do utilizador.	62

Índice de Tabelas

1	Comparação entre integração via CSV e via API	9
2	Comparação entre soluções de envio e etiquetagem	10
3	Comparação de plataformas de envio	11
4	Comparação de funcionalidades em soluções de gestão de encomendas	12
5	<i>Endpoints</i> aplicativos adicionais para além do Create, Read, Update, Delete (CRUD). .	39
6	Resultados <i>Lighthouse</i> (<i>desktop</i>) — relatório de referência da <i>Home</i>	62

Índice de Listagens

4.1	Configuração de <i>middlewares</i> globais do servidor	31
4.2	Trechos essenciais do <i>schema.prisma</i> com chaves compostas e unicidades	37
4.3	<i>Endpoint</i> de consulta de encomenda com dados operacionais	42
4.4	Interceptador global do Axios com inclusão do JWT	48
4.5	Bloco Nginx de produção (<i>frontend</i> estático e proxy da API)	49

Lista de Abreviaturas

API	Application Programming Interface (pp. iv , v , xi , 5 , 7–10 , 13 , 16 , 18 , 19 , 24 , 27 , 39 , 40 , 42 , 44 , 45 , 47–49 , 67)
B2B	Business to Business (p. 15)
B2C	Business to Consumer (p. 15)
BLOB	Binary Large Object (pp. 41 , 48)
CI/CD	Continuous Integration/Continuous Deployment (p. 51)
CLI	Command Line Interface (p. 30)
CMS	Content Management System (pp. 24 , 27 , 32 , 39)
CORS	Cross-Origin Resource Sharing (pp. 31 , 41 , 49 , 66)
CRM	Customer Relationship Management (p. 8)
CRUD	Create, Read, Update, Delete (pp. xi , 30 , 38 , 39)
CSRF	Cross-Site Request Forgery (p. 33)
CSV	Comma-Separated Values (pp. iv , v , x , xi , 1 , 2 , 8 , 9 , 13 , 22–24 , 33 , 34 , 39–41 , 54 , 59–61 , 63 , 66)
DNS	Domain Name System (p. 49)
DX	Developer Experience (p. 67)
EAN	European Article Number (p. 35)
ER	Entity-Relationship (pp. 25 , 34)
ERP	Enterprise Resource Planning (pp. 6 , 8 , 14 , 15)
HSTS	HTTP Strict Transport Security (pp. 41 , 49 , 51)
HTTP	Hypertext Transfer Protocol (pp. 28 , 31 , 38 , 39 , 49)
HTTPS	Hypertext Transfer Protocol Secure (pp. 49 , 62)

ID	Identifier (<i>p. 8</i>)
IT	Information Technology (<i>p. 14</i>)
JSON	JavaScript Object Notation (<i>pp. 8, 13, 31</i>)
JWT	JSON Web Token (<i>pp. iv, v, xii, 31–33, 38, 42, 44, 45, 48, 52, 65, 66</i>)
NPM	Node Package Manager (<i>p. 30</i>)
ORM	Object-Relational Mapping (<i>p. 27</i>)
POS	Point of Sale (<i>p. 9</i>)
PWA	Progressive Web App (<i>p. 15</i>)
RBAC	Role-Based Access Control (<i>pp. 28, 32, 33</i>)
REST	Representational State Transfer (<i>pp. iv, v, 13</i>)
REST API	Representational State Transfer Application Programming Interface (<i>pp. 24, 28, 32, 52, 65, 66</i>)
SAFT	Standard Audit File for Tax (<i>p. 67</i>)
SEO	Search Engine Optimization (<i>p. 5</i>)
SKU	Stock Keeping Unit (<i>pp. 8, 35, 54, 57, 60, 66</i>)
SOA	Service-Oriented Architecture (<i>pp. ix, 18</i>)
SPA	Single Page Application (<i>pp. iv, v, 5, 28, 42, 43, 45, 46, 49, 52, 62, 63, 65, 66</i>)
SSG	Static Site Generation (<i>p. 5</i>)
SSH	Secure Shell Protocol (<i>pp. 50, 51</i>)
SSR	Server-Side Rendering (<i>p. 5</i>)
TLS	Transport Layer Security (<i>pp. iv, v, 49, 65</i>)
UI	User Interface (<i>pp. 7, 45–48, 63, 67</i>)
URL	Uniform Resource Locator (<i>pp. 34, 38, 39, 44, 45, 47</i>)
UX	User Experience (<i>p. 7</i>)
WCAG	Web Content Accessibility Guidelines (<i>p. 62</i>)
WMS	Warehouse Management System (<i>p. 12</i>)
XML	eXtensible Markup Language (<i>pp. 8, 13</i>)

XSS Cross-Site Scripting (p. [31](#))

Introdução

Nos últimos anos, o comércio eletrónico deixou de ser uma alternativa e passou a ser, para muitos, o canal principal de venda. Com o crescimento das lojas *online* e a evolução constante das plataformas digitais, o comportamento dos consumidores tem acompanhado este ritmo acelerado. No meio desta transformação, há uma questão que continua a preocupar quem vende: como gerir, de forma prática e eficaz, tudo aquilo que acontece nos bastidores, desde o controlo do *stock* até ao momento em que a encomenda sai pela porta, pronta a ser entregue.

Apesar da tecnologia disponível, a verdade é que muitos vendedores continuam a depender de processos lentos, desorganizados ou, em alguns casos, quase artesanais. A gestão de encomendas e produtos, quando não é centralizada ou automatizada, pode rapidamente tornar-se um verdadeiro pesadelo. Erros na faturação, falhas de *stock*, etiquetas mal geradas, tudo isto não só compromete a experiência do cliente, como desgasta quem está por trás do processo.

É precisamente para responder a estes desafios que surge a proposta desta dissertação: a criação de uma plataforma de *backoffice* que facilite, de forma intuitiva e eficiente, a gestão operacional de lojas de *e-commerce*. A ideia é simples, mas poderosa: permitir dar entrada de *stock*, processar e finalizar encomendas, gerar faturas, criar etiquetas de envio e, ainda, gerir preços e quantidades de produtos em diferentes sites através de um ficheiro [Comma-Separated Values \(CSV\)](#). Tudo isto num único espaço, pensado para quem quer ter controlo total sem perder tempo com tarefas repetitivas e demoradas.

Esta proposta é, assim, dirigida a todos os que gerem lojas de *e-commerce*, desde pequenos empreendedores a negócios em expansão, que procuram uma solução prática, escalável e desenhada com base nas dificuldades reais que enfrentam no dia a dia.

1.1 Motivação

Vender *online*, à primeira vista, pode parecer simples: coloca-se o produto numa plataforma, define-se um preço, e espera-se que os pedidos comecem a entrar. No entanto, para quem está diariamente a gerir uma loja de *e-commerce*, a realidade é bem mais complexa. Por trás de cada venda existe um conjunto de processos, desde a atualização do *stock* até à emissão da fatura, que exigem tempo, atenção

ao detalhe e ferramentas adequadas. É precisamente na gestão dessas operações que muitos negócios enfrentam dificuldades significativas.

Apesar da evolução das plataformas digitais, grande parte das pequenas e médias lojas *online* continua a recorrer a sistemas dispersos, folhas de cálculo, ou até mesmo processos manuais para gerir as encomendas e os produtos. Esta abordagem não só consome muito tempo como também aumenta o risco de erros: preços desatualizados, ruturas de *stock* não detetadas, encomendas por processar ou etiquetas de envio mal geradas são apenas alguns dos problemas mais comuns.

Estes entraves não afetam apenas a eficiência interna, mas refletem-se também na experiência do cliente. Um estudo da *Shopify* [2] destaca que falhas logísticas e atrasos na entrega estão entre os principais motivos de insatisfação entre consumidores. Outros dados, como os apresentados pela *Metrobi* [3], reforçam esta ideia ao demonstrar que a gestão logística é um dos fatores mais determinantes na perceção do serviço por parte do cliente final. Já segundo a *HulkApps* [4], a ineficiência nos processos de expedição e preparação de encomendas pode ter impactos diretos na retenção de clientes e na reputação da marca.

Neste contexto, torna-se evidente a necessidade de uma solução que centralize e simplifique os processos operacionais de uma loja de *e-commerce*. A presente dissertação parte da necessidade prática e real: oferecer uma plataforma que reduza o peso da gestão diária e permita uma operação mais ágil, controlada e escalável. Ao automatizar tarefas rotineiras e permitir maior visibilidade sobre o negócio, espera-se não só melhorar a eficiência do sistema, mas também libertar tempo e energia para aquilo que realmente importa, fazer crescer o negócio.

1.2 Objetivos

As operações de uma loja de *e-commerce* exigem uma articulação eficiente entre a gestão de *stock*, o processamento de encomendas, a faturação e a expedição dos produtos. A presente dissertação tem como objetivo conceber e implementar uma arquitetura de sistema que facilite essa coordenação interna, permitindo aos responsáveis de loja gerir de forma centralizada e eficiente os fluxos de trabalho.

A plataforma proposta visa automatizar e integrar as etapas mais críticas do *backoffice*, contribuindo para a redução de erros, maior controlo operacional e melhor experiência do cliente. Entre as principais funcionalidades a desenvolver, destacam-se:

- Painel de gestão de produtos;
- Entrada de *stock* e gestão de inventário;
- Processamento e finalização de encomendas com emissão de faturas;
- Geração de etiquetas de envio personalizadas;
- Importação e atualização de preços e quantidades via ficheiro [CSV](#).

Com este sistema, pretende-se criar uma solução prática, escalável e adaptável à realidade das lojas *online*, oferecendo aos utilizadores maior autonomia e controlo sobre os seus processos logísticos e comerciais.

1.3 Estrutura do Documento

Este documento está organizado em seis capítulos distintos: Introdução, Estado da Arte, Análise e Conceção, Implementação, Testes e Análise, e Conclusão. Cada um deles aborda aspetos específicos da investigação e desenvolvimento da plataforma proposta.

O capítulo Estado da Arte apresenta uma revisão da literatura existente, explorando conceitos chave, tecnologias relevantes e trabalhos relacionados com tema da dissertação. Segue-se o capítulo Análise e Conceção, onde é estabelecida a base teórica da solução proposta. Aqui são apresentados os esquemas arquitetónicos, as tecnologias selecionadas para a implementação e os modelos de dados a utilizar.

No capítulo de Implementação, descreve-se a concretização prática da solução. São discutidas as decisões tomadas ao longo do desenvolvimento e explicada, de forma pormenorizada, a forma como foram aplicadas, com base nos fundamentos do capítulo anterior.

O capítulo de Testes e Análise centra-se na avaliação do sistema desenvolvido. Esta avaliação é realizada tanto a nível funcional, como a nível técnico, utilizando ferramentas de medição de desempenho para aferir a eficácia do sistema.

Por fim, o capítulo de Conclusão resume os resultados alcançados ao longo do trabalho, refletindo sobre os objetivos inicialmente definidos e sobre o contributo global desta investigação.

Estado da Arte

A rápida evolução tecnológica tem levado o setor de *e-commerce* a adotar, de forma crescente, soluções inovadoras para otimizar as operações internas e melhorar a experiência do cliente. Este capítulo analisa e revê as tecnologias emergentes e consolidadas que têm moldado a gestão de lojas *online*. Desde sistemas automatizados de gestão de encomendas a soluções avançadas de controlo de inventário, este estudo destaca os métodos e ferramentas mais eficazes para aumentar a eficiência operacional, melhorar a precisão na execução das tarefas e potenciar os resultados de negócios digitais, mesmo em estruturas com recursos limitados [5, 6].

Muitas lojas de *e-commerce* começaram a abandonar métodos manuais ou dispersos para adotar plataformas integradas que permitem a automatização de tarefas essenciais, como atualização de preços, sincronização de *stock* e emissão de faturas. Com o objetivo de reduzir custos operacionais e proporcionar uma experiência de compra mais fluida, tem-se assistido a uma adoção crescente de sistemas digitais que permitem aos gestores centralizar e controlar as operações com maior rapidez e eficácia [7, 8].

2.1 Tecnologias de Suporte à Gestão no E-commerce

O crescimento acelerado do comércio eletrónico nas últimas décadas está intrinsecamente ligado ao desenvolvimento das tecnologias digitais que o suportam [9]. À medida que os consumidores se tornaram mais exigentes e os mercados mais competitivos, tornou-se essencial para os negócios não só apresentarem produtos de forma atrativa, mas também assegurarem operações internas robustas, escaláveis e eficientes [10].

Neste contexto, os sistemas de gestão em *e-commerce* evoluíram de soluções rudimentares para plataformas altamente especializadas, capazes de integrar funções críticas como o controlo de *stock*, gestão de encomendas, faturação, relatórios em tempo real e integração com canais de venda externos. A complexidade crescente das operações e a pressão por tempos de resposta mais rápidos tornaram a separação entre o site de vendas (*frontoffice*) e o sistema de gestão (*backoffice*) uma necessidade técnica e estratégica [11].

Esta secção explora, assim, a estrutura técnica do *e-commerce* moderno, distinguindo as funções e tecnologias aplicadas no *frontoffice* e no *backoffice*, evidenciando as vantagens de manter essas duas

camadas de forma independente [12].

2.1.1 Frontoffice no E-commerce: A Vitrine Digital

O *frontoffice* de uma loja *online* representa a camada visual e funcional com a qual o utilizador interage diretamente. Este representa o ponto de contacto entre o cliente e o negócio digital, a vitrine onde produtos são apresentados, descrições consultadas, filtros aplicados e onde o processo de compra acontece. Esta interface deve ser atrativa, rápida, responsiva e intuitiva, já que qualquer atrito ou atraso pode resultar na perda imediata de uma venda [13].

2.1.1.1 Tecnologias e Abordagens Modernas

Nos últimos anos, ferramentas de desenvolvimento como **React.js**, **Vue.js** e **Next.js** tornaram-se fundamentais na criação de interfaces ricas e dinâmicas. Estas tecnologias permitem desenvolver [Single Page Application \(SPA\)](#) que otimizam o carregamento de conteúdos, proporcionando uma navegação fluida e contínua [14]. O **Next.js**, em particular, oferece renderização do lado do servidor ([Server-Side Rendering \(SSR\)](#)) e geração de sites estáticos ([Static Site Generation \(SSG\)](#)), características que melhoram o desempenho e a indexação em motores de busca ([Search Engine Optimization \(SEO\)](#)) [15].

Além da parte técnica, práticas como *design* centrado no utilizador, carregamento progressivo de conteúdos e otimização para dispositivos móveis são essenciais para garantir compatibilidade com múltiplos dispositivos e condições de rede diversas. Tais cuidados têm impacto direto nas métricas de conversão e satisfação do utilizador [13].

2.1.1.2 Arquitetura Headless e Benefícios Operacionais

A separação entre o *frontend* e o *backend*, através de uma arquitetura *headless*, veio transformar a forma como os sistemas de *e-commerce* são pensados. Com este modelo, o *frontend* comunica com o *backend* exclusivamente através de [Application Programming Interface \(API\)](#), o que permite um desenvolvimento modular, flexível e adaptável a diferentes canais (loja *online*, redes sociais, *marketplaces*, etc.) [16].

Esta independência entre camadas significa que atualizações ou alterações no *frontoffice* não afetam o funcionamento interno da loja e vice-versa. Equipas distintas podem trabalhar em paralelo, aumentando a eficiência e reduzindo o tempo de lançamento de novas funcionalidades, facilitando testes, o que reduz o risco de falhas em produção. Por outro lado, a arquitetura *headless* melhora o desempenho geral e a escalabilidade, visto que cada componente pode ser otimizado de forma independente [17].

2.1.1.3 Impacto Estratégico

A adoção de uma abordagem centrada na experiência do utilizador, aliada a uma infraestrutura técnica moderna, conseguem melhorar substancialmente os seus indicadores de sucesso, como taxas de conversão, retenção de clientes e valor médio por encomenda. O *frontoffice*, sendo o primeiro ponto

de contacto digital com a marca, é um elemento central na construção da credibilidade, usabilidade e competitividade de qualquer loja *online*.

2.1.2 Backoffice no E-commerce: O Motor Invisível da Operação Digital

O *backoffice* de uma loja *online* é o centro nevrálgico que garante que tudo o que acontece no *frontoffice* funcione corretamente. É responsável por processar encomendas, gerir *stock*, emitir faturas, comunicar com as transportadoras e manter a sincronização entre os vários canais de venda. Sem esta estrutura, a operação digital colapsaria, por mais bonito e funcional que fosse o site, como se pode observar na Figura 1 [18].



Figura 1: Importância estratégica de um sistema de *backoffice* no e-commerce: otimização de processos, redução de erros e apoio à análise de dados.

2.1.2.1 Integração com Sistemas ERP e Gestão Modular

A integração com sistemas de [Enterprise Resource Planning \(ERP\)](#) é uma das grandes evoluções no *backoffice* de *e-commerce*. Plataformas como o SAP, Odoo ou NetSuite permitem gerir de forma centralizada e automatizada operações como inventário, faturação, clientes e fornecedores. A ligação direta entre [ERP](#) e a loja *online* reduz erros manuais, melhora o controlo financeiro e permite uma tomada de decisões baseada em dados em tempo real [19].

A integração eficaz permite:

- **Controlo de inventário em tempo real**, evitando ruturas ou excesso de *stock*;
- **Automatização do processamento de encomendas**, com redução de erro humano;
- **Sincronização multicanal**, mantendo os dados consistentes entre a loja *online*, marketplaces e pontos de venda físicos;
- **Consolidação de dados financeiros**, simplificando obrigações fiscais e relatórios estratégicos [20].

2.1.2.2 Arquitetura Headless: Separação Estratégica

A separação entre o *frontend* e o *backend* através de uma arquitetura *headless* oferece uma flexibilidade fundamental para negócios em crescimento. Esta abordagem permite que ambas as camadas sejam escaladas, atualizadas e adaptadas de forma independente, sem comprometer a estabilidade do sistema [11].

Segundo especialistas, esta separação possibilita:

- Desenvolvimento paralelo entre equipas de [User Interface \(UI\)/User Experience \(UX\)](#) e equipas de infraestrutura;
- Maior agilidade na implementação de novas funcionalidades;
- Integrações mais fáceis com serviços externos através de [API](#) ou GraphQL [21].

2.1.2.3 Automação para Eficiência e Escalabilidade

A utilização de sistemas automatizados no *backoffice* permite reduzir significativamente a carga operacional e aumentar a precisão dos processos. *Softwares* de gestão de inventário e faturação, por exemplo, tornam as operações mais ágeis e menos propensas a falhas, o que permite à equipa focar-se na expansão do negócio e não na resolução de problemas diários [22].

Além disso, essa automação:

- **Melhora a satisfação do cliente** com entregas mais rápidas e menos erros;
- **Reduz custos operacionais** ao eliminar tarefas manuais repetitivas;
- **Facilita adaptações rápidas** a alterações de mercado ou comportamentos de consumo.

2.2 Trabalhos Relacionados

Com o crescimento do *e-commerce* e a constante evolução tecnológica, surgiram diversas soluções digitais para otimizar a gestão operacional de lojas *online*. Esta secção explora algumas das abordagens e ferramentas já implementadas no setor, com foco especial em plataformas de *backoffice* e na forma como estas contribuem para a eficiência, automação e escalabilidade dos negócios digitais. Através da análise de casos reais e soluções amplamente utilizadas, pretende-se contextualizar e justificar a proposta desenvolvida nesta dissertação.

2.2.1 Backoffice para E-commerce: Plataformas Existentes

O *backoffice* é o motor invisível das operações de uma loja *online*. É neste sistema que se processam tarefas críticas como entrada de *stock*, faturação, criação de etiquetas de envio e gestão de encomendas. A centralização e automação destas funções são essenciais para manter a eficiência num mercado cada vez mais competitivo.

Separar o *backoffice* do *frontoffice* (a loja visível para o cliente) traz múltiplas vantagens: permite modularidade técnica, facilita atualizações sem impacto direto na experiência do utilizador e melhora a segurança dos dados sensíveis [23]. Além disso, esta separação permite escalar operações sem comprometer a estabilidade do site de vendas [24].

Plataformas como **WooCommerce** oferecem extensões para transformar uma simples loja em sistemas de gestão completos. Um exemplo é o *plugin* **Shelf Planner**, que integra previsão de vendas e gestão automática de inventário [25].

Soluções externas como o **Zoho Inventory** disponibilizam funcionalidades avançadas, gestão de múltiplos armazéns, sincronização com vários canais de venda e geração automática de etiquetas [26]. Já o **Fishbowl Inventory** destaca-se pela integração com ERP como SAP e QuickBooks, sendo ideal para operações com cadeias de produção ou logística complexas [27]. A necessidade de integrar o inventário com sistemas de contabilidade é também sublinhada por soluções ERP como o **BlueLink ERP** [22].

Por fim, o caso da loja **Cabin Street** mostra os benefícios práticos da integração: com o *WooCommerce* e o *WooPayments*, eliminaram pausas nas vendas *online* provocadas por operações de conciliação financeira, garantindo *stock* sincronizado entre loja física e digital [28].

2.2.2 Integração via CSV e API

A integração de ficheiros *Comma-Separated Values* (CSV) e o uso de API são estratégias amplamente adotadas para sincronizar preços e *stock* entre diferentes plataformas e sistemas de *backoffice*. Essas práticas oferecem flexibilidade e automação eficientes, especialmente para lojas que operam em múltiplos canais.

No caso da integração via CSV, o *WooCommerce* disponibiliza ferramentas nativas de importação e exportação de ficheiros, permitindo atualizar atributos, preços e *stock* em massa — uma funcionalidade bastante útil para lojas com grandes volumes de produtos [29]. Para necessidades mais avançadas, a extensão premium *Product CSV Import Suite* suporta o carregamento de grandes volumes de dados, incluindo variações e campos personalizados, além de permitir o agendamento de tarefas automáticas [30]. Outra alternativa focada exclusivamente em preços e *stock* é o *Price and Stock Import*, que possibilita a pré-visualização das alterações antes de aplicá-las e oferece gestão por Identifier (ID) ou Stock Keeping Unit (SKU) [31]. Estas soluções contribuem para a redução de erros manuais e promovem consistência operacional entre várias lojas online.

Quanto à integração via API, destaca-se o *plugin* *WooCommerce Product Sync* da miniOrange, que facilita a sincronização automática de produtos através de API em formato JavaScript Object Notation (JSON) ou eXtensible Markup Language (XML). Esta solução permite importar imagens, categorias, atributos e níveis de *stock*, além de oferecer suporte para operações programadas e integração com sistemas externos como Customer Relationship Management (CRM), fornecedores, *dropshippers* e plataformas ERP [32]. Outra opção relevante é o sistema *Stok.ly*, que funciona como um centro de gestão centralizada no modelo “hub-and-spoke”, permitindo que alterações feitas no *backoffice* (inclusive via CSV) sejam refletidas automaticamente em todos os canais de venda — incluindo lojas de *e-commerce*, pontos de venda

físicos (*Point of Sale (POS)*) e *marketplaces* — tudo em tempo real [33]. A Tabela 1 sintetiza de forma comparativa as principais diferenças entre a integração via *CSV* e via *API*, em termos de automação, complexidade técnica, frequência de atualização, flexibilidade e risco de erro.

Tabela 1: Comparação entre integração via *CSV* e via *API*

Critério	Integração via CSV	Integração via API
Automação	Requer ação manual ou agendada	Totalmente automatizada em tempo real
Complexidade técnica	Fácil de implementar e gerir	Requer configuração técnica inicial
Frequência de atualização	Periódica, definida pelo utilizador	Contínua, sempre que há alterações
Flexibilidade	Menos flexível para múltiplos canais	Alta flexibilidade e integração com múltiplos sistemas
Risco de erro	Elevado se o ficheiro estiver mal formatado	Reduzido, depende da estabilidade da <i>API</i>

A escolha entre estas abordagens depende do grau de automação, volume de atualizações e complexidade do ecossistema tecnológico da loja.

2.2.3 Automação de Envio e Etiquetagem

A automação dos processos de envio e etiquetagem é crucial para lojas de *e-commerce* com volumes de encomendas medianos a elevados. Além de reduzir o trabalho manual, diminui erros e agiliza prazos, melhorando a experiência do cliente, conforme ilustrado no fluxo apresentado na Figura 2.

Entre as soluções mais utilizadas, destaca-se o *ShipStation*, integrado ao *WooCommerce* através de um plugin dedicado. Esta ferramenta sincroniza automaticamente encomendas, permite a geração de etiquetas em lote, aplica regras logísticas personalizadas e atualiza o estado dos envios com códigos de *tracking* diretamente na plataforma [34].

Outra opção relevante é o *WooCommerce Shipping*, uma solução nativa da plataforma que possibilita a criação imediata de etiquetas para transportadoras como *UPS*, *USPS* e *DHL*. Esta integração facilita o processo de envio ao oferecer descontos em tarifas e impressão direta a partir do painel de administração [35].

O *Sendcloud* é uma alternativa robusta para quem procura maior flexibilidade, permitindo a ligação com mais de 100 transportadoras. Entre as suas funcionalidades incluem-se a criação em massa de etiquetas, automação baseada em regras predefinidas, um portal dedicado ao *tracking* e ainda a gestão simplificada de devoluções [36]. A Tabela 2 apresenta uma comparação entre processos manuais e soluções automatizadas de envio e etiquetagem, evidenciando o impacto da automação em termos de tempo, erros e integração com transportadoras.

Por fim, soluções como *Easyship*, *Packlink PRO* e *Shippo* são especialmente indicadas para envios internacionais. Estas plataformas oferecem cotações em tempo real, integração direta com *WooCommerce*, seguros opcionais e funcionalidades avançadas de *tracking* [37, 38]. As diferenças funcionais entre as principais plataformas especializadas encontram-se sintetizadas na Tabela 3, permitindo avaliar qual a solução mais adequada ao contexto e às necessidades específicas de cada loja.

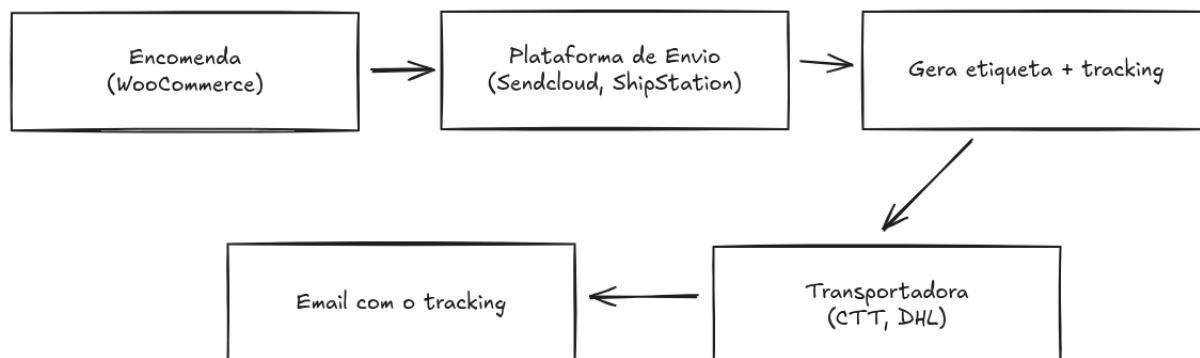


Figura 2: Fluxo de automação do envio e etiquetagem em plataformas integradas com sistemas de e-commerce

Tabela 2: Comparação entre soluções de envio e etiquetagem

Critério	Soluções Manuais	Soluções Automatizadas
Geração de etiquetas	Manual, em plataformas dos transportadores	Automática, integrada com sistema de gestão
Tempo necessário	Elevado, exige <i>login</i> em múltiplos sistemas	Reduzido, processo centralizado e automático
Erros humanos	Frequentes (moradas mal copiadas, formatos errados)	Mínimos, devido à automatização
Integração com transportadoras	Manual, por exportação/importação de dados	Direta via API com CTT, DHL, UPS, etc.
Seguimento de encomendas	Requer input manual do tracking	Envio automático do tracking por email ou SMS

Tabela 3: Comparação de plataformas de envio

Funcionalidade	ShipStation	WooCommerce Shipping	Sendcloud	Easyship / Shippo
Etiquetas em lote	Sim	Sim	Sim	Sim
Suporte a múltiplas transportadoras	Sim	Limitado	Sim	Sim
Tracking e notificações automáticas	Sim	Parcial	Sim	Sim
Regras automáticas de envio	Sim	Não	Sim	Sim
Gestão de devoluções	Não*	Não	Sim	Varia por transportadora

*Requer *plugin* adicional.

2.2.4 Gestão de Encomendas (Picking, Packing e Shipping)

A gestão eficaz de encomendas, desde a seleção (*picking*), embalagem (*packing*) até ao envio (*shipping*), é essencial para a eficiência operacional em lojas de *e-commerce*. Automatizar estes passos reduz erros, acelera prazos e melhora a experiência do cliente, conforme ilustrado no *workflow* apresentado na Figura 3.



Figura 3: *Workflow* automatizado do processo de *picking* e *packing* com validação por código de barras

No contexto do *WooCommerce*, existem *plugins* focados especificamente em tarefas de *picking*, como o *Order Picking For WooCommerce* e o *Picking List for WooCommerce*, que permitem gerar listas de recolha detalhadas e gerir diferentes estados de encomenda, como "parcialmente processada" ou "processada". Estas ferramentas incluem suporte a *scanning* de *QR Codes* ou códigos de barras para validação dos produtos durante a recolha [39, 40]. Além disso, integram funcionalidades de *packing*, como a

impressão de etiquetas, listas de embalagem e atualização automática do estado da encomenda na plataforma [41].

Em operações com maior volume, soluções mais avançadas como o *Veeqo*, *SkuVault* ou *TradShipBo-beGecko* permitem gerir ondas de *picking*, criar rotas otimizadas dentro do armazém, priorizar encomendas e integrar-se com sistemas de gestão de armazéns (*Warehouse Management System (WMS)*). Estas ferramentas oferecem também relatórios em tempo real e aumentam significativamente o controlo sobre o inventário [42, 43].

Entre as principais vantagens destas soluções, destaca-se a precisão operacional resultante do uso de *scanning*, que praticamente elimina erros durante o processo de *picking* e *packing*. A organização em *workflows* otimizados por zonas e ondas de recolha acelera o tempo de processamento, enquanto a atualização automática de *stock* e do estado das encomendas garante maior consistência entre os sistemas. Finalmente, estas abordagens são altamente escaláveis, sendo particularmente adequadas para operações com volumes elevados e múltiplos armazéns [42]. As funcionalidades mais relevantes destas soluções encontram-se sintetizadas de forma comparativa na Tabela 4.

Tabela 4: Comparação de funcionalidades em soluções de gestão de encomendas

Funcionalidade	Order Picking (Woo)	Pick List (Woo)	Pick & Pack Software (G2)
Listas de <i>picking</i> detalhadas	Sim	Sim	Sim
Scanner por código	Não	Sim	Sim
Estado de encomenda integrado	Sim	Sim	Sim
Workflow ondas/prioridade	Não	Não	Sim
Relatórios em tempo real	Não	Não	Sim

2.2.5 Sincronização Multi-Canal

A sincronização de inventário e preços em múltiplos canais — como lojas *online*, *marketplaces*, redes sociais e pontos de venda físicos — é essencial para evitar situações de “*overselling*” e garantir uma experiência de compra consistente em todos os ambientes. Recorrer a uma plataforma centralizada permite ajustar automaticamente o *stock* em tempo real: por exemplo, quando ocorre uma venda num *marketplace* como a *Amazon*, *eBay* ou *Google Shopping*, o registo é automaticamente refletido nos restantes canais, prevenindo ruturas, conforme ilustrado no modelo apresentado na Figura 4 [44].

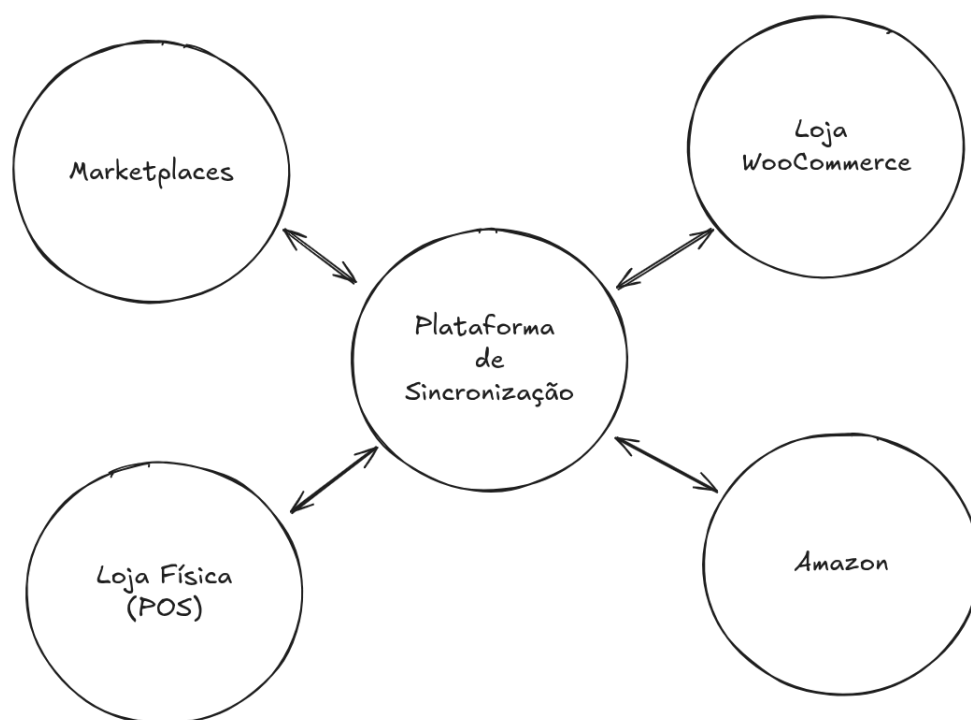


Figura 4: Modelo de sincronização de inventário e preços com múltiplos canais de venda através de uma plataforma central

Além disso, estas plataformas oferecem visibilidade unificada sobre o inventário disponível, o que facilita decisões mais fundamentadas ao nível de compras, logística e campanhas de marketing [45]. A escalabilidade e resiliência são igualmente reforçadas por soluções como o *miniOrange WooCommerce Product Sync*, que permite replicar preços, *stock* e variações entre diferentes lojas *WooCommerce*, com suporte a atualizações via [API](#) ou [Representational State Transfer \(REST\)](#) em tempo real [46].

Outro aspeto diferenciador é a automação através de arquiteturas orientadas a eventos. Ferramentas como o *Feedonomics* utilizam esse modelo para garantir que alterações críticas, como modificações de preços ou níveis de *stock*, sejam propagadas imediatamente entre os canais envolvidos [45].

Entre os exemplos mais eficazes destaca-se o próprio *miniOrange WooCommerce Product Sync*, que sincroniza não apenas *stock* e variações, mas também imagens, categorias e preços, suportando múltiplos formatos como [JSON](#), [XML](#) e [CSV](#) [46]. Por outro lado, o *Feedonomics* centraliza a gestão de inventário e *feeds*, distribuindo-os de forma inteligente por vários canais de venda e publicidade, e garantindo sincronização em tempo real com mecanismos que previnem a sobre-venda [44, 45].

2.2.6 Casos de Estudo

Para ilustrar na prática os benefícios da integração entre plataformas de *e-commerce* e sistemas de gestão empresarial, esta secção apresenta exemplos reais de empresas que adotaram soluções automatizadas para otimizar os seus processos internos. Através destes casos, é possível compreender os desafios enfrentados, as tecnologias escolhidas e os resultados obtidos com a transformação digital das operações.

2.2.6.1 Caso 1: Zippo

A **Zippo**, conhecida globalmente pelos seus icónicos isqueiros, enfrentava dificuldades em escalar o canal *Shopify Plus* devido à gestão manual entre a loja *online* e o sistema **ERP** interno. A Figura 5 apresenta a página principal da loja *online*, que funciona como ponto de entrada para o canal digital da marca. Para resolver os obstáculos de integração, foi implementada uma solução robusta baseada na conectividade entre o *Shopify Plus* e o **ERP** da empresa, usando a plataforma **VL OMNI**, especializada em conectividade omnicanal [47].

A solução contemplou a sincronização bidirecional de produtos, stock, encomendas e dados de clientes, assegurando que toda a informação circulasse com consistência entre os sistemas. Também foi incluída a gestão de encomendas personalizadas com aprovações prévias antes do processamento, bem como o mapeamento automático de ordens para centros de distribuição específicos — assegurando que cada encomenda fosse corretamente encaminhada conforme a sua origem ou destino. Além disso, foram integrados mecanismos de monitorização e relatórios de erro alinhados com as políticas internas de **Information Technology (IT)**.

Os resultados desta transformação foram expressivos: registou-se uma redução significativa de tarefas manuais e de erros de processamento, acompanhada por uma maior agilidade no ciclo de encomendas. Os tempos de resposta a falhas também diminuíram, graças aos relatórios automáticos que permitiam à equipa técnica atuar rapidamente. Por fim, a estrutura omnicanal da *Zippo* foi reforçada, permitindo expandir com confiança para novos mercados e plataformas.

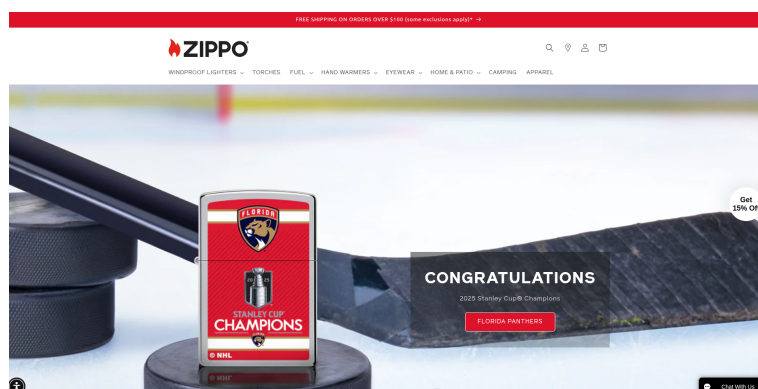


Figura 5: Página principal da loja *online* da Zippo

2.2.6.2 Caso 2: 3DXTECH

A **3DXTECH**, especializada em consumíveis para impressão 3D, enfrentava limitações operacionais provocadas pela existência de múltiplos *sites* *WooCommerce* desconectados, pela gestão manual de faturação e pela segmentação ineficaz entre clientes *Business to Business* (B2B) e *Business to Consumer* (B2C). A Figura 6 apresenta a página principal da loja *online*, que serve como ponto central de interação com estes diferentes segmentos de clientes. Para ultrapassar esses entraves, a empresa optou por migrar para o *Shopify Plus*, integrado com o **ERP Acumatica**, com apoio da consultora **Kensium** [48].

A nova solução permitiu suportar jornadas diferenciadas para clientes B2B e B2C, com preços personalizados e gestão de contas dedicada. Foi ainda implementada sincronização bidirecional de dados — incluindo produtos, *stock*, encomendas e perfis de cliente — através da integração nativa do Acumatica. Para otimizar a experiência de navegação, foi desenvolvido um *Progressive Web App* (PWA), tornando a plataforma mais rápida e responsiva. Por fim, a ligação entre loja e **ERP** assegurou um processamento fiável de encomendas e uma atualização precisa do inventário em tempo real.

Os benefícios foram claros: o desempenho do site aumentou 54%, reduzindo o tempo de carregamento de mais de 40 segundos para menos de 4. A sincronização automática reduziu drasticamente os erros humanos, melhorando o fluxo operacional. O processamento das encomendas tornou-se mais eficiente, refletindo-se em prazos de envio mais curtos e previsíveis. Consequentemente, a satisfação do cliente aumentou, impulsionada por uma experiência de compra mais fluida e fiável.

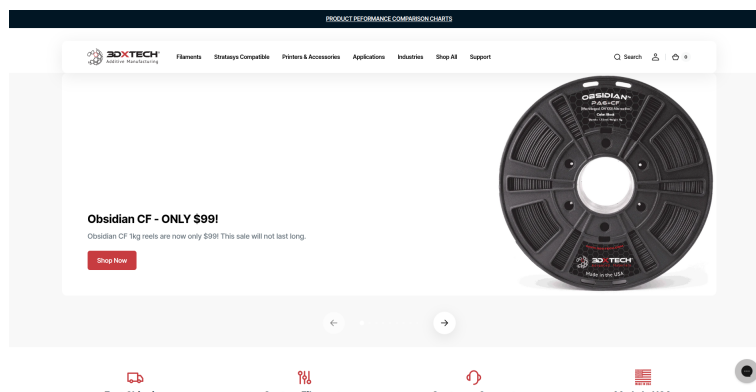


Figura 6: Página principal da loja *online* da 3DXTECH

2.3 Tecnologias e Arquiteturas

A escolha das tecnologias de desenvolvimento representa um dos determinantes do sucesso de projetos digitais, especialmente em ambientes de *e-commerce*, onde a necessidade de escalar, integrar múltiplos canais e responder com agilidade ao mercado é constante. Nesta secção, faremos primeiro uma análise à seleção das linguagens de programação, e depois exploramos, as principais arquiteturas de *software* adequadas ao contexto da solução em estudo.

2.3.1 Linguagens e Frameworks

No *frontend*, **JavaScript** (e o seu superconjunto **TypeScript**) continua a dominar, sustentado por *frameworks* como **React.js** ou **Next.js** que possibilitam interfaces dinâmicas, responsivas e altamente interativas, essenciais para experiências de compras fluidas [49, 50]. No *backend*, a adoção de **Node.js** oferece uma *stack* unificada, reduzindo as barreiras entre equipas e acelerando ciclos de desenvolvimento [51]. Para serviços interligados e *scripts* de suporte, **Python** destaca-se pela sua clareza e vasto ecossistema, especialmente útil em integrações com **API** e automação [52]. Ainda que menos moderno, o **PHP** permanece relevante em plataformas como *WooCommerce*, onde a sua integração nativa acelera customizações [53]. No domínio mobile linguagens nativas como **Swift** (iOS) e **Kotlin** (Android) oferecem um ótimo desempenho, mas *frameworks* multiplataforma como **React Native** e **Flutter** (Dart) têm ganho terreno, permitindo criar aplicações móveis para *Android* e *iOS* a partir de uma única base de código comum, o que reduz custos sem sacrificar a qualidade do produto [54, 55].

2.3.2 Arquiteturas Web

A arquitetura de *software* define como vários componentes de uma aplicação interagem entre si, sendo decisiva para garantir escalabilidade, desempenho e facilidade de manutenção. A seguir são apresentados um conjunto de arquiteturas relevantes no contexto de soluções para *e-commerce*.

Arquitetura Monolítica

Neste modelo, toda a lógica da aplicação reside num único bloco de código, que engloba interface, lógica de negócios e acesso a dados. A sua principal vantagem é a simplicidade: implementação, testes e *deployment* podem ser feitos de forma direta e centralizada. Contudo, à medida que o sistema cresce, a sua escalabilidade e manutenção tornam-se mais complexas, comprometendo a eficiência do desenvolvimento (ver Figura 7) [56, 57].

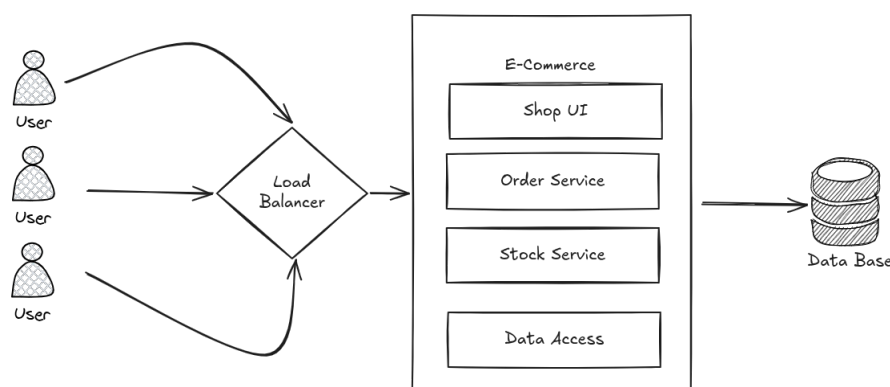


Figura 7: Diagrama de uma arquitetura monolítica

Arquitetura em Camadas

A arquitetura em camadas (*Layered Architecture*) divide o sistema em níveis com responsabilidades distintas, interface, lógica de negócios, persistência e dados. Cada camada comunica apenas com a camada imediatamente adjacente, promovendo modularidade e facilitando testes, manutenção e futuras expansões (Figura 8) [56].

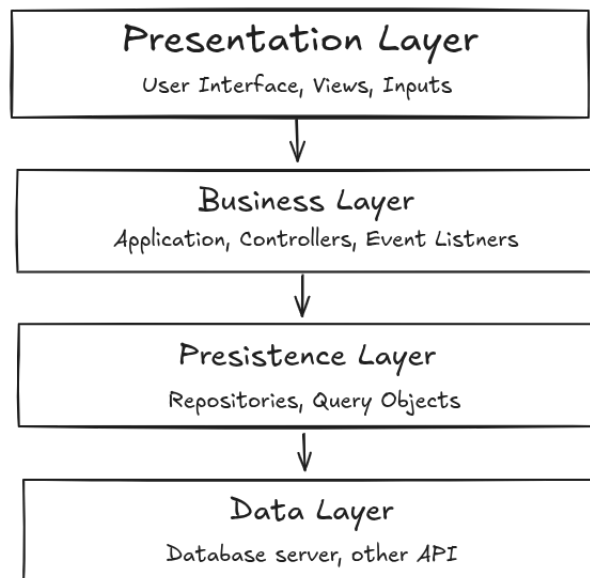


Figura 8: Diagrama de uma arquitetura em camadas

Arquitetura Space-Based

A arquitetura *space-based* distribui o processamento e armazenamento de dados entre unidades independentes (*processing units*), usando uma "tuple space" ou *grid* de memória compartilhada. Esta abordagem permite escalabilidade linear e alto desempenho em sistemas com forte carga concorrente, evitando pontos de estrangulamento em bases de dados centralizadas (Figura 9) [58, 59].

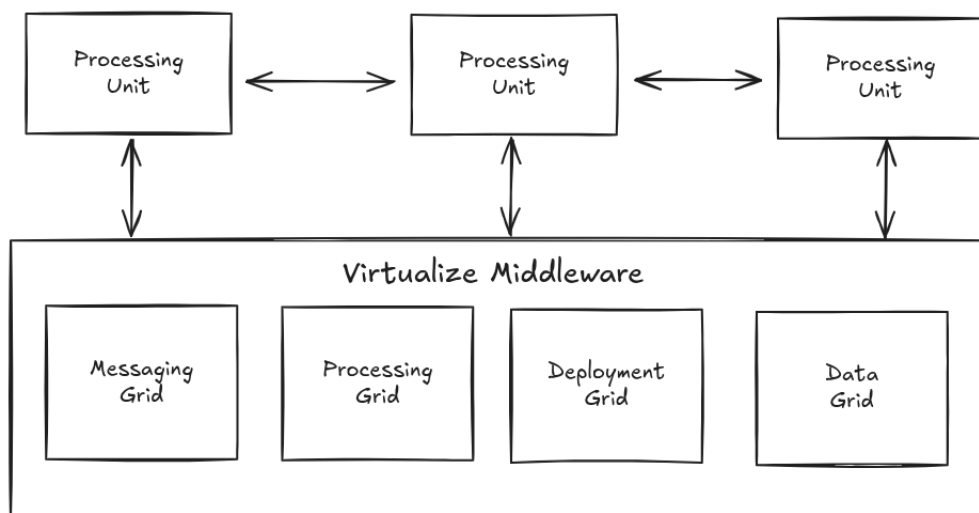


Figura 9: Diagrama de uma arquitetura *space-based*

Arquitetura de Microserviços

A arquitetura de microserviços fragmenta a aplicação em serviços independentes, cada um dedicado a uma função específica, comunicando entre si através de [API](#) ou mensagens. Este modelo promove resiliência, isolamento de falhas, *deploys* rápidos e escalabilidade seletiva, ideal para sistemas complexos e dinâmicos como os de *e-commerce*, com módulos de pagamento, gestão de inventário e processamento de encomendas (Figura 10), embora exija soluções avançadas de orquestração e monitorização [60, 61].

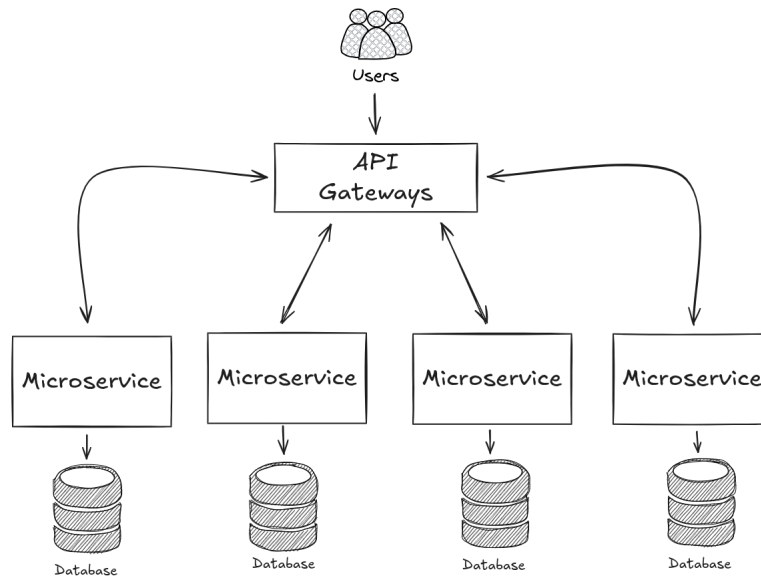


Figura 10: Diagrama de uma arquitetura de microserviços

Arquitetura Orientada a Serviços

A [Service-Oriented Architecture \(SOA\)](#) baseia-se em componentes de negócio, serviços autónomos acessíveis pela rede, o que permite reutilização, interoperabilidade e integração com sistemas heterogêneos. Em *e-commerce*, é útil para organizar serviços como pagamento, inventário ou envios (ver Figura 11), mas requer gestão adequada de contratos e protocolos de comunicação [62, 63].

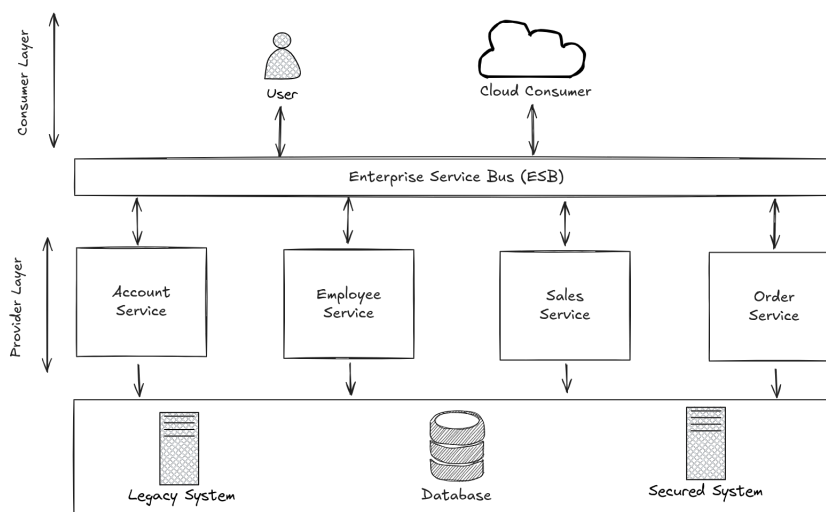


Figura 11: Diagrama de uma arquitetura orientada a serviços (SOA)

Arquitetura Orientada a Eventos

Nesta arquitetura, os componentes interagem através de eventos produzidos e consumidos de forma assíncrona. Ideal para sistemas que precisam de reagir em tempo real (p.ex. atualizações de inventário, notificações de encomendas), assegura baixo acoplamento e alta escalabilidade. Muitas plataformas de *e-commerce* modernas combinam este padrão com microserviços (Figura 12) [64–66].

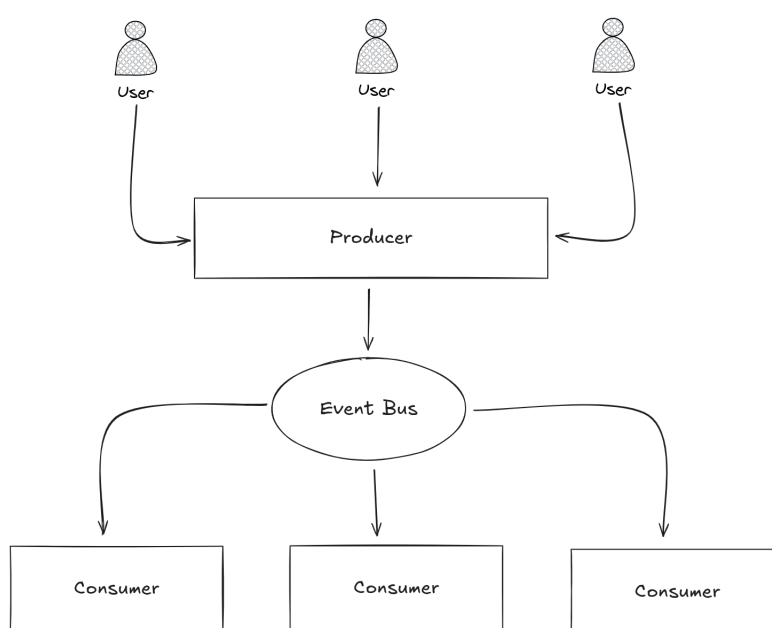
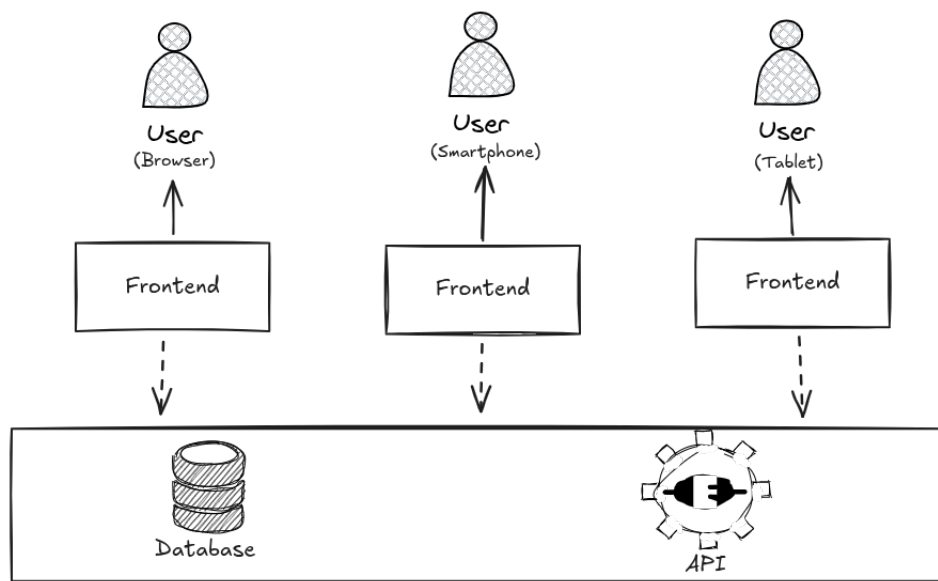


Figura 12: Diagrama de uma arquitetura orientada a eventos (*Event-Driven*)

Arquitetura *Headless*

A arquitetura *headless* separa o *frontend* do *backend*, estabelecendo comunicação entre ambas as camadas exclusivamente através de [API](#). Esta abordagem permite que o mesmo *backend* sirva diferentes interfaces, como *websites*, aplicações móveis ou *smart devices*, oferecendo assim uma experiência omni-canal coerente e flexível. No contexto de *e-commerce*, a arquitetura *headless* facilita atualizações rápidas no *frontend* sem impacto direto na lógica de negócio, reduzindo o tempo de desenvolvimento e o risco de erros em produção (Figura 13). Esta independência entre camadas torna possível escalar e adaptar cada componente de forma autónoma, otimizando o desempenho e a manutenção da infraestrutura [67, 68].

Figura 13: Diagrama de uma arquitetura *headless*

Em síntese, as arquiteturas apresentadas oferecem diferentes abordagens para responder a requisitos de escalabilidade, desempenho, integração e flexibilidade em soluções de *e-commerce*. Modelos mais tradicionais, como a arquitetura monolítica ou em camadas, podem ser adequados para cenários de menor complexidade ou em fases iniciais de um projeto, enquanto arquiteturas baseadas em microserviços, eventos ou *headless* se revelam mais indicadas para ambientes dinâmicos, com múltiplos canais e elevada exigência de integração com sistemas externos. A escolha da arquitetura mais apropriada deve, por isso, ser guiada pelos objetivos de negócio, pelas características operacionais da organização e pelo nível de maturidade tecnológica existente.

Análise e Conceção

Este capítulo faz a ponte entre a motivação prática identificada na introdução e as soluções propostas para o desenvolvimento da plataforma. São apresentados os requisitos que orientam o projeto, os principais casos de utilização para cada tipo de utilizador e a arquitetura escolhida, sempre com o objetivo de garantir uma gestão mais eficiente e centralizada das operações numa loja *online*. Termina-se com o planeamento visual da interface, pensado para simplificar o uso diário do sistema.

3.1 Requisitos do Sistema

O processo de análise de requisitos serve de ponte entre a visão inicial do projeto e a sua concretização prática, orientando o desenvolvimento para uma compreensão clara do que deve ser alcançado. Tal como foi referido na secção dos objetivos, é essencial identificar os requisitos funcionais e não funcionais que garantem que a plataforma cumpre o seu propósito.

Importa sublinhar que, tratando-se de uma aplicação conceptual, alguns destes requisitos poderão divergir dos que seriam definidos num projeto real para um cliente específico. Ainda assim, estes requisitos estabelecem uma base sólida e abrangente para o desenvolvimento, servindo de referência para as decisões técnicas tomadas ao longo do trabalho.

3.1.1 Requisitos Funcionais

A definição dos requisitos funcionais permite garantir que a plataforma responde às necessidades essenciais da gestão de uma loja *online*. Estes requisitos resultam de uma análise prática dos desafios diários enfrentados por quem gere o negócio, servindo de base para o desenvolvimento de uma solução eficaz, simples e adaptada à realidade do comércio digital.

Assim, a plataforma contempla as seguintes funcionalidades principais:

- **Gestão de Inventário:** A plataforma permite registar a entrada de novos produtos em *stock*, bem como atualizar quantidades existentes de forma ágil e transparente. O *coworker* pode visualizar, em tempo real, o inventário disponível, facilitando a identificação de ruturas ou excesso de produtos, e planeando atempadamente reabastecimentos.

- **Processamento de Encomendas:** Todas as encomendas recebidas são centralizadas num único painel, onde podem ser acompanhadas desde a sua receção até à expedição. É possível consultar o histórico e estado de cada pedido, alterar o seu progresso (ex: "em preparação", "expedida", "concluída") e garantir o rastreio eficaz de todas as etapas do processo.
- **Emissão de Faturas:** Após a finalização de cada encomenda, o sistema gera automaticamente a respetiva fatura, assegurando a conformidade legal e a organização documental. As faturas podem ser consultadas e exportadas facilitando também a comunicação com o cliente.
- **Geração de Etiquetas de Envio:** Para cada encomenda pronta a ser expedida, a plataforma cria etiquetas de envio personalizadas, contendo as informações essenciais para a transportadora e para o cliente. Este processo automatizado reduz erros e acelera significativamente a preparação das expedições.
- **Atualização via CSV:** O sistema possibilita a importação em massa de dados, permitindo atualizar preços e quantidades de vários produtos simultaneamente através de ficheiros [Comma-Separated Values \(CSV\)](#). Esta funcionalidade é especialmente útil para lojas com grandes catálogos, tornando as operações mais eficientes e menos sujeitas a erro humano.
- **Gestão de Utilizadores:** Existem diferentes perfis de utilizador (por exemplo, administrador e operador), cada um com níveis de acesso e permissões distintos. Esta diferenciação garante que cada colaborador só pode aceder às funcionalidades relevantes para o seu papel, aumentando a segurança e a organização interna.
- **Integração Multi-Canal:** A plataforma foi pensada para consolidar produtos e encomendas provenientes de vários canais de venda, como diferentes lojas *online*, *marketplaces* ou lojas físicas, permitindo sincronizar o *stock* e acompanhar pedidos em tempo real, evitando erros e duplicação de tarefas.

3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais definem as características de qualidade essenciais para o funcionamento estável e eficiente da plataforma. Estes aspetos garantem que a solução responde com fiabilidade às necessidades das operações diárias numa loja *online*.

Assim, foram definidos os seguintes requisitos não funcionais:

- **Usabilidade:** O interface é desenhado para ser intuitivo, com menus claros e fluxos de trabalho simples, permitindo que qualquer utilizador, mesmo sem experiência técnica avançada, consiga utilizar o sistema de forma eficaz e autónoma no ambiente de embalamento.
- **Desempenho:** A plataforma assegura tempos de resposta rápidos, mesmo quando utilizada por vários operadores em simultâneo ou durante picos de atividade, garantindo que as operações do dia a dia não são prejudicadas por atrasos ou lentidão.

- **Escalabilidade:** O sistema está preparado para crescer juntamente com o negócio, acomodando um número crescente de encomendas, produtos e utilizadores, sem comprometer o desempenho ou a estabilidade.
- **Segurança:** Foram implementados mecanismos de autenticação e gestão de permissões para proteger os dados críticos da loja e dos clientes, garantindo o cumprimento das melhores práticas de segurança digital.
- **Manutenção:** O código e a estrutura do sistema facilitam intervenções futuras, seja para corrigir problemas, implementar melhorias ou adicionar novas funcionalidades, promovendo uma manutenção rápida e eficaz.
- **Fiabilidade:** A plataforma foi desenhada para garantir uma elevada disponibilidade, minimizando riscos de falha e interrupção, para que o funcionamento das operações nunca seja comprometido, mesmo em situações imprevistas.

3.2 Visão Geral dos Casos de Utilização

O funcionamento diário da plataforma é marcado por rotinas simples e diretas, pensadas para responder às necessidades concretas de uma loja *online* que faz toda a gestão operacional numa bancada de embalamento. O sistema foi desenhado para que os utilizadores possam executar as principais tarefas do dia a dia sem perder tempo com processos desnecessários ou dispersos.

No início do turno, o operador acede ao sistema e verifica o inventário disponível, podendo de imediato registar a entrada de novos produtos ou atualizar as quantidades de *stock* conforme a chegada de mercadoria ao armazém. Este processo garante que o inventário está sempre atualizado e pronto a responder aos pedidos dos clientes.

À medida que as encomendas vão sendo recebidas de diferentes canais de venda, estas aparecem centralizadas no painel principal da plataforma. O operador pode consultar o detalhe de cada pedido, alterar o seu estado à medida que progride no processo (por exemplo, de "em preparação" para "expedida") e gerar, de forma automática, tanto a fatura como a etiqueta de envio para cada encomenda.

Sempre que for necessário atualizar preços ou quantidades de vários produtos ao mesmo tempo, o administrador pode importar ficheiros *CSV*, agilizando operações de grande volume. Além disso, é o administrador que gere os acessos dos diferentes utilizadores, atribuindo permissões de acordo com o papel de cada um na operação.

A integração multi-canal assegura que produtos e encomendas provenientes de vários sites, *market-places* ou pontos físicos são sempre sincronizados, evitando erros de *stock* e simplificando a preparação dos envios.

No fundo, cada caso de utilização traduz um fluxo natural e direto: desde a entrada de produtos, passando pelo tratamento das encomendas, até à expedição final, tudo centralizado numa única plataforma, sem necessidade de recorrer a sistemas externos ou processos manuais complexos.

3.3 Conceção

Esta secção apresenta as principais decisões tomadas durante a fase de desenho do sistema, tendo como base os requisitos funcionais e não funcionais definidos anteriormente. Inicialmente, são explicadas as opções arquitetónicas fundamentais, seguidas pela estruturação do modelo de dados que serve de base à gestão operacional. Por fim, descrevem-se as escolhas tecnológicas para o *backend* e *frontend*, bem como a visão geral da arquitetura final da plataforma.

3.3.1 Arquitetura

A arquitetura proposta resulta diretamente das necessidades identificadas nos requisitos funcionais e não funcionais mencionados em 3.1, nomeadamente a necessidade de centralização, facilidade de manutenção, escalabilidade e integração multi-canal. Todo o sistema foi desenhado para garantir que operações como a gestão de inventário, o processamento de encomendas, a emissão de faturas, a geração de etiquetas de envio, a importação via ficheiro [CSV](#) e a gestão de utilizadores possam ser realizadas de forma eficiente e sem redundâncias.

Neste sentido, toda a informação operacional, desde os registos de produtos e utilizadores, passando pelo acompanhamento das encomendas e atualização do *stock*, está organizada numa única base de dados central. Esta decisão assegura que existe uma única fonte de verdade para todos os intervenientes, prevenindo discrepâncias e facilitando a automatização dos processos do dia a dia.

Para garantir a separação entre a lógica de negócio e a camada de apresentação, a comunicação entre o *frontend* e o *backend* é efetuada através de uma [Representational State Transfer Application Programming Interface \(REST API\)](#). Esta abordagem permite, por um lado, manter o sistema modular e facilmente extensível, e por outro, criar condições para futuras integrações com outros sistemas ou plataformas externas.

A escolha de uma arquitetura *headless*, com recurso a um [Content Management System \(CMS\)](#) dedicado à gestão dos conteúdos e entidades, está diretamente relacionada com a necessidade de compatibilidade e integração com diferentes plataformas de venda, como referenciado nos objetivos e requisitos do sistema. Esta estratégia permite que a plataforma centralize e sincronize informação proveniente de vários canais (por exemplo, diferentes lojas *online* ou *marketplaces*), garantindo sempre a atualização em tempo real de dados críticos como o *stock* e o estado das encomendas.

A centralização da informação, associada à utilização de uma [Application Programming Interface \(API\)](#) e de um *backend headless*, permite responder eficazmente aos desafios de usabilidade, desempenho, escalabilidade, segurança e fiabilidade destacados nos requisitos não funcionais, assegurando que a plataforma consegue crescer e adaptar-se às exigências do negócio sem comprometer a integridade ou a eficiência operacional, conforme ilustrado na Figura 14.

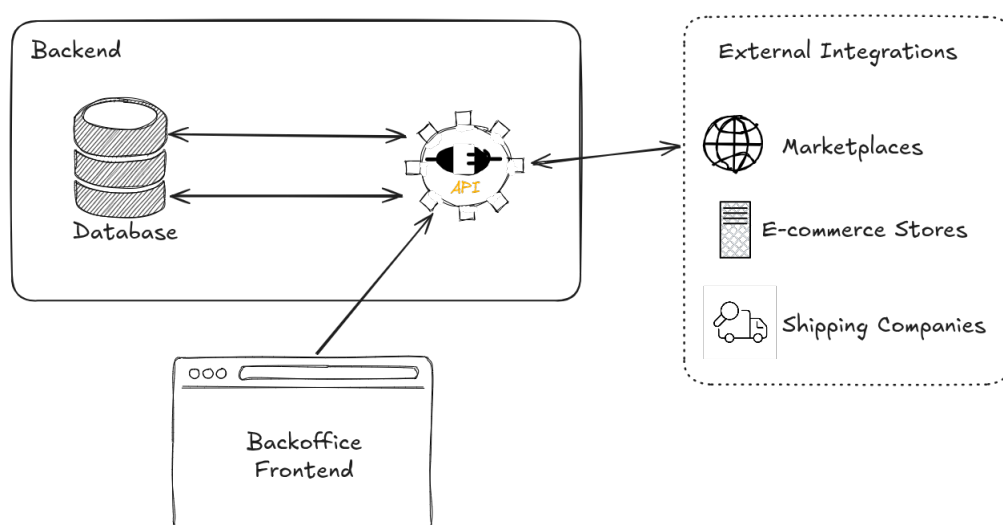


Figura 14: Esquema da arquitetura

3.3.2 Modelo de Dados

O modelo de dados da plataforma foi desenvolvido para suportar de forma eficaz todas as operações críticas de gestão, integrando de forma centralizada informação sobre utilizadores, produtos, encomendas, faturação e integração multi-canal. A estrutura foi desenhada para garantir robustez, flexibilidade e facilidade de manutenção, assegurando a consistência dos dados ao longo de todo o processo operacional.

A Figura 15 apresenta o diagrama entidade-relacionamento ([Entity-Relationship \(ER\)](#)) simplificado da base de dados implementada. Nesta representação são exibidas apenas as chaves primárias e estrangeiras de cada tabela, com o objetivo de realçar as relações estruturais entre entidades e melhorar a legibilidade do modelo. Os restantes atributos encontram-se documentados no dicionário de dados incluído em anexo.

No centro do modelo encontra-se a tabela **user**, responsável pelo registo e autenticação dos utilizadores da plataforma, bem como pela associação aos respetivos perfis de permissões (*Super-Admin*, *Admin* e *Coworker*). A tabela **site** representa os diferentes canais de venda e estabelece ligações diretas às principais entidades operacionais, incluindo **order**, **productSite** e **wooCommerceCredentials**, esta última dedicada ao armazenamento das credenciais necessárias à integração com plataformas externas.

As encomendas são geridas através da tabela **order**, que relaciona cada pedido com o cliente, o endereço de entrega e o canal de origem. As tabelas **orderTransaction** e **orderShippingInfo** complementam o módulo de encomendas, registando respetivamente os artigos incluídos e as informações de envio. As entidades **client** e **address** garantem o registo estruturado dos dados dos clientes e das suas moradas.

A gestão de produtos é centralizada na tabela **product**, à qual se associam diferentes configurações por canal através da tabela **productSite**. A rastreabilidade e o controlo do *stock* são assegurados pela tabela **productTransactions**, que mantém o histórico completo de entradas, saídas e ajustes de

inventário, permitindo auditoria e atualização contínua dos níveis de produtos.

O módulo de faturação é composto pelas tabelas **invoice** e **invoiceTransactions**, que registam as faturas de compra e os respetivos produtos adquiridos a fornecedores, identificados na tabela **supplier**. Este conjunto de relações possibilita a atualização automática do inventário e o acompanhamento das operações de abastecimento. Adicionalmente, a tabela **fedExCurrency** mapeia os códigos de moeda utilizados pela FedEx, garantindo compatibilidade nas transações internacionais.

A estrutura apresentada promove uma arquitetura de dados coesa e escalável, que suporta a automação de tarefas críticas, simplifica o processamento das encomendas e assegura que toda a informação relevante permanece acessível e atualizada, sustentando de forma fiável a operação do *backoffice*.

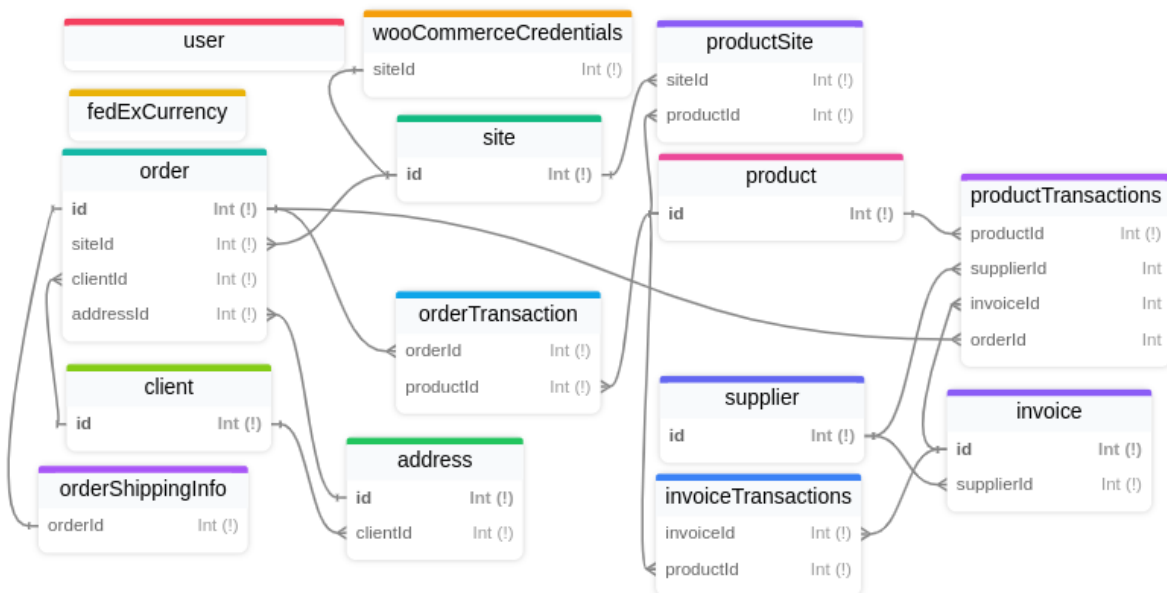


Figura 15: Diagrama entidade-relacionamento simplificado (chaves primárias e estrangeiras).

3.4 Arquitetura Final

Esta secção apresenta a versão final da arquitetura da plataforma, consolidando as decisões tomadas ao longo do processo de análise e desenho descritas anteriormente. São aqui detalhadas as principais escolhas tecnológicas que suportam a solução, assim como os motivos da sua adoção. A arquitetura final integra todas as componentes essenciais do sistema, garantindo o cumprimento dos requisitos funcionais e não funcionais previamente identificados.

A Figura 16 ilustra a visão global da arquitetura implementada, destacando as ligações entre *frontend*, *backend*, base de dados central e as integrações externas necessárias. A seleção das tecnologias recaiu sobre critérios como desempenho, escalabilidade, segurança, facilidade de integração e custo, assegurando que a plataforma se mantém robusta e preparada para responder a futuras necessidades de crescimento ou adaptação.

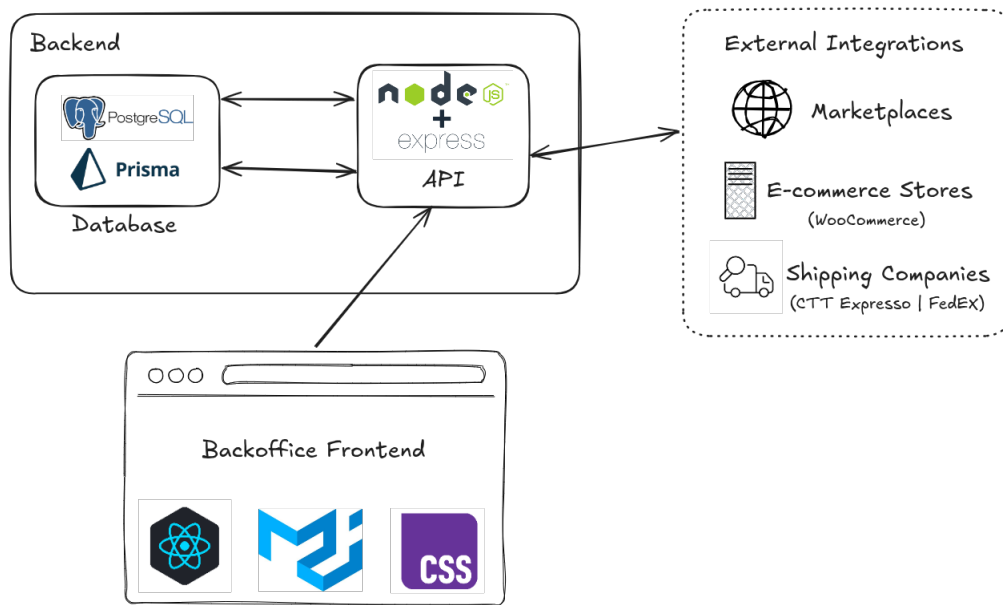


Figura 16: Arquitetura geral do sistema e integração entre componentes.

3.4.1 Backend

O *backend* da plataforma foi integralmente desenvolvido de raiz, recorrendo a Node.js [69] e Express [70] para garantir máxima flexibilidade e total controlo sobre a lógica de negócio, o modelo de dados e as integrações externas. Esta abordagem permitiu construir uma solução totalmente ajustada aos requisitos específicos do projeto, sem restrições impostas por *frameworks* ou sistemas pré-existentis.

Toda a arquitetura backend segue o princípio *headless*, disponibilizando uma *API* que serve tanto o *backoffice* como os dois sites de *e-commerce* implementados em *WooCommerce*. Assim, é possível centralizar a gestão dos dados e garantir a sincronização de informações críticas, como inventário e encomendas, entre múltiplos canais de venda.

O modelo de dados foi desenhado de raiz para responder às necessidades reais do negócio, não havendo dependência de modelos predefinidos ou limitações típicas de *CMS*. A camada de persistência é assegurada por uma base de dados PostgreSQL [71], cuja comunicação é gerida através do *Object-Relational Mapping (ORM)* Prisma [72], facilitando operações transacionais seguras e eficientes.

No que respeita à autenticação de utilizadores, foi implementado um sistema próprio, recorrendo a *hashing* seguro das palavras-passe e armazenamento de *tokens*, de modo a garantir a proteção dos dados de acesso. Adicionalmente, todas as credenciais sensíveis — nomeadamente as chaves e *tokens* de acesso às *API* externas (*WooCommerce*, CTT Expresso, FedEx) — são encriptadas e armazenadas de forma segura, com utilização de variáveis de ambiente (`.env`) para evitar exposição de informação crítica no código-fonte.

As integrações externas com as plataformas de *e-commerce* (*WooCommerce*), operadores logísticos (FedEx, CTT Expresso) e mecanismos de *tracking* são geridas diretamente no *backend*, permitindo automatizar tarefas como a sincronização de encomendas, atualização de *stock* e gestão de envios. Este

desenho reduz a dependência de serviços terceiros e simplifica a manutenção evolutiva do sistema.

A implementação focou-se na robustez e simplicidade, sem recurso a funcionalidades extra como *web-sockets*, sistemas de permissões complexos ([Role-Based Access Control \(RBAC\)](#)) ou *plugins*. O objetivo foi assegurar uma base sólida, fácil de manter e expandir, preparada para integrar novas funcionalidades no futuro caso seja necessário.

A opção por um *backend* desenvolvido de raiz permitiu também uma abordagem personalizada à segurança, desempenho e gestão de dados, garantindo a conformidade com as melhores práticas sem comprometer a autonomia do projeto.

3.4.2 Frontend

O desenvolvimento do *frontend* da plataforma foi realizado com **React**, uma das bibliotecas mais populares e robustas para aplicações *web* modernas [73]. A escolha do *React* deve-se à sua flexibilidade, grande comunidade e capacidade de construir interfaces modulares, baseadas em componentes reutilizáveis.

A gestão do estado da aplicação foi feita através da passagem de *props* entre componentes *pai* e *filho*, evitando dependências externas complexas. Para a navegação entre as diferentes páginas da plataforma foi utilizado o **React Router**, o que permitiu implementar uma abordagem de [Single Page Application \(SPA\)](#), onde cada funcionalidade principal (como encomendas, produtos ou gestão de expedição) tem a sua própria rota dedicada [74]. Desta forma, o fluxo de navegação é intuitivo e direto, sem recarregamentos completos de página.

Ao nível do *design* visual, recorreu-se principalmente à biblioteca **Material UI**, que oferece um conjunto abrangente de componentes prontos a usar, com um aspeto moderno e responsivo [75]. Sempre que necessário, foram feitos pequenos ajustes com CSS puro, especialmente em elementos específicos que exigiam personalização adicional. Esta combinação permitiu garantir uma interface consistente, rápida e fácil de manter.

A comunicação entre o *frontend* e o *backend* foi implementada com a biblioteca **Axios**, que simplifica o processo de realizar chamadas [Hypertext Transfer Protocol \(HTTP\)](#) ([REST API](#)) e gerir respostas e erros de forma assíncrona [76]. Todas as interações com dados — como criação de encomendas, atualização de produtos ou consulta de *stock* — são feitas diretamente a partir dos componentes, através de chamadas *Axios* ao *backend Node.js/Express*.

O sistema não contempla internacionalização nem preocupações específicas de acessibilidade, visto que o público-alvo é restrito e a aplicação está totalmente em inglês. Da mesma forma, não foi dada prioridade à responsividade, já que a plataforma está pensada para uso exclusivo em ambiente *desktop*, numa bancada de embalamento.

As notificações e feedback visual ao utilizador são assegurados através dos componentes **Snackbar** e **Alert** do *Material UI*, garantindo uma experiência clara e sem ruído visual excessivo. Não existem animações complexas ou estilos pesados, privilegiando sempre o desempenho e a rapidez de interação.

No geral, as decisões tecnológicas privilegiaram a simplicidade, a robustez e a eficiência, sem dependências excessivas nem camadas desnecessárias.

Implementação

Este capítulo descreve a implementação do projeto, organizada em duas secções: *backend* e *frontend*. No *backend*, detalham-se a configuração inicial do serviço, o modelo de controlo de acesso e papéis, a gestão da base de dados e o desenvolvimento de métodos funcionais personalizados que vão além do [Create, Read, Update, Delete \(CRUD\)](#). Na secção do *frontend*, apresenta-se a configuração da aplicação *React*, a organização do mapa de navegação e a estrutura geral, com ênfase na reutilização de componentes e funções para construir uma interface eficiente e fácil de manter, seguida de uma visão geral das principais funcionalidades.

4.1 Backend

Nesta secção apresentam-se os detalhes da implementação do *backend*. A discussão inicia-se com a configuração e arranque do serviço. De seguida, descrevem-se os papéis e as permissões definidos para reforçar a robustez e a segurança do sistema, garantindo que apenas utilizadores autorizados acedem a determinadas funcionalidades. É também caracterizada a estrutura das tabelas criadas na base de dados, salientando de que modo suportam as funcionalidades do sistema. Adicionalmente, explicam-se os métodos personalizados implementados no projeto, evidenciando como respondem a necessidades específicas e melhoram a eficiência do sistema. Por fim, apresenta-se o *backoffice*, com destaque para as suas principais funcionalidades de administração e monitorização das operações.

4.1.1 Setup do Backend

Nesta subsecção descreve-se o *setup* e a configuração do serviço *backend*, desenvolvido em **Node.js 22** com **Express** (^{4.18.2}) e **PostgreSQL 14.18** como sistema de gestão de base de dados. O mapeamento objeto-relacional é assegurado pelo **Prisma** (6.14.0), adotando-se *PostgreSQL* em todos os ambientes para garantir consistência entre desenvolvimento e produção.

O projeto foi inicializado via [Command Line Interface \(CLI\)](#) ([Node Package Manager \(NPM\)](#)), com a instalação das dependências e a configuração do Prisma (`npx prisma init`). Os parâmetros sensíveis residem no ficheiro `.env`, incluindo `DATABASE_URL`, `JWT_SECRET_KEY`, `JWT_EXPIRES_IN` (definida para

7d) e `ENCRYPTION_KEY`. A autenticação de utilizadores recorre a [JSON Web Token \(JWT\)](#) em esquema *bearer*, enquanto as palavras-passe são protegidas com **bcrypt**. Credenciais e dados externos sensíveis são encriptados em repouso utilizando a `ENCRYPTION_KEY`.

Logo no arranque, a aplicação aplica *middlewares* de segurança e robustez: *helmet* (cabeçalhos [Hypertext Transfer Protocol \(HTTP\)](#)), [Cross-Origin Resource Sharing \(CORS\)](#) com lista explícita de origens permitidas e deteção dinâmica de *localhost*, *express-rate-limit* (100 pedidos/hora por IP no prefixo `/api`), *parsers* de corpo ([JavaScript Object Notation \(JSON\)](#) e *urlencoded*) com limite de 10 KB, *cookie-parser*, sanitização contra [Cross-Site Scripting \(XSS\)](#) e *compression*. Em desenvolvimento ativa-se *morgan* para registo de pedidos.

A configuração central destes *middlewares* encontra-se resumida na **Listagem 4.1**, enquanto o respetivo fluxo de execução é apresentado na Figura 17, que distingue os *middlewares* globais — aplicados no arranque da aplicação — dos *middlewares* de segurança *protect* e *restrictTo*, aplicados a nível de rota.

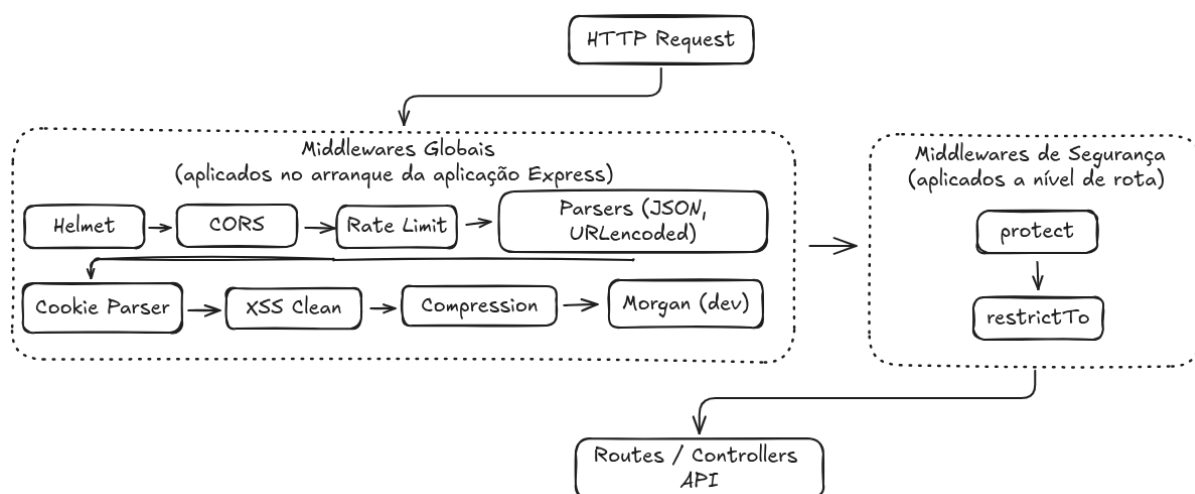


Figura 17: Fluxo de execução dos *middlewares* globais e de segurança no servidor *Express*.

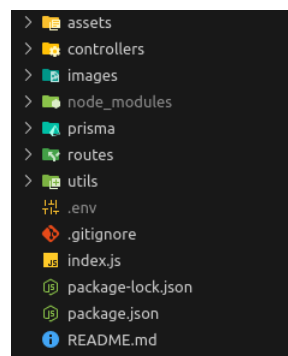
```

1 const allowedOrigins = new Set([
2   'https://lobao-backoffice.pt', 'http://localhost:3001', 'http://127.0.0.1:3001',
3 ]);
4
5 const corsOptions = {
6   origin(origin, cb) {
7     if (!origin) return cb(null, true);
8     const isLocalhost = /^https?:\/\/(localhost|127\.0\.0\.1)(:\d+)?$/i.test(origin)
9     ;
10    cb(null, allowedOrigins.has(origin) || isLocalhost);
11  },
12  credentials: true,
13 };
14 app.use(cors(corsOptions));
15 app.use(helmet());
16
  
```

```
17 if (process.env.NODE_ENV === 'development') app.use(morgan('dev'));
18
19 app.use(
20   rateLimit({
21     max: 100,
22     windowMs: 60 * 60 * 1000,
23     message: 'Too many requests, please try again later.',
24   })
25 );
26
27 app.use(express.json({ limit: '10kb' }));
28 app.use(express.urlencoded({ extended: true, limit: '10kb' }));
29 app.use(cookieParser());
30 app.use(xss());
31 app.use(compression());
```

Listagem 4.1: Configuração de *middlewares* globais do servidor

Toda a administração e operação do *backoffice* é exposta via [Representational State Transfer Application Programming Interface \(REST API\)](#), sem recurso a [Content Management System \(CMS\)](#) de terceiros; a Figura 18 apresenta a estrutura de pastas do serviço. Existem, também, integrações externas específicas (ver Fig. 16): (i) geração de etiquetas com *CTT Expresso* e *FedEx* e (ii) sincronização com as lojas de *e-commerce* (*WooCommerce*) para criação e atualização de produtos e encomendas provenientes do site, mantendo a base de dados central como fonte única de verdade.

Figura 18: Estrutura de pastas do *backend*.

4.1.2 Autenticação e Papéis

Nesta secção analisam-se o fluxo de autenticação e a configuração de [Role-Based Access Control \(RBAC\)](#) do *backend*, implementados de forma nativa em *Node.js/Express*. A autenticação é baseada em [JWT](#), emitidos após validação de credenciais e transmitidos no cabeçalho *Authorization: Bearer*. Os *tokens* têm uma validade de 7d e, nesta versão, não existem *refresh tokens*. O controlo de acesso é aplicado por *middleware* a nível de rota, garantindo que apenas utilizadores com o papel adequado executam operações sensíveis. As contas podem ser marcadas como inativas (*isActive*), impedindo o acesso a *endpoints* protegidos mesmo que as credenciais estejam corretas.

4.1.2.1 Autenticação

O fluxo de autenticação é exposto sob o prefixo `/api/auth` e assenta em [JWT](#) enviado no cabeçalho `Authorization: Bearer {token}`. O `login` (`POST /api/auth/login`) valida credenciais e, em caso de sucesso, emite um `token` com validade de `7d`, cujo `payload` inclui apenas o identificador do utilizador. A determinação do papel e das permissões é efetuada no servidor a cada pedido subsequente, após validação do `token` e leitura do utilizador em base de dados.

A criação de contas não é pública. O `signup` (`POST /api/auth/signup`) é um `endpoint` reservado ao contexto administrativo: em operação normal, apenas perfis administrativos o utilizam para criar novas contas; na fase de instalação, o mesmo `endpoint` aceita a chave `ADMIN_KEY` para permitir o `bootstrap` do primeiro utilizador com papel `super_admin`. Fora desse cenário inicial, a `ADMIN_KEY` não é utilizada nem exposta ao `frontend`.

Foram implementados os fluxos de recuperação e redefinição de palavra-passe: `POST /api/auth/forgotPassword` emite um `token` temporário (válido por 10 minutos) enviado por e-mail; `PATCH /api/auth/resetPassword/:token` valida o `token` e aplica a nova palavra-passe. As palavras-passe são protegidas com `bcrypt` (12 rounds); a data de alteração (`passwordChangedAt`) é registada, e o `middleware` de proteção invalida `tokens` emitidos antes dessa alteração. Não existe `logout` dedicado nem `refresh tokens` nesta versão — a sessão termina por expiração do [JWT](#) ou por bloqueio de conta.

O `middleware protect` extrai e verifica o `token`, carrega o utilizador corrente e impede o acesso quando a conta não existe, se encontra inativa (`active = false`) ou se a palavra-passe foi alterada após a emissão do `token`. Conforme ilustrado na Figura 17, este `middleware` é executado após os `middlewares` globais do servidor, atuando como camada de segurança que valida o contexto de autenticação antes da execução das rotas protegidas. As rotas do próprio utilizador em `/api/v1/users` (por exemplo, `GET /me`, `PATCH /updateMe`, `PATCH /updateMyPassword` e `DELETE /deleteMe`) exigem autenticação. Por não serem usados `cookies` de sessão, não se aplica proteção [Cross-Site Request Forgery \(CSRF\)](#); `rate limiting` específico para as rotas de autenticação não foi introduzido nesta versão.

4.1.2.2 Papéis e Permissões

O controlo de acesso é implementado segundo um modelo [RBAC](#) aplicado por `middleware` a nível de rota, combinando `protect` (validação do [JWT](#) e carregamento do utilizador) com `restrictTo` (verificação do papel). Como ilustrado na Figura 17, o `restrictTo` é aplicado posteriormente ao `protect`, assegurando que apenas utilizadores autenticados e com o papel apropriado acedem a cada rota. Adotam-se três papéis globais — `super_admin`, `admin` e `coworker` (doravante, *Operador*) — sem `scoping` por loja: as permissões são globais à plataforma.

Super-Admin e Admin. Estes dois perfis são equivalentes em termos de privilégios. Têm acesso total ao `backoffice` e a todas as rotas administrativas, incluindo: gestão de utilizadores (`/api/v1/admin`), gestão de `sites` e respetivas credenciais de integrações (`/api/v1/sites`, com criação/atualização de credenciais `WooCommerce`), operações de sincronização e consulta por site (`/api/v1/woocommerce`), e atualização em massa do catálogo por [Comma-Separated Values \(CSV\)](#) (`/api/v1/sites/:siteId/updater`).

Para além disso, podem executar todas as operações operacionais (encomendas, faturação, expedição, clientes e *stock*) tal como o *Operador*. Em síntese, *super_admin* e *admin* podem “fazer tudo”, incluindo alterar credenciais externas e atualizar produtos dos *sites* via [CSV](#).

Operador (*coworker*). Foca-se nas tarefas diárias de operação. Tem acesso autenticado às rotas de encomendas (*/api/v1/orders*, incluindo consulta, pesquisa, listagem por estado e *picking*, bem como atualização de estado e cancelamento quando aplicável), faturação operacional (*/api/v1/invoices* e */api/v1/invoice-client*), clientes e moradas (*/api/v1/clients*), e expedição através das integrações com transportadoras (*/api/v1/carrier* — *FedEx* e *CTT Expresso*). Os fluxos de expedição encadeiam a emissão de fatura, a atualização de *stock* e a criação do envio, garantindo que a transição de estado da encomenda ocorre de forma transacional. No catálogo, o *Operador* pode consultar e pesquisar produtos (incluindo operações de *picking*) e realizar ajustes de *stock* quando previstos pelos fluxos operacionais; a criação/edição estrutural em massa e a gestão de credenciais permanecem reservadas a perfis administrativos.

Esta configuração — refletida nas rotas com *restrictTo('admin','super_admin')* para módulos administrativos (utilizadores, *sites*, credenciais e *CSV updater*) e *restrictTo('coworker','admin','super_admin')* para módulos operacionais (encomendas, faturação, clientes, expedições) — separa claramente responsabilidade administrativa da operação diária, mantendo a superfície de ataque reduzida e o ciclo de trabalho objetivo para quem está na bancada.

4.1.3 Base de Dados

A base de dados do projeto assenta em **PostgreSQL**, com modelação declarativa em *Prisma* e migrações versionadas que asseguram a evolução controlada do esquema entre ambientes. O objetivo foi garantir integridade referencial, rastreabilidade e simplicidade de operação: as entidades nucleares incluem campos de auditoria (*created_at* e *updated_at* com *@updatedAt*) e as relações são expressas por chaves estrangeiras explícitas, evitando ambiguidade nos fluxos de escrita.

As Figuras 19 e 20 apresentam a coleção de tabelas que compõem a base de dados implementada. Por razões de legibilidade, o diagrama foi dividido em duas partes: a primeira ilustra as entidades associadas à gestão de utilizadores, canais de venda, clientes e encomendas, enquanto a segunda apresenta as entidades relacionadas com a gestão de fornecimentos, faturas, fornecedores e movimentos de *stock*. O diagrama [Entity-Relationship \(ER\)](#) completo foi apresentado no capítulo anterior (Figura 15) e serve de referência estrutural para a descrição que se segue.

No núcleo do sistema está a tabela ***user***, responsável pela gestão de contas e perfis de acesso. Cada utilizador possui *email* único, palavra-passe protegida com *hash*, indicadores de segurança como *passwordChangedAt* e campos de recuperação, bem como o papel (*Role*) que determina as permissões efetivas no *backoffice*. Em paralelo, a tabela ***site*** representa cada canal de venda operado pela plataforma e estabelece o contexto para sincronização de catálogos e encomendas. A integração com *WooCommerce* é modelada por uma relação 1:1 entre ***site*** e ***woocommerceCredentials*** (chave primária *siteId*), onde são armazenados [Uniform Resource Locator \(URL\)](#), *consumer_key* e *consumer_secret*; estes valores são encriptados em repouso pela aplicação, mantendo os segredos fora do código e do *payload* das respostas.

wooCommerceCredentials	site	client	address
siteId Int (!)	id Int (!)	id Int (!)	id Int (!)
created_at DateTime (!)	created_at DateTime (!)	created_at DateTime (!)	clientId Int (!)
updated_at DateTime	updated_at DateTime	updated_at DateTime	street String (!)
url String (!)	name String (!)	first_name String (!)	city String (!)
consumer_key String (!)		last_name String (!)	state String (!)
consumer_secret String (!)		email String (!)	country String (!)
		phone String	zip_code String (!)
		nif String	
user	order	orderShippingInfo	orderTransaction
id Int (!)	id Int (!)	orderId Int (!)	orderId Int (!)
created_at DateTime (!)	siteId Int (!)	shippingInfo Json	productId Int (!)
email String (!)	created_at DateTime (!)	created_at DateTime (!)	created_at DateTime (!)
password String (!)	updated_at DateTime	updated_at DateTime	updated_at DateTime
passwordChangedAt DateTime	siteOrderId String (!)	shipping_date DateTime	name String (!)
passwordResetToken String	clientId Int (!)	tracking String	sku String (!)
passwordResetExpires DateTime	addressId Int (!)	carrier String	quantity Int (!)
first_name String (!)	status OrderStatus (!)	weight Float	qty_allocated Int (!)
last_name String (!)	payment String (!)	width Float	price Float (!)
phone String	shipping_site String (!)	height Float	taxes Float (!)
role Role (!)	date_site DateTime (!)	length Float	total Float (!)
active Boolean (!)	total Float (!)	label String	total_taxes Float (!)
	coin String (!)	delivered Boolean (!)	
	invoice String		
	priority priority (!)		

Figura 19: Coleção de tabelas — Parte 1 (Utilizadores, canais de venda, clientes e encomendas).

O catálogo de produtos é mantido em **product**, que agrega atributos logísticos e fiscais (peso, *hscode*, país de origem, taxa de imposto e validade) e o estado do *stock* através do enumerado *Stock_status*. Para acomodar variações e referências múltiplas, os identificadores [Stock Keeping Unit \(SKU\)](#) e [European Article Number \(EAN\)](#) são armazenados como *arrays*, permitindo mapear códigos distintos por fornecedor ou por canal. A parametrização específica por loja é tratada em **productSite**, que liga cada *product* a um *site* e concentra campos como *price*, *salePrice*, preços automáticos, *stock* por canal e o identificador externo *siteProductId*. A identidade composta (*siteId*, *productId*) garante consistência entre catálogos e impede duplicações.

A gestão de clientes e moradas é feita pelas tabelas **client** e **address**. Os clientes possuem *email* único e, quando existentes, *phone* e *nif* também únicos; as moradas ficam normalizadas e podem ser reutilizadas por várias encomendas do mesmo cliente. As encomendas são registadas em **order**, que referencia o *site* de origem, o cliente e a morada de envio, e apresenta unicidade composta (*siteId*, *siteOrderId*) para evitar duplicados provenientes das integrações. Cada encomenda evolui por estados definidos no enumerado *OrderStatus* (*processing*, *picking*, *packing*, *completed*, *cancelled*) e incorpora ainda uma prioridade de tratamento. As linhas da encomenda são mantidas em **orderTransaction** — uma relação por produto com identidade (*orderId*, *productId*) — onde se registam quantidades, preços,

supplier		product		invoice		productSite	
id	Int (!)	id	Int (!)	id	Int (!)	siteId	Int (!)
created_at	DateTime (!)	created_at	DateTime (!)	created_at	DateTime (!)	productId	Int (!)
updated_at	DateTime	updated_at	DateTime	updated_at	DateTime	created_at	DateTime
name	String (!)	stock	Int (!)	invoiceId	String	updated_at	DateTime
email	String	stock_virtual	Int (!)	supplierId	Int (!)	stock	Int (!)
phone	String	stock_status	Stock_status (!)	total	Float (!)	price	Float (!)
url	String	sku	String [] (!)	totalPaid	Float	salePrice	Float
productTransactions		ean	String [] (!)	invoiceTransactions		siteProductId	Int (!)
id	Int (!)	name	String (!)	id	Int (!)		
productId	Int (!)	weight	Float (!)	invoiceId	Int (!)		
created_at	DateTime (!)	image	String (!)	created_at	DateTime (!)		
quantity	Int (!)	hscode	String (!)	updated_at	DateTime		
type	TransactionType (!)	country	String (!)	productId	Int (!)		
stock	Int	validity	DateTime	name	String (!)		
supplierId	Int	tax	Float (!)	sku	String (!)		
invoiceId	Int	p_cost	Float	weight	Float (!)		
orderId	Int	fedExCurrency		quantity	Int (!)		
reason	String	country	String (!)	price	Float (!)		
		currency	String (!)	total	Float (!)		
				validity	DateTime		

Figura 20: Coleção de tabelas – Parte 2 (Aprovisionamento, faturas, fornecedores e movimentos de *stock*).

impostos, totais e a quantidade alocada para efeitos de reserva e abatimento de *stock*. A componente logística fica concentrada em ***orderShippingInfo***, uma relação 1:1 com *order* que guarda metadados de expedição (transportadora, serviço, *tracking*, pesos e dimensões), a etiqueta gerada e o indicador de entrega, permitindo sincronizar o ciclo operacional com o estado físico do envio. Para compatibilizar expedições internacionais com os formatos exigidos pela transportadora, a tabela auxiliar ***fedExCurrency*** mapeia países para códigos de moeda.

O circuito de aprovisionamento e o controlo de *stock* são suportados por **supplier**, **invoice** e **invoiceTransactions**, que registam faturas de compra a fornecedores com decomposição por linha (produto, quantidades, pesos e preços), assegurando rastreabilidade nas entradas de inventário. Sobre este núcleo incide **productTransactions**, o livro de movimentos de *stock* que centraliza entradas (*in*) e saídas (*out*) e referencia, quando aplicável, a fatura (*invoiceId*) ou a encomenda (*orderId*); o campo *reason* permite qualificar ajustes manuais e auditorias subsequentes. Este desenho facilita reconciliações e mantém um histórico completo das alterações de inventário ao longo do tempo.

Em termos de integridade e desempenho, o modelo combina chaves primárias simples com identidades compostas nos pontos em que a semântica do negócio assim o exige — casos de *productSite*

e *orderTransaction* — e unicidades funcionais como (*siteId*, *siteOrderId*) em *order*. A aplicação executa operações encadeadas (por exemplo, faturar, lançar movimentos de *stock*, gerar etiqueta e transitar o estado da encomenda) em transações atômicas, prevenindo inconsistências em situações de falha parcial. Para as consultas mais frequentes foram definidos índices nas colunas de maior seletividade, nomeadamente datas e estados de encomendas e pesquisas por identificadores de produto, mantendo o equilíbrio entre rapidez de leitura e custos de escrita. No conjunto, a estrutura garante consistência, simplifica o processamento das encomendas e sustenta, de forma fiável, a operação diária do *backoffice*.

Para complementar a Figura 20, inclui-se a Listagem 4.2 com trechos do *schema.prisma* que explicitam identidades compostas e unicidades.

```

1 model productSite {
2   siteId      Int
3   productId   Int
4   // ...
5   @@id([siteId, productId])
6 }
7 model order {
8   siteId      Int
9   siteOrderId String
10  // ...
11  @@unique([siteId, siteOrderId])
12 }
13 model orderTransaction {
14   orderId     Int
15   productId   Int
16   // ...
17   @@id([orderId, productId])
18 }

```

Listagem 4.2: Trechos essenciais do *schema.prisma* com chaves compostas e unicidades

Por fim, a Figura 21 complementa a análise estrutural da base de dados, apresentando o mapeamento entre os controladores e os respetivos módulos de dados. Esta representação evidencia a separação lógica entre camadas, onde cada *controller* atua sobre um conjunto específico de entidades, assegurando coerência, modularidade e baixo acoplamento. As setas tracejadas identificam interações indiretas, nomeadamente sincronizações executadas pelo *woocommerceController*, enquanto a tabela ***orderShippingInfo*** é atualizada exclusivamente pelos controladores das transportadoras.

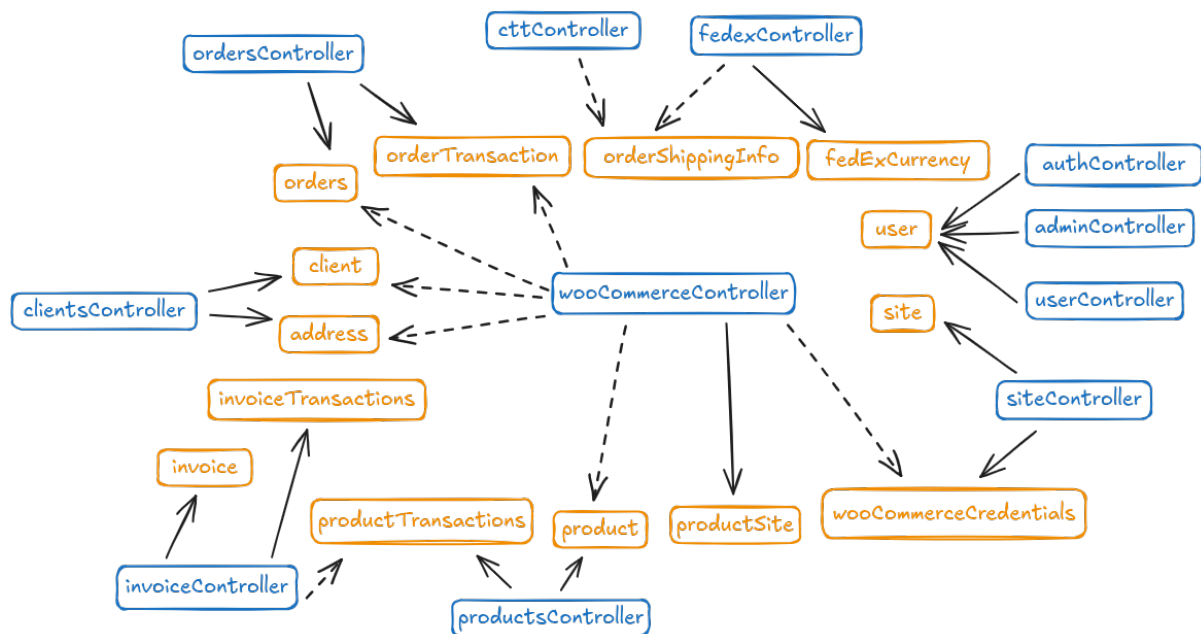


Figura 21: Relação entre controladores e entidades da base de dados.

4.1.4 Métodos Personalizados

A aplicação expõe métodos que vão além do **CRUD** simples, encadeando regras de negócio e integrações externas. A autenticação por **JWT** e o *middleware* de autorização (*protect + restrictTo*) travam o acesso antes de qualquer lógica de domínio, o que garante que só perfis válidos executam operações sensíveis.

4.1.4.1 Endpoints Personalizados

Cada rota é definida no módulo *routes*, delega a lógica no respectivo *controller* e está protegida por *middlewares* de autenticação e autorização. Não existem *endpoints* públicos: todo o acesso requer **JWT** válido. A Tabela 5 resume os *endpoints* personalizados implementados, detalhando o contexto funcional, a **URL** e o método **HTTP** associado.

O *coworker* usa os módulos operacionais (encomendas, expedições, faturação de cliente, clientes e produtos). *admin* e *super_admin* têm acesso total, incluindo administração de utilizadores, gestão de *sites* e integrações *WooCommerce*.

Nos **Carriers**, a criação de envios encadeia controladores numa única chamada: *generateInvoice* → *updateProductsStock* → *createShipment* (CTT Expresso ou FedEx) → *updateOrderStatus*. A emissão de fatura é idempotente: se a encomenda já tiver fatura associada, o *middleware* deteta a condição e salta essa etapa, prosseguindo para a atualização de *stock* e restante cadeia. O fluxo produz etiqueta e *tracking*, persiste metadados em **orderShippingInfo** e promove o estado da *order*. Apenas entram encomendas em *packing*; à saída ficam prontas a fecho. Em caso de falha em qualquer passo, o operador pode repetir a chamada em segurança: a idempotência na faturação e as unicidades do modelo evitam duplicações e regularizam estados, garantindo consistência operacional.

Contexto	URL	HTTP
Carriers — FedEx (criar envio)	/api/v1/carriers/fedex/createShipment	POST
Carriers — FedEx (agendar recolha)	/api/v1/carriers/fedex/schedulePickup	POST
Carriers — CTT Expresso (PA/PT)	/api/v1/carriers/ctt/createShipment/PA/PT	POST
Carriers — CTT Expresso (internacional)	/api/v1/carriers/ctt/createShipment/international	POST
Orders — pesquisa	/api/v1/orders/search	POST
Orders — por estado	/api/v1/orders/status	POST
Orders — <i>picking</i> (alocação de stock)	/api/v1/orders/picking	GET
Orders — alterar estado	/api/v1/orders/:id	PATCH
Orders — cancelar	/api/v1/orders/:id	DELETE
Products — validação (<i>pick</i>)	/api/v1/products/pick	POST
Products — ajuste de stock	/api/v1/products/:id/adjust-stock	PATCH
Sites — atualização em massa (CSV)	/api/v1/sites/:siteId/updater	POST
WooCommerce — encomendas por site	/api/v1/woocommerce/:siteId/orders	GET
WooCommerce — produtos por site	/api/v1/woocommerce/:siteId/products	GET
InvoiceClients — emitir fatura	/api/v1/invoiceClients/	GET

Tabela 5: *Endpoints* aplicativos adicionais para além do [CRUD](#).

Em **Orders**, */search* aceita filtros compostos, */status* devolve listagens por estado e */picking* faz a alocação de *stock* com base na disponibilidade atual. A rota *PATCH /:id* promove a encomenda de *picking* para *packing* (pronta para embalagem/fecho). O *DELETE /:id* permite cancelar sempre que a encomenda não esteja concluída.

No módulo **Products**, */pick* valida leituras de código de barras e devolve os dados mínimos para confirmar se o artigo pertence à encomenda em preparação. */:id/adjust-stock* realiza acertos diretos (defeitos, correções de contagem) sem regras extra; cada ajuste origina um movimento em *productTransactions*, o que garante histórico de inventário.

Em **Sites**, */:siteId/updater* processa ficheiros [CSV](#) com base num *template* do *frontend*. Depois de iniciado, o processo corre até terminar, atualizando preços e quantidades por canal (*productSite*) sem passos intermédios, o que garante alinhamento rápido de catálogos extensos.

As rotas **WooCommerce** */:siteId/orders* e */:siteId/products* sincronizam dados do *site* para o *backoffice* (novas encomendas e novos produtos). A unicidade (*siteId*, *siteOrderId*) torna a operação idempotente e evita duplicados, mantendo a base de dados como fonte única de verdade.

4.1.4.2 Lifecycles

Sem recorrer a *lifecycles* de um [CMS](#), a aplicação implementa ganchos de negócio através de serviços e encadeamentos de controladores, acionados em pontos previsíveis do fluxo. A ideia é simples: cada operação “crítica” valida pré-condições, executa ações atômicas sobre as entidades relevantes e aplica pós-condições consistentes, sempre por trás de [Application Programming Interface \(API\)](#)s autenticadas. Onde faz sentido, recorre-se a *Prisma.\$transaction* para assegurar atomicidade; nos restantes casos, a idempotência prática e a reexecução controlada fazem a correção por sobreposição de informação nova — o que garante um estado coerente mesmo após falhas externas.

No ciclo de expedição (`/api/v1/carriers`), a criação de envios (CTT Expresso e FedEx) é um *lifecycle* composto: apenas encomendas elegíveis (já promovidas para *packing*) podem avançar; a chamada encadeia a emissão da fatura de cliente, a atualização do *stock* de saída, a criação do envio na transportadora — com geração de etiqueta e *tracking* — e, por fim, a transição do estado da *order*. O operador só tem esta ação disponível quando tudo se encontra confirmado a montante; se a criação da *label* falhar, o erro é devolvido de forma explícita para que seja corrigido e a chamada possa ser repetida sem efeitos colaterais indevidos, o que garante consistência entre dados comerciais e logísticos.

As transições de estado da encomenda (`PATCH /api/v1/orders/:id`) implementam o *lifecycle* operativo do armazém. Quando existe *stock* suficiente, a aplicação aloca quantidades à encomenda e promove o estado: de *processing* para *picking* quando a preparação se inicia e, posteriormente, para *packing* quando está pronta a ser fechada e a avançar para expedição. Esta promoção verifica as pré-condições (alocação concluída, inexistência de bloqueios) e impede regressões inválidas. A criação do envio só ocorre a partir de *packing*; a saída bem-sucedida da cadeia deixa a encomenda pronta a fecho.

O cancelamento (`DELETE /api/v1/orders/:id`) é possível enquanto a encomenda não estiver concluída. O *lifecycle* aplica as regras de negócio mínimas (proibição de cancelar *completed*) e desfaz apenas reservas internas que ainda não implicaram movimentos efetivos de *stock* para fora. Como o abatimento real ocorre no fluxo de expedição, cancelar antes desse passo não exige compensações pesadas; se, por algum motivo, for necessário corrigir estados, a estratégia mantida é reexecutar a operação com os dados certos, sobrepondo o anterior — o que garante recuperação simples sem *rollbacks* complexos.

Os acertos de inventário (`PATCH /api/v1/products/:id/adjust-stock`) constituem um *lifecycle* direto: a aplicação regista o ajuste e cria, sempre, um movimento em *productTransactions* identificando a origem (ajuste) e a quantidade. Não existem validações adicionais além das de integridade básica; o objetivo é refletir rapidamente danos, quebras ou correções de contagem, ficando o histórico completo para auditoria futura — o que garante rastreabilidade sobre todas as variações de *stock*.

O *updater* por *CSV* (`POST /api/v1/sites/:siteId/updater`) processa ficheiros segundo um *template* do *frontend* e propaga alterações diretamente para o *WooCommerce* via *API*. Trata-se de um processo sequencial que, depois de iniciado, corre até terminar sem intervenção adicional, atualizando preços e quantidades por canal segundo as instruções do ficheiro. Não há efeitos no esquema interno durante este fluxo; em caso de falhas por linha, o erro é reportado e o operador pode corrigir e repetir, com deteção natural de duplicados no lado remoto — o que garante alinhamento rápido do catálogo do site sem comprometer a base de dados central.

Por fim, a sincronização com *WooCommerce* (`GET /api/v1/woocommerce/:siteId/orders` e `/products`) atua como *lifecycle* de ingestão: os dados recebidos são normalizados e cada registo é criado ou atualizado conforme exista já correspondência. A unicidade (*siteId*, *siteOrderId*) impede duplicações de encomendas e os mapeamentos de produto (*productSite*) evitam entradas redundantes; repetir a chamada não replica informação, limitando-se a aplicar diferenças — o que garante idempotência na integração e mantém a base de dados como fonte única de verdade.

Em operações financeiras — nomeadamente criação e atualização de faturas de compra — a aplicação recorre a *Prisma.\$transaction* para agrupar a escrita da fatura, as linhas em *invoiceTransactions*, a

atualização do *stock* e o lançamento dos respetivos movimentos, tudo no mesmo contexto transacional. Se algum passo falhar, nada é persistido; quando a execução termina, o estado resultante já é o definitivo, o que garante atomicidade e evita inconsistências intermédias.

4.1.4.3 Considerações Adicionais no Backend

Algumas opções de implementação são relevantes para compreender a operação diária do sistema:

- **Etiquetas e faturas em Base64.** As etiquetas de envio e as faturas são persistidas diretamente na base de dados em *Base64*. Esta decisão simplifica o *backend* (sem gestão de armazenamento externo) e mantém o acesso transacional. Como contrapartida, a codificação aumenta o volume armazenado e o tamanho das respostas.
- **Download de documentos.** O *backend* devolve o conteúdo em *Base64* na própria resposta; cabe ao *frontend* criar o [Binary Large Object \(BLOB\)](#) e disponibilizar o ficheiro ao utilizador. Não existem *endpoints* públicos de ficheiros nem *URLs* assinadas.
- **CSV temporários.** Os ficheiros [CSV](#) para importações são processados em memória (validação/pré-visualização/aplicação) e não são guardados.
- **Integrações síncronas.** As chamadas a *CTT Expresso* e *FedEx* são síncronas e sem *retries*; erros indicam condições que requerem intervenção do operador. Na sincronização com *WooCommerce*, um *retry* faria sentido, mas não foi implementado.
- **Consistência de stock.** O *picking* previne o fecho de encomendas sem *stock* alocado, reduzindo o risco de *stock* negativo. Situações negativas apenas poderão resultar de acertos manuais (erro humano).
- **Índices e unicidades.** Estão definidos os *constraints* essenciais: *product.sku* único; *product-Site(productId, siteId)* único; *order(orderNumber, siteId)* único; e índices em *orders(status, createdAt)* e *productTransactions(productId, createdAt)*.
- **Controlo e segurança adicionais.** Não foram aplicados [HTTP Strict Transport Security \(HSTS\)](#) dedicados, *Vary: Origin* no [CORS](#) nem *rate limit* específico para *auth*, para além do limitador geral configurado.

Embora não diretamente relacionadas com a operação diária do sistema, áreas como *contratos de API* (*OpenAPI/Swagger*), *observabilidade* (*healthz, readyz*, métricas) e trilhos de auditoria foram identificadas como potenciais melhorias a considerar em iterações futuras.

Exemplo de consulta unificada. O *endpoint* `GET /api/v1/orders/:id` agrega, numa única chamada, a informação da encomenda e os seus relacionamentos operacionais, conforme a Listagem [4.3](#).

```

1 exports.getOrder = catchAsync(async (req, res, next) => {
2   const id = req.params.id;
3   const order = await prisma.order.findUnique({
4     where: {
5       id: parseInt(id)
6     },
7     include: {
8       client: {
9         select: {first_name: true, last_name: true, email: true, phone: true, nif:
10           true}
11       },
12       address: {
13         select: {street: true, city: true, state: true, zip_code: true, country:
14           true}
15       },
16       site: {
17         select: {name: true}
18       },
19       shipping: true,
20       orderTransaction: {
21         select: {
22           productId: true, name: true,
23           sku: true, quantity: true, qty_allocated: true,
24           taxes: true, price: true, total: true, total_taxes: true,
25           product: {
26             select: {ean: true, weight: true, country: true, hscode: true
27             },},},},},},});
28   if (!order) {
29     return next(new AppError('No order found with that ID', 404));
30   }
31   res.status(200).json({
32     status: 'success',
33     data: {order}
34   });
35 });

```

Listagem 4.3: *Endpoint* de consulta de encomenda com dados operacionais

4.2 Frontend

A camada *frontend* concentra o desenvolvimento da interface de utilizador, funcionando como a ponte entre os operadores e a aplicação. Enquanto o *backend* processa dados e lógica de negócio, o *frontend* apresenta a informação de forma clara e responsiva ao contexto de trabalho. A aplicação foi construída como uma [Single Page Application \(SPA\)](#) em **React**, com *React Router* para navegação, *Material UI* como base de componentes e *Axios* para comunicação com a [API](#). A autenticação recorre a [JWT](#) armazenado em *localStorage*, com rotas protegidas (*guard*) que regulam o acesso conforme o papel do utilizador.

A interface está em inglês, orientada a uso em *desktop* e sem requisitos de internacionalização, adotando o tema *default* do *Material UI*. Esta secção descreve o *backoffice* e a organização do *sitemap*,

bem como as funcionalidades implementadas e as práticas de reutilização de componentes e funções que suportam uma interface eficiente e fácil de manter.

4.2.1 Setup do Frontend (React)

De acordo com as opções técnicas definidas para a interface, o *frontend* foi iniciado com *Create React App* (`npx create-react-app`), resultando numa [SPA](#) em **React** com navegação assegurada por *React Router*. A estrutura do projeto organiza-se em *src/* e *public/*. Em *src/* encontram-se as pastas *components/*, *pages/*, *images/* e *utils/*. A pasta *pages/* contém as vistas de domínio utilizadas pelo *backoffice* (*Home*, *Login*, *Orders*, *OrderPage*, *Products*, *ProductPage*, *Clients*, *ClientPage*, *Invoices*, *InvoiceCreatePage*, *InvoiceUpdatePage*, *SuppliersPage*, *Sites*, *UsersManagement*, *AdminPage*, *ProfilePage*, *ForbiddenPage*), enquanto *components/* agrega componentes reutilizáveis como *Layout*, *Navbar*, *ProtectedRoute* e os formulários/labels específicos dos *carriers* (*CTTExpressoForm*, *CTTExpressoLabel*, *FedExForm*, *FedExLabel*, *OrderDetails*, *ProductList*, *ShippingCompanySelect*). Em *public/* residem os recursos estáticos (*index.html*, *favicon*, ícones e *manifest*). A biblioteca *Material UI* fornece a base de componentes e é usada com o tema *default*; estilos pontuais são aplicados com *styled* do próprio *Material UI* (por exemplo, contentores e cartões com espaçamentos do tema).

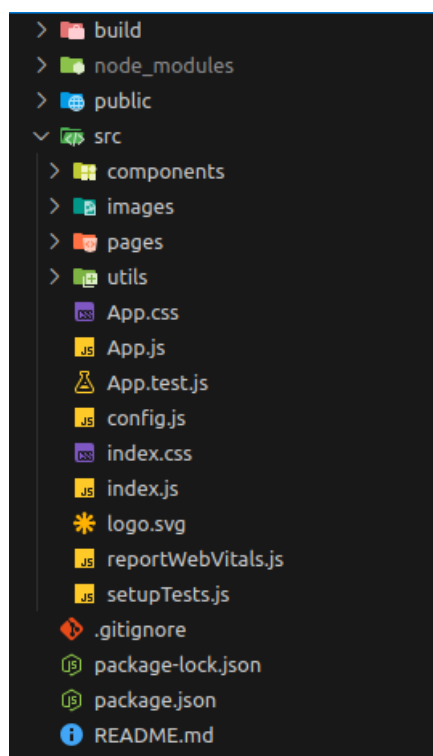
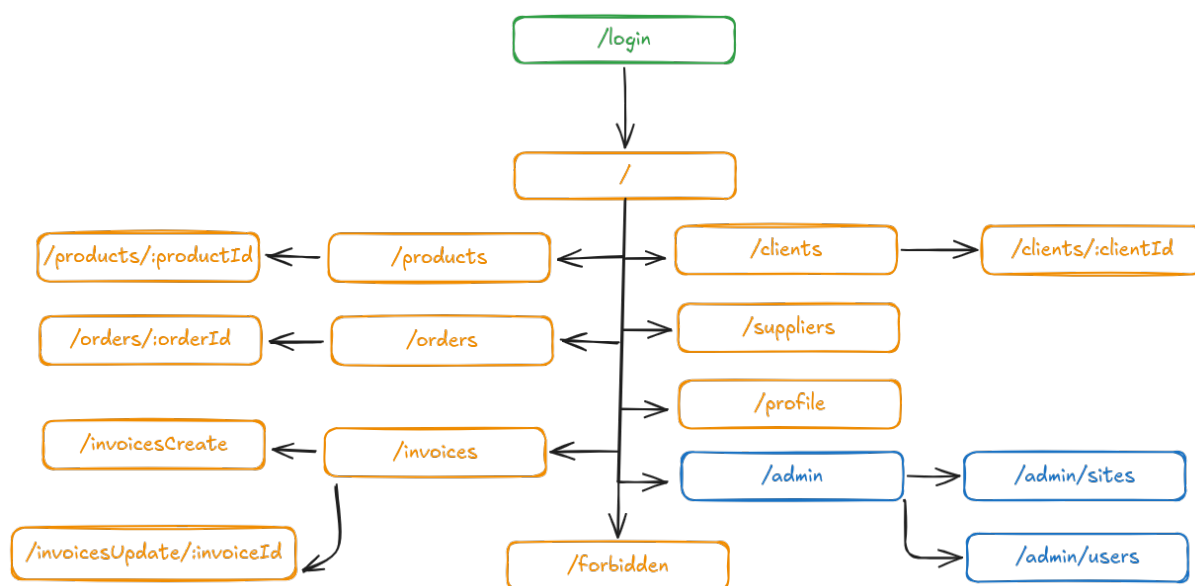


Figura 22: Estrutura de pastas do *frontend*.



Perfis: verde — acesso público (User); laranja — acesso autenticado (Coworker, Admin, Super-Admin); azul — acesso restrito a Admin e Super-Admin.

Figura 23: Mapa de rotas do *frontend* e distribuição de acessos.

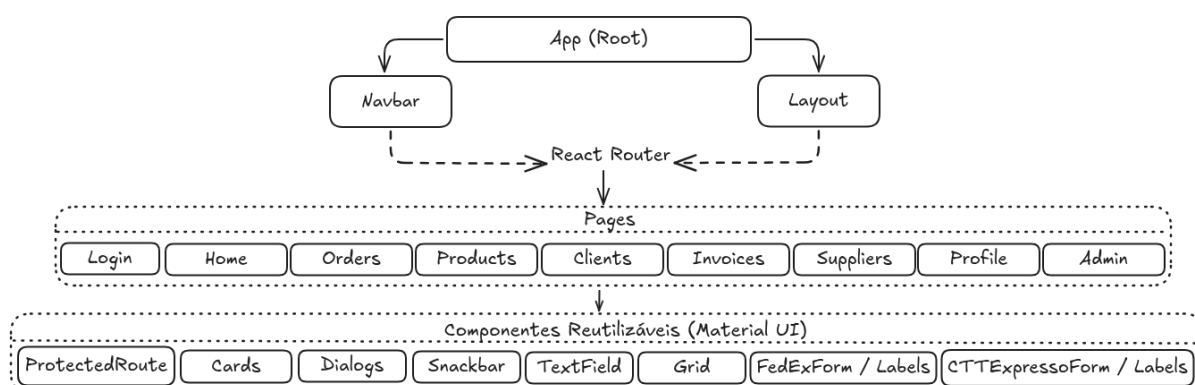


Figura 24: Arquitetura funcional do *frontend* com fluxo de componentes, rotas e elementos reutilizáveis.

A definição das rotas é feita de forma declarativa, combinando caminhos estáticos e dinâmicos, conforme ilustrado nas Figuras 23 e 24. As rotas principais incluem / (página inicial autenticada) e /login (pública), bem como as páginas operacionais e administrativas: /orders e /orders/:orderId, /products e /products/:productId, /clients e /clients/:clientId, /invoices, /invoicesCreate, /invoicesUpdate/:invoiceId, /suppliers, /profile, /admin, /admin/sites, /admin/users e /forbidden. O projeto utiliza React Router v6 com camada de compatibilidade (react-router-dom-v5-compat). As rotas dinâmicas recebem o identificador via URL e usam-no para obter, através da API, os dados específicos do registo a apresentar no detalhe.

A autenticação é baseada em JWT armazenado em *localStorage*. No arranque da aplicação é lido o token e o objeto de utilizador (userData, contendo id, email, name e role); quando presentes, é estabelecida a sessão local. Um interceptor global do Axios injeta o cabeçalho Authorization: Bearer com o JWT

em todas as chamadas à API. A base de URL é definida em *src/config.js* (por omissão, */api/*, com alternativa comentada para desenvolvimento). O componente *ProtectedRoute* valida a existência de sessão e, quando aplicável, o conjunto de papéis exigido (por exemplo, *roles = [admin, super_admin]*); acessos não autorizados são encaminhados para */forbidden*. O *Layout* comum envolve as páginas autenticadas e fornece navegação, cabeçalho e *feedback* com *Snackbar/Alert* do *Material UI*.

A interface é *desktop-only*, em inglês, e não contempla internacionalização nem responsividade móvel nesta versão. A combinação de *React*, *React Router*, *Material UI* e *Axios*, aliada à organização modular em *pages* e *components* e à centralização da configuração em *config.js*, resulta numa **User Interface (UI)** previsível, eficiente e fácil de manter, adequada ao contexto operacional do *backoffice*.

4.2.2 Frontend Sitemap

O *frontend* é a camada onde os diferentes perfis interagem com o sistema. As Figuras 23 e 24 sintetizam, respetivamente, o mapeamento das rotas da SPA e a arquitetura funcional do *frontend*, evidenciando a organização das rotas, o *layout* persistente e a ligação aos componentes reutilizáveis. O ponto de entrada é */login*; após autenticação, o utilizador é encaminhado para */* (Home) e, a partir daí, navega pelas áreas operacionais e administrativas de acordo com o seu papel. As rotas dinâmicas (por exemplo, */orders/:orderId* ou */products/:productId*) são habitualmente acedidas a partir das respetivas listagens, garantindo uma transição natural entre consulta e detalhe na UI. A distribuição de acessos entre perfis — *User* (público), *Coworker* (operador), *Admin* e *Super-Admin* — encontra-se também representada graficamente, permitindo uma leitura imediata das restrições de navegação e da hierarquia funcional das rotas.

4.2.3 Estrutura Base da Interface

A estrutura base da aplicação é implementada por um componente *Layout* (*src/pages/layout/Layout.js*), que atua como *shell* principal da UI para todas as páginas autenticadas. A Figura 25 ilustra esta estrutura, composta pela *Navbar* no topo e pela zona de conteúdo (*{children}*) onde são injetadas as páginas do *backoffice*. O *Layout* assegura uma moldura visual consistente em toda a interface e envolve o conjunto de **páginas autenticadas**, conforme representado na Figura 24.

A *Navbar* (*src/pages/layout/Navbar.js*) é construída com componentes do *Material UI* (*AppBar* e *Toolbar*) e apresenta a marca da aplicação à esquerda e o menu de conta à direita, com saudação e opções de *Profile* e *Logout*. O *Layout* recebe as *props* *userData* (*{ id, email, name, role }*) e *logoutHandler*, propagando-as à *Navbar* para centralizar o comportamento de sessão. A página pública */login* encontra-se fora do *Layout*, enquanto */forbidden* reutiliza o mesmo esqueleto visual, garantindo consistência entre as vistas autenticadas e as de erro.

O controlo de acesso é aplicado ao nível das rotas através do componente *ProtectedRoute*, que valida a existência do JWT em *localStorage* e, quando exigido, o papel do utilizador (por exemplo, *admin* ou *super_admin*); acessos não autorizados são redirecionados para */forbidden*. A *Navbar* não filtra elementos por papel, ficando a responsabilidade de autorização concentrada no *guard* de rota. O tema do *Material*

UI é aplicado globalmente e herdado pelo *Layout*; estilos pontuais são definidos com *styled* (por exemplo, ajuste de *z-index* no *AppBar*) para garantir sobreposição correta e espaçamentos coerentes. Por se tratar de uma *SPA* criada com *Create React App*, todo o *render* do *Layout* ocorre no cliente, enquanto metadados e recursos estáticos (ícones e *manifest*) residem em *public/*. Esta abordagem centraliza a navegação, o *shell* visual e o ciclo de sessão num único ponto, reduzindo a complexidade e promovendo uma experiência consistente ao longo das diferentes páginas.

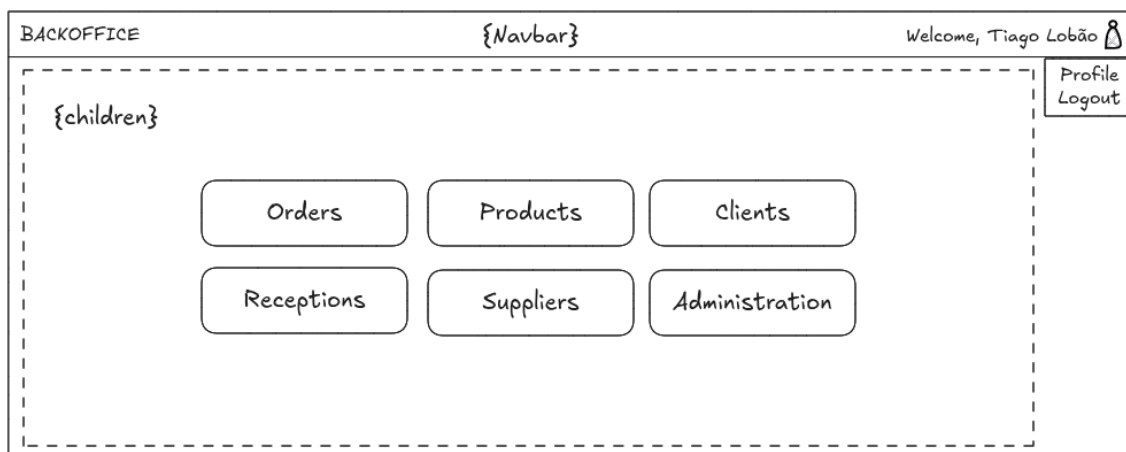


Figura 25: Esquema simplificado do componente *Layout* e da sua integração com a *Navbar* e as páginas autenticadas.

4.2.4 Componentes Reutilizáveis

A aplicação recorre extensivamente à reutilização de componentes para acelerar o desenvolvimento e manter a interface (*UI*) coesa. A abordagem privilegia a composição com *Material UI* — *Card*, *Table*, *Dialog*, *TextField*, *Select*, *Snackbar/Alert*, entre outros — e pequenas extensões com *styled* do próprio *Material UI*. Este padrão permite centralizar comportamentos e estilos, reduzir código duplicado e aplicar correções de forma transversal.

4.2.4.1 Cartões (Cards)

Os *cards* são utilizados como unidades visuais reutilizáveis para apresentar ações e resumos nas páginas principais (por exemplo, a “grelha” do menu inicial do *backoffice*). Cada cartão agrega ícone, título e breve descrição, encaminhando o utilizador para áreas como *Orders*, *Products*, *Clients* ou *Invoices*. Construídos com *Card*, *CardContent* e *Typography*, partilham espaçamentos do tema e um *wrapper* estilizado, garantindo consistência entre páginas e facilitando a adição de novas entradas sem esforço.

4.2.4.2 Tabelas e Listas

As listagens de dados reutilizam um conjunto de elementos do *Material UI* — *Table*, *TableHead*, *TableRow*, *TableCell*, *TableBody* e *TableContainer* — com cabeçalhos uniformes, formatação de colunas e células de ação. O mesmo esqueleto é aplicado em *Orders*, *Products*, *Clients* e *Invoices*, alterando

apenas as colunas e os carregadores de dados. Este reaproveitamento torna as páginas previsíveis para o utilizador e simplifica a manutenção.

4.2.4.3 Diálogos e Confirmações

A confirmação de ações sensíveis (por exemplo, emissão de fatura, geração de etiqueta, alterações de estado) é feita com um *Dialog* padrão (*Dialog*, *DialogTitle*, *DialogContent*, *DialogActions*), recebendo por *props* o texto, as ações e o estado de carregamento. O mesmo padrão de diálogo é reutilizado em múltiplas páginas, evitando implementações ad hoc e mantendo a experiência homogénea.

4.2.4.4 Formulários e Controlos

Os formulários baseiam-se nos controlos do *Material UI* — *TextField*, *Select*, *FormControl*, *InputLabel*, *MenuItem*, *FormHelperText*, *Tooltip*, *Chip* — aplicando validações simples e mensagens de erro consistentes. O mesmo conjunto de entradas é reutilizado em *Products*, *Invoices* (criação e atualização) e fluxos de expedição (por exemplo, seleção de transportadora), o que reduz a curva de aprendizagem e facilita a evolução do esquema de dados exposto pela [API](#).

4.2.4.5 Notificações e Feedback

O *feedback* de operação é padronizado com *Snackbar* e *Alert*, utilizados de forma uniforme em todas as páginas para reportar sucesso, aviso e erro. A decisão de reutilizar os componentes nativos do *Material UI* evita dependências adicionais e garante acessibilidade e comportamentos consistentes (tempo de exibição, cores e ícones).

4.2.4.6 Componentes Operacionais Específicos

Alguns componentes encapsulam interações frequentes com o domínio do sistema, mas continuam a ser compostos apenas com *Material UI*. Entre eles destacam-se *OrderDetails* (resumo de encomenda), *ProductList* (listagem reutilizada em diferentes ecrãs) e *ShippingCompanySelect* (seleção de transportadora para criação de envio). Ao concentrar lógica de apresentação nestes componentes, o código das páginas permanece focado no fluxo e na orquestração de chamadas à [API](#).

Em conjunto, estes padrões de reutilização — baseados exclusivamente em componentes do *Material UI* e pequenas extensões estilísticas — permitem escalar a [UI](#) com menor esforço, manter uma experiência previsível entre ecrãs e aplicar alterações de forma transversal e controlada.

4.2.5 Funções e Métodos

As funções e métodos do *frontend* organizam a lógica de comunicação com a [API](#) e a transformação dos dados antes de chegarem à [UI](#), promovendo modularidade e reutilização. A comunicação é centralizada em utilitários de *Axios*, com a base [URL](#) definida em *src/config.js* (por omissão, */api/*). No arranque da aplicação, é registado um *interceptor* de pedidos que injeta o cabeçalho *Authorization: Bearer* com o

[JWT](#) persistido em *localStorage*; desta forma, todas as páginas e componentes que consomem serviços passam a autenticar-se de forma consistente, sem duplicar código.

Para leitura de dados (*GET*), as funções de serviço recebem os parâmetros de rota (por exemplo, *orderId* em */orders/:orderId*) e devolvem objetos JavaScript prontos a renderizar nos componentes (tabelas, cartões, formulários). Como o projeto não utiliza *TypeScript*, a validação leve é feita no próprio componente (existência de campos obrigatórios, formatos básicos) e os erros são comunicados ao utilizador via *Snackbar/Alert*. Para operações de escrita (*POST/PATCH/DELETE*), as funções estruturam o *payload* de acordo com o modelo exposto pela [API](#) do *backend* (por exemplo, alteração de estado da encomenda, criação/atualização de *invoice* de compra, geração de etiqueta), mantendo o [JWT](#) no cabeçalho para verificação de permissões no servidor.

A figura dedicada às sequências de interação (consultas e operações) ilustra estes dois fluxos típicos: (i) obtenção de dados, em que a página solicita a informação ao serviço, que por sua vez chama a [API](#) e devolve o resultado normalizado para a [UI](#); e (ii) criação/atualização, em que o componente prepara o *payload*, chama o serviço e apresenta o *feedback* de sucesso/erro. Quando os dados recebidos incluem conteúdos em *Base64* (por exemplo, etiquetas de envio ou faturas), o *frontend* trata-os no componente de destino, criando o [BLOB](#) para pré-visualização ou *download*, sem depender de ficheiros estáticos.

A Listagem 4.4 mostra o registo do *interceptor* global de *Axios* responsável por anexar o [JWT](#) em todas as chamadas à [API](#), garantindo um ponto único de aplicação da política de autenticação no *frontend*.

```
1 componentDidMount() {
2   const hasToken = localStorage.getItem('token');
3   const userData = JSON.parse(localStorage.getItem('userData'));
4
5   if (hasToken && userData) {
6     this.setState({ isLoggedIn: true, token: hasToken, userData, isCheckingAuth:
7       false });
8   } else {
9     this.setState({ isCheckingAuth: false });
10  }
11
12  axios.interceptors.request.use((config) => {
13    const token = localStorage.getItem('token');
14    if (token) config.headers.Authorization = `Bearer ${token}`;
15    return config;
16  });
17 }
```

Listagem 4.4: Interceptor global do *Axios* com inclusão do [JWT](#)

Em síntese, a combinação de serviços finos para acesso à [API](#), o *interceptor* de autenticação por [JWT](#) e a apresentação com componentes do *Material UI* permite manter o código limpo e previsível: as páginas focam-se no fluxo e no *feedback* ao utilizador, enquanto a lógica de comunicação e tratamento de dados fica centralizada e reutilizável.

4.3 Deploy e Operação

Esta secção descreve a implementação em produção do sistema, incluindo infraestrutura, topologia de rede e domínio, *reverse proxy*/[Transport Layer Security \(TLS\)](#), *backend* e *frontend*, base de dados e rotinas operacionais.

4.3.1 Infraestrutura e Sistema Operativo

A plataforma corre num *Droplet* da DigitalOcean (*VPS*) com 25 GB de disco e 1 GB de *RAM*, sobre Ubuntu 22.04. A configuração de *timezone* do servidor não foi alterada explicitamente, mantendo-se a predefinição do sistema.

4.3.2 Topologia e Domínio

O domínio de produção é *lobao-backoffice.pt*, com registos [Domain Name System \(DNS\)](#) geridos via Cloudflare e apontados para o IP público do servidor. O [CORS](#) do *backend* utiliza uma *allowlist* que inclui o domínio de produção e origens de desenvolvimento (*localhost*/127.0.0.1 nas portas usadas), garantindo que apenas estas origens podem invocar a [API](#).

4.3.3 Backend em Produção

O *backend* (*Node.js*/*Express*) corre sob *pm2* na porta interna 3000. As variáveis de ambiente residem num ficheiro *.env* com permissões restritas (600). No momento de *deploy*, são aplicadas as migrações com *Prisma* (*npx prisma migrate deploy*). As etiquetas e faturas são armazenadas em *Base64* na base de dados (não há dependência de *filesystem* para binários), mantendo a aplicação sem estado no disco.

4.3.4 Frontend em Produção

O *frontend* (*React*, [SPA](#)) é compilado com *npm run build* e servido como conteúdo estático pelo *Nginx*. O *root* do *site* aponta para o diretório do *build* (p. ex., */var/www/html/react*). O *fallback* para [SPA](#) encontra-se ativo através de *try_files*, garantindo que rotas internas (p. ex., */orders/:orderId*) são corretamente resolvidas pelo *index.html*.

4.3.5 Reverse Proxy e TLS

O *Nginx* atua como *reverse proxy*, terminando [TLS](#) (certificados *Let's Encrypt*/*certbot* com renovação automática) e encaminhando: (i) pedidos ao *frontend* estático e (ii) pedidos */api/* para o *backend* em *localhost:3000*. O redirecionamento de [HTTP](#)→[Hypertext Transfer Protocol Secure \(HTTPS\)](#) está ativo. Não foram adicionados cabeçalhos de endurecimento específicos (p. ex., [HSTS](#)) nesta versão, conforme demonstrado na Listagem 4.5.

```
1 # Redirecionamento HTTP para HTTPS
2 server {
```

```

3     listen 80;
4     server_name lobao-backoffice.pt www.lobao-backoffice.pt;
5
6     # Redirecionar todo o tráfego HTTP para HTTPS
7     return 301 https://$host$request_uri;
8 }
9
10 # Servindo conteúdo com HTTPS
11 server {
12     listen 443 ssl;
13     server_name lobao-backoffice.pt www.lobao-backoffice.pt;
14
15     # Certificados SSL
16     ssl_certificate /etc/letsencrypt/live/lobao-backoffice.pt/fullchain.pem;
17     ssl_certificate_key /etc/letsencrypt/live/lobao-backoffice.pt/privkey.pem;
18     include /etc/letsencrypt/options-ssl-nginx.conf;
19     ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
20
21     # Configuração do Backoffice (React)
22     location / {
23         root /var/www/html/react; # Caminho onde está o build do React
24         index index.html;
25         try_files $uri $uri/ /index.html; # Garante que todas as requisições vão
26         para index.html
27     }
28
29     # Configuração da API (Node.js no localhost:3000)
30     location /api/ {
31         rewrite ^/api(/.*)$ $1 break; # Remove o primeiro /api da
32
33         proxy_pass http://localhost:3000;
34         proxy_http_version 1.1;
35         proxy_set_header Upgrade $http_upgrade;
36         proxy_set_header Connection 'upgrade';
37         proxy_set_header Host $host;
38         proxy_cache_bypass $http_upgrade;
39     }
40 }

```

Listagem 4.5: Bloco Nginx de produção (*frontend* estático e proxy da API)

4.3.6 Base de Dados

O PostgreSQL corre no mesmo servidor (porta 5432, acesso local). Não existem *backups* automatizados configurados; este ponto é assumido como trabalho futuro.

4.3.7 Operação

A gestão de processos é feita com *pm2* (*pm2 status*, *pm2 reload*). A observabilidade é mínima nesta fase: os *logs* são consultados via *pm2 logs*. O *deploy* é manual (ligação [Secure Shell Protocol \(SSH\)](#), atualização de código, *build* do *frontend*, migrações e *reload* do *backend* com *pm2*).

4.3.8 Segurança

A *firewall* expõe apenas as portas necessárias (22/80/443). O [SSH](#) utiliza chave pública (com *passphrase*). Os serviços correm sob utilizador não-root. Não foi configurado *fail2ban* nem cabeçalhos [HSTS](#) dedicados nesta versão.

4.3.9 Notas e Melhorias Futuras

Entre as melhorias planeadas destacam-se: *backups* regulares da base de dados (p. ex., *pg_dump* agendado), cabeçalhos de segurança adicionais no *Nginx* ([HSTS](#)), monitorização básica (estado do processo, consumo de recursos) e [Continuous Integration/Continuous Deployment \(CI/CD\)](#) simples para reduzir passos manuais no *deploy*.

Análise e Testes

Este capítulo apresenta uma avaliação global das principais funcionalidades do *backoffice*. Combina a leitura funcional das páginas e fluxos com análise técnica automatizada (desempenho, acessibilidade e boas práticas), oferecendo uma visão equilibrada do comportamento e qualidade do sistema.

5.1 Análise de Funcionalidades

Esta secção apresenta uma visão geral das principais funcionalidades do *backoffice*, descrevendo como cada ecrã suporta os fluxos operacionais do dia a dia. O foco recai nos ecrãs *Login* (ponto de entrada), *Home* (menu), *Encomendas* (listagem, detalhe e ações de negócio), *Produtos*, *Inventário/Compras*, *Clientes*, *Fornecedores*, *Admin* e *Perfil*. As funcionalidades interligam-se através da [Representational State Transfer Application Programming Interface \(REST API\)](#) do *backend*, com autenticação por [JSON Web Token \(JWT\)](#), o que garante uma experiência consistente em [Single Page Application \(SPA\)](#) *desktop-only* baseada em *Material UI*. Destaca-se, em particular, que a expedição (*CTT Expresso/FedEx*) é executada a partir do detalhe da encomenda, concentrando contexto e reduzindo passos manuais.

5.1.1 Login

A página de *login* é o primeiro ponto de contacto com o sistema e funciona também como *landing page*. O ecrã foi desenhado com foco na simplicidade, para que o acesso às áreas protegidas seja rápido e claro: o utilizador introduz o e-mail e a palavra-passe e recebe uma indicação imediata de sucesso ou erro.

No arranque, a aplicação verifica se existe sessão previamente guardada; quando há dados válidos, o utilizador é encaminhado automaticamente para a *Home*, evitando passos desnecessários. Caso contrário, permanece na página de *login*.

Após a submissão do formulário, as credenciais são validadas no *backend*. Em caso de sucesso, a sessão é guardada localmente para manter o acesso entre visitas; em caso de falha, é apresentada uma mensagem simples e objetiva. Não existe página pública de registo: a criação de contas é realizada pelos

administradores. A Figura 26 apresenta o ecrã de *login*, destacando o formulário simples e o fluxo direto de autenticação do utilizador.

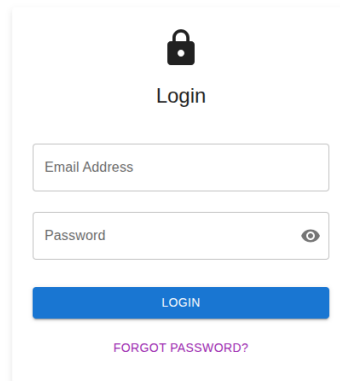
The image shows a login screen with a white background and a light gray border. At the top center is a black padlock icon. Below it, the word "Login" is centered in a bold, black font. There are two input fields: the first is labeled "Email Address" and the second is labeled "Password" with a small eye icon to its right. Below the password field is a blue button with the text "LOGIN" in white. At the bottom, there is a link that says "FORGOT PASSWORD?" in a purple font.

Figura 26: Ecrã de *ogin*

5.1.2 Home

A *Home* funciona como o menu principal do *backoffice*. Em grelha simples, cada cartão apresenta imagem, título e um botão de ação, encaminhando rapidamente para as áreas de trabalho. Estão visíveis: *Orders*, *Products*, *Clients*, *Receptions*, *Suppliers* e *Administration*. A página privilegia a leitura imediata e reduz cliques desnecessários, servindo de ponto de partida para as tarefas diárias. A Figura 27 mostra a organização visual desta página, com os cartões que dão acesso às principais secções.

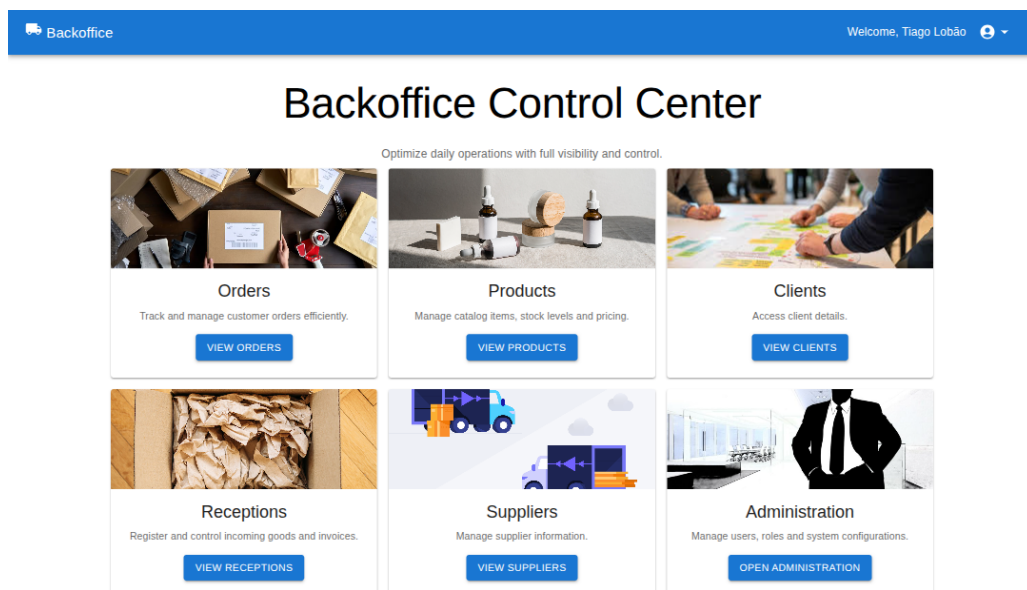


Figura 27: *Home* do *backoffice* com cartões de acesso: *Orders*, *Products*, *Clients*, *Receptions*, *Suppliers* e *Administration*.

5.1.3 Encomendas

A página de *Encomendas* é o centro do trabalho diário. A listagem apresenta as encomendas em tabela, com pesquisa rápida e filtros por estado e data, permitindo localizar de imediato o pedido certo e avançar para a ação. A leitura é direta (número da encomenda, cliente, origem, estado e data), e a navegação para o detalhe é feita num único clique. A Figura 28 apresenta a listagem de encomendas, evidenciando as opções de pesquisa e filtros que simplificam a localização de pedidos específicos.

No detalhe, toda a informação relevante está reunida no mesmo ecrã: dados do cliente e morada, artigos da encomenda (com referências e quantidades). Neste ecrã executam-se também as ações principais: emitir fatura, criar o envio (CTT Expresso ou FedEx) e atualizar o estado. Após a expedição, a etiqueta e o código de *tracking* ficam disponíveis para consulta e *download*. Apenas as encomendas elegíveis apresentam as ações ativas (por exemplo, quando o *stock* já foi alocado). Em caso de falha, o operador pode repetir a ação; o sistema evita duplicações e mantém a informação consistente. A Figura 29 ilustra o ecrã de detalhe da encomenda, reunindo toda a informação relevante e as ações de negócio, como faturação e expedição.

Order ID	Client	Total Price	Status	Articles	Actions
4	Tiago Lobão	45.99 EUR	completed		No tracking available

Figura 28: Lista de encomendas com pesquisa e filtros.

5.1.4 Produtos

A página de *Produtos* reúne o catálogo da loja num formato simples de tabela, com pesquisa por nome ou [Stock Keeping Unit \(SKU\)](#) e filtros que ajudam a localizar rapidamente o artigo certo. A partir da lista é possível abrir o detalhe de cada produto para consultar e editar a informação principal — identificação, preços, impostos e *stock*. A Figura 30 ilustra esta listagem e o modo como os produtos são organizados e geridos no sistema.

No detalhe, os campos são apresentados de forma organizada para facilitar pequenas correções do dia a dia (por exemplo, ajustar preço ou ativar/desativar um artigo). As alterações guardadas ficam imediatamente disponíveis para as restantes áreas do *backoffice*. As operações de atualização em massa são tratadas separadamente na funcionalidade de importação [Comma-Separated Values \(CSV\)](#). A Figura 31 ilustra o ecrã de detalhe do produto.

Backoffice

Welcome, Tiago Lobão

Order Details

Order Status:
packing

Payment Method:
Credit / Debit Card

Shipping Info
Address: Rua Rebelo de Carvalho, n°252, Santão
City: Lixa, **Zip Code:** 4615-469
Country: PT

Name:
Miguel Fonseca

Email:
miguelcfonseca2008@gmail.com

Phone:
962859363

Products

Scan SKU or EAN

Figura 29: Detalhe da encomenda: informação consolidada e ações (fatura, expedição, estado).

Backoffice

Welcome, Tiago Lobão

Search Products

ID	Name	Price	Description
59	SYL - White Tee - Size: L		
60	SYL - White Tee - Size: XL		
48	Crochet Bucket Hat - Color: Purple		
46	Crochet Bucket Hat - Color: Blue		
47	Crochet Bucket Hat - Color: Multicolor		
49	SYL - Black Tee With Purple Detail - Size: S		
50	SYL - Black Tee With Purple Detail - Size: M		
51	SYL - Black Tee With Purple Detail - Size: L		
52	SYL - Black Tee With Purple Detail - Size: XL		
53	SYL - Black Tee With Blue Detail - Size: S		

Figura 30: Lista de produtos com pesquisa e filtros.

Backoffice

Welcome, Tiago Lobão

[← BACK TO PRODUCTS](#)



SYL - White Tee - Size: L

Product Details

EAN:

SKU:

syl-tee-l

Price: €

Stock: 0 [ADJUST](#)

Weight: 0 g

Tax: 23%

Validity: N/A

Stock Status: instock

HS Code: N/A

Country: N/A

[EDIT](#)

Site Information

Site	Price	Sale Price	Stock
Gramhery	40 €	40 €	3

Figura 31: Detalhe do produto com edição dos campos principais.

5.1.5 Fornecedores

A área de *Fornecedores* é simples e direta: apresenta apenas uma tabela com os registos e um *popup* para edição. A listagem mostra as colunas essenciais (identificação e contactos) e, ao seleccionar um fornecedor, abre-se um diálogo com os campos principais para ajustar dados e guardar as alterações.

A Figura 32 apresenta a tabela de fornecedores com a informação essencial organizada de forma compacta. Já a Figura 33 ilustra o *popup* de edição, que permite alterar rapidamente os dados sem sair da página principal.

Este formato evita navegação adicional e acelera pequenas correções do dia a dia.

Backoffice

Welcome, Tiago Lobão

Suppliers

[+ ADD SUPPLIER](#)

ID	Name	Email	Phone	Actions
1	Camisolas	teste@teste.com	910000000	
2	Chapéus Lda.	chapeu@chapeu.com	910000001	

Figura 32: Tabela de fornecedores.

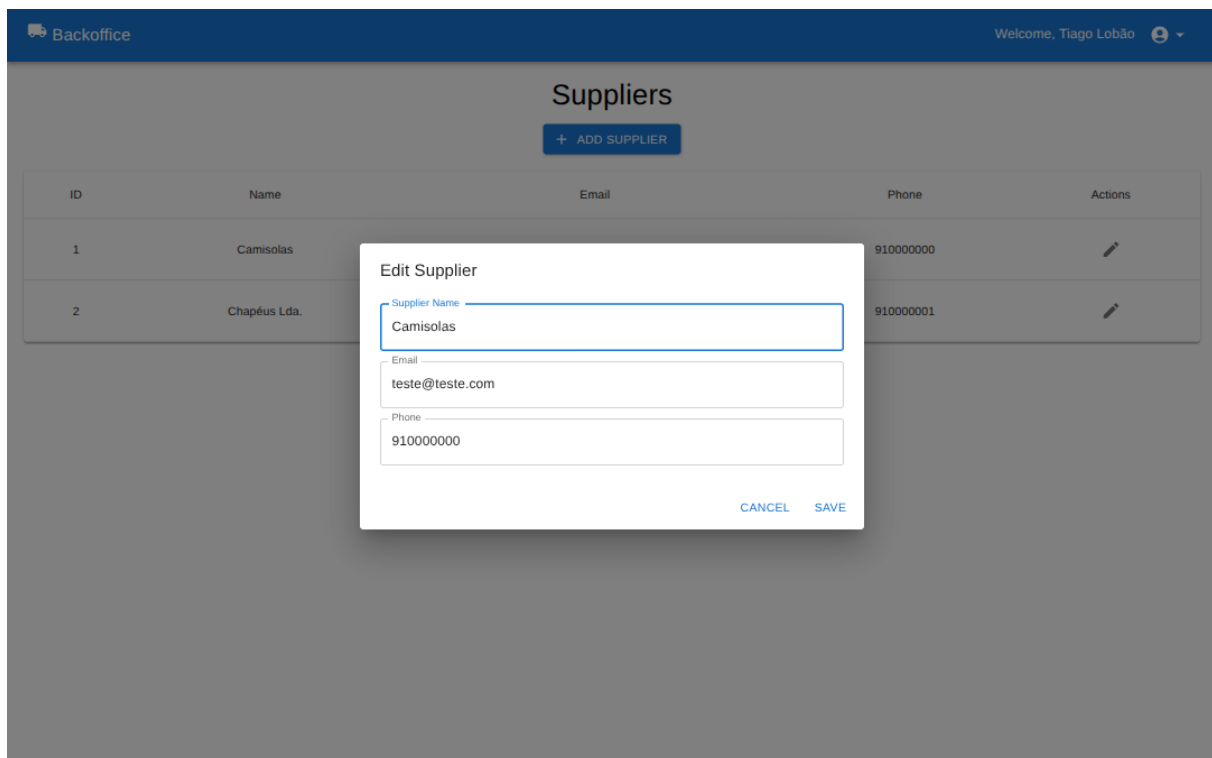


Figura 33: *Popup* de edição de fornecedor.

5.1.6 Receções (Inventário/Compras)

A área de *Receções* regista as entradas de mercadoria e as respetivas faturas de fornecedor. A lista apresenta os documentos já registados — fornecedor, número, data e total — com pesquisa simples e filtro por período, facilitando a consulta do histórico. A Figura 34 mostra a vista de lista em *desktop*, onde se apresentam as faturas de fornecedor com opções de pesquisa.

Na criação/edição, o fluxo é direto: escolhe-se o fornecedor, introduz-se o número e a data da fatura e acrescentam-se linhas com os artigos recebidos (procura por nome ou *SKU*), quantidades e preço de compra. Ao confirmar, o *stock* dos produtos é atualizado e a receção fica registada para referência futura. O objetivo é tornar a reposição rápida e rastreável, mantendo a informação organizada num só ecrã. A Figura 35 ilustra o formulário de receção, destacando o fluxo de introdução de dados do fornecedor e das linhas de artigos recebidos.

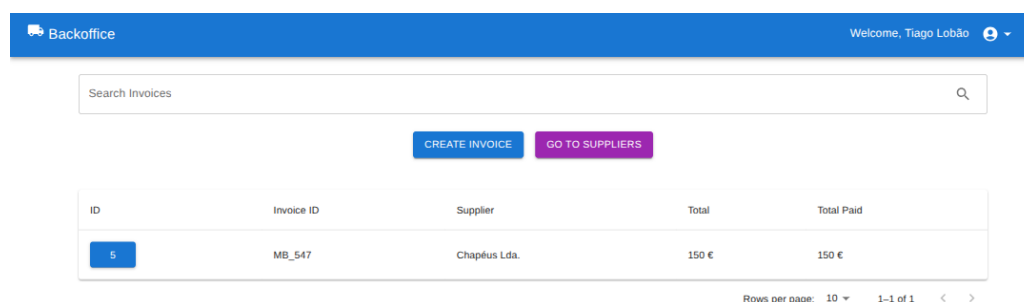


Figura 34: Lista de receções (faturas de fornecedor) com pesquisa e filtros.

Invoice Update

Invoice ID: 5

Total Paid: 150

Supplier: Chapéus Lda.

Product Information

SKU: cbh-purple

EAN:

Search Product (EAN/SKU): cbh-purple

Product	SKU	Price	Quantity	Total	Validity
Crochet Bucket Hat - Color: Purple	cbh-purple	10	5	50 €	mm/dd/yyyy
Crochet Bucket Hat - Color: Blue	cbh-blue	10	5	50 €	mm/dd/yyyy
Crochet Bucket Hat - Color: Multicolor	cbh-multi	10	5	50 €	mm/dd/yyyy
Total:				150 €	

Figura 35: Criação/edição de receção: seleção de fornecedor e linhas de artigos.

5.1.7 Clientes

A área de *Clientes* reúne os registos de clientes numa tabela simples, com pesquisa por nome ou e-mail para chegar rapidamente ao contacto certo. A partir da lista abre-se o detalhe, onde estão os dados principais e um resumo do histórico (encomendas e moradas), facilitando a navegação direta para cada encomenda quando é preciso esclarecer alguma questão. A Figura 36 mostra a vista de lista em *desktop*, apresentando os clientes registados e permitindo pesquisa rápida por nome ou e-mail.

No detalhe, a informação é apresentada de forma limpa para apoiar tarefas do dia a dia: confirmar contactos, rever moradas e abrir encomendas recentes num clique. O objetivo é reduzir passos quando é necessário responder ao cliente ou conferir informação antes de avançar com uma ação na encomenda. A Figura 37 ilustra o ecrã de detalhe do cliente, reunindo contactos e moradas.

Search Clients

ID	First Name	Last Name	Email	Phone	NIF
3	Tiago	Lobão	tiobao44@teste.com	321456789	
2	Tiago	Ferreira	tiago.babo@teste.com	123456987	
1	Miguel	Fonseca	miguelcfonseca@teste.com	325641789	

Rows per page: 10 1-3 of 3

Figura 36: Lista de clientes com pesquisa.

Backoffice Welcome, Tiago Lobão

[← BACK TO CLIENTS](#)

Tiago Lobão

Client Details

First Name: Tiago
Last Name: Lobão
Email: tlobao44@teste.com
Phone: 321456789
NIF:

[EDIT](#)

Client Addresses

[+ ADD ADDRESS](#)

Street	City	State	ZIP Code	Country	Actions
Rua do Lobão 665	Barrosas, Idães		4516-874	PT	✎ 🗑

Rows per page: 5 1-1 of 1 < >

Figura 37: Detalhe do cliente com contactos e moradas

5.1.8 Administração

A área de *Administração* é reservada a *Admin* e *Super-Admin* e concentra duas páginas: *Utilizadores* e *Gestão de Sites*. O objetivo é manter num só lugar as operações de governação — controlo de acesso e atualização em massa de catálogo por **CSV** — sem expor funcionalidades administrativas aos restantes perfis.

5.1.8.1 Utilizadores

A página de *Utilizadores* apresenta uma tabela simples com pesquisa por nome ou e-mail. A edição faz-se num *popup* compacto, onde é possível ajustar os campos essenciais: nome, e-mail, papel (*Super-Admin*, *Admin*, *Operador*) e estado (ativo/inativo). A criação de contas é responsabilidade exclusiva dos administradores; não existe registo público.

A Figura 38 apresenta a interface de gestão de utilizadores, mostrando a tabela principal e o *popup* de edição utilizado para pequenas alterações de dados. Este desenho reduz cliques e acelera pequenas alterações do dia a dia, mantendo a gestão de acessos sob controlo.

5.1.8.2 Gestão de Sites e Importação CSV

Em *Gestão de Sites*, o administrador vê a lista de *sites* configurados e abre o detalhe para operações específicas. A Figura 39 mostra a listagem de sites disponíveis, onde cada linha dá acesso às opções de edição e atualização de dados.

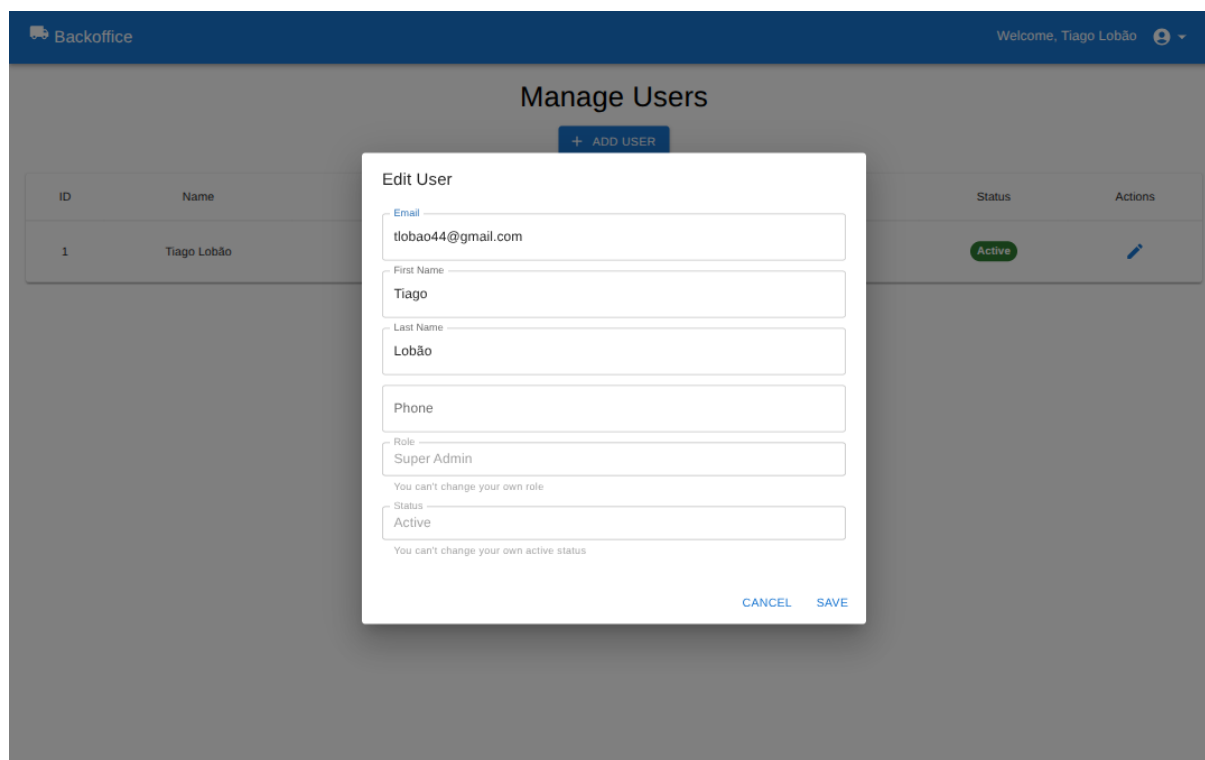


Figura 38: *Admin* — gestão de utilizadores (tabela e *popup* de edição).

A funcionalidade de importação [CSV](#) vive nesta área: o ficheiro é carregado, validado e processado em memória, aplicando atualizações de preços e quantidades ao catálogo do *site* selecionado. O mapeamento faz-se por [SKU](#), e no final é apresentado um resumo com linhas aplicadas e rejeitadas. A [Figura 40](#) ilustra o ecrã de importação, destacando o fluxo de carregamento e validação de ficheiros.

O ficheiro não é guardado no servidor e as alterações ficam imediatamente refletidas no *backoffice* e no *site* público, garantindo alinhamento entre sistemas. O acesso é restrito a *Admin/Super-Admin* para evitar alterações massivas por perfis operacionais.

5.1.9 Perfil

A página de Perfil reúne a informação básica do utilizador autenticado e permite pequenas atualizações do dia a dia, como ajustar o nome e rever os contactos. A apresentação é simples e direta, com campos claros e *feedback* imediato ao guardar alterações, sem expor opções administrativas. O objetivo é que cada utilizador mantenha os seus dados em ordem, sem sair do fluxo normal de trabalho. A [Figura 41](#) ilustra o ecrã de Perfil.

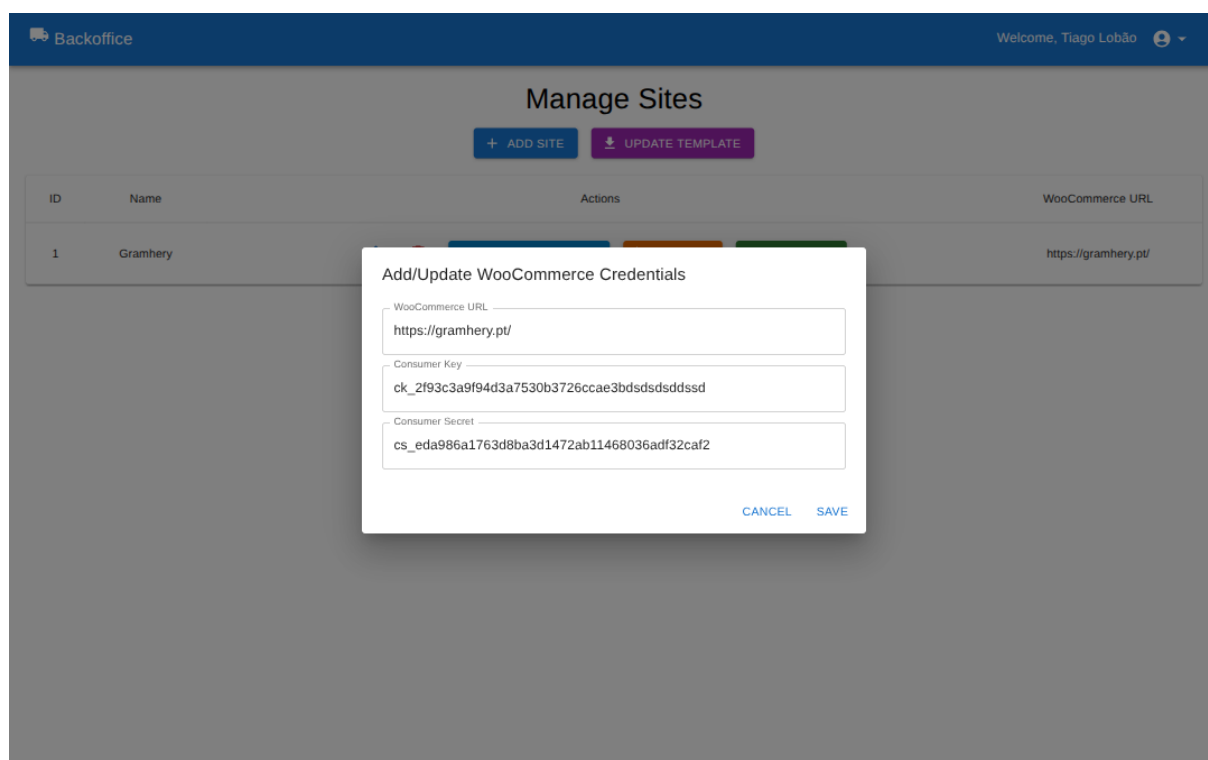


Figura 39: Admin — gestão de sites.

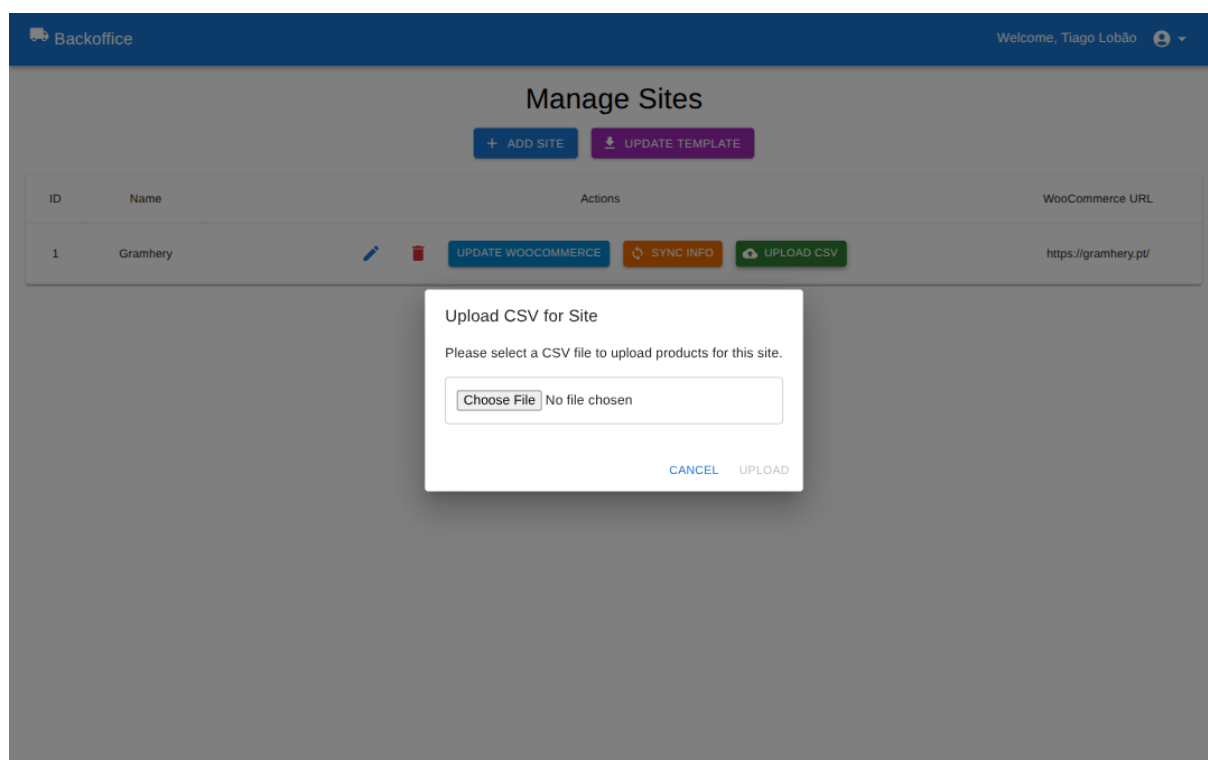


Figura 40: Importação CSV por site: atualização em massa de preços e quantidades.

Figura 41: Página de Perfil — consulta e edição de dados básicos do utilizador.

5.2 Análise Técnica Automatizada

A qualidade técnica do *frontend* foi aferida com o Google Lighthouse (perfil *Desktop*) sobre o *build* de produção servido por *Nginx*. Para auditar páginas autenticadas, manteve-se a sessão ativa (sem limpar *storage*) e mediu-se a *Home (menu)*. A Tabela 6 resume os resultados do relatório de referência.

Tabela 6: Resultados *Lighthouse (desktop)* — relatório de referência da *Home*.

Página (<i>desktop</i>)	Desempenho (%)	Acessibilidade (%)	Boas práticas (%)	SEO (%)
Home (menu)	99	88	100	100

Leitura dos resultados.

- **Desempenho 99%** — renderização muito rápida em *desktop*; a entrega estática via *Nginx* e a [SPA](#) ajudam a reduzir latência percebida.
- **Accessibility 88%** — valor bom, com margem para melhorias simples.
- **Best Practices 100%** — conformidade elevada com recomendações modernas ([Hypertext Transfer Protocol Secure \(HTTPS\)](#), uso de APIs seguras, etc.).
- **SEO 100%** — embora não seja central num *backoffice*, confirma configuração técnica correta (metadados básicos, servir via [HTTPS](#)).

Melhorias rápidas (acessibilidade).

- Garantir **rótulos visíveis** ou `aria-label` em campos e *IconButton*s (ex.: ações dos cartões).
- Rever **contraste de cores** em texto sobre imagens dos cartões para cumprir [Web Content Accessibility Guidelines \(WCAG\)](#) ($\geq 4,5:1$); no *Material UI* pode ajustar-se a paleta ou usar *overlays*.

- Definir **alt** adequado nas imagens dos cartões (ou **alt=** se forem meramente decorativas).
- Confirmar **lang** no `<html>` e foco visível em *links/buttons* para navegação por teclado.

Estes ajustamentos costumam elevar a *Accessibility* para valores acima de 95% sem impacto negativo no desempenho.

5.3 Testes de Usabilidade (plano e critérios)

Não foram realizados testes formais de usabilidade com utilizadores finais por ausência de uma entidade parceira que acolhesse o ensaio. Para colmatar esta limitação, define-se abaixo um plano de testes e os critérios de aceitação que permitiriam concluir sobre a qualidade da [User Interface \(UI\)](#); caso os limiares não fossem cumpridos, indicam-se ajustes ou reestruturação.

5.3.1 Amostra e Ambiente

A caracterização dos participantes e do ambiente de teste é essencial para assegurar condições controladas e resultados comparáveis entre sessões.

- **Participantes:** 4–6 (três Operadores, um a dois Admins e um Super-Admin).
- **Ambiente:** *build* de produção da [SPA](#) servido por Nginx; sessão autenticada; dados não sensíveis.
- **Procedimento:** instruções breves, execução das tarefas, registo de tempos/erros e questionário final (escala 1–5).

5.3.2 Tarefas (Cenários)

As tarefas seguintes simulam ações reais no sistema e permitem observar a eficácia, eficiência e clareza da interface durante o uso normal da aplicação.

- **T1** — Iniciar sessão e chegar à *Home*.
- **T2** — Localizar uma encomenda por número/cliente e abrir o detalhe.
- **T3 — Encadeado:** emitir fatura → criar envio (CTT Expresso/FedEx) → obter etiqueta/*tracking*.
Nota: a fatura é emitida antes do envio para garantir consistência dos dados entre faturação e expedição.
- **T4** — Pesquisar um produto e ajustar um campo simples (por exemplo, preço).
- **T5** — Registar uma receção (fornecedor, linhas, quantidades) e confirmar *stock*.
- **T6** — (Admin) Importar [CSV](#) por site (atualização de preços/quantidades).
- **T7** — Rever/atualizar dados no Perfil.

5.3.3 Métricas a Recolher

As métricas apresentadas a seguir permitem quantificar o desempenho e a satisfação dos utilizadores, servindo de base para avaliar a qualidade da experiência global.

- **Taxa de sucesso** por tarefa (concluída sem ajuda).
- **Tempo de conclusão** por tarefa (segundos).
- **Erros e bloqueios** observados (menores/críticos).
- **Satisfação** (questionário de 1 a 5: facilidade, clareza visual, rapidez percebida, confiança/robustez, satisfação global, recomendação).
- **Comentários abertos** (o que gostou / o que melhorava).

5.3.4 Orçamentos de Tempo (Referência)

Os tempos de referência abaixo estabelecem limites de conclusão aceitáveis para cada tarefa, permitindo identificar eventuais dificuldades de utilização.

T1: $\leq 30s$; T2: $\leq 45s$; T3 (faturação+expedição): $\leq 180s$; T4: $\leq 60s$; T5: $\leq 120s$; T6: $\leq 90s$; T7: $\leq 45s$.

5.3.5 Critérios de decisão

Os critérios seguintes definem os limiares de sucesso e as condições que determinam se a interface é satisfatória, requer ajustes ou necessita de reestruturação.

- **Bom**: sucesso $\geq 80\%$ em todas as tarefas; médias $\geq 4,0$ no questionário; tempos dentro dos orçamentos; 0 erros críticos e ≤ 1 erro menor por participante.
- **Ajustes necessários**: sucesso entre 70–79% *ou* médias entre 3,5–3,9 *ou* tempos até +20% dos orçamentos *ou* 1 erro crítico isolado.
- **Reestruturação**: sucesso 70% *ou* médias 3,5 *ou* tempos $\geq +20\%$ dos orçamentos em tarefas nucleares (T2, T3, T5) *ou* mais de um erro crítico recorrente.

Este plano estabelece um método objetivo para validar a usabilidade do *backoffice* e decidir, com base em evidência, entre aprovação, ajustes pontuais ou redesenho de fluxo/*layout*.

Conclusão

Esta dissertação apresentou a conceção e a implementação de uma plataforma *backoffice* leve para a gestão diária de uma loja *online*, centralizando operações como controlo de inventário, processamento de encomendas, faturação e criação de envios em múltiplos canais. A solução adota uma arquitetura desacoplada: [Representational State Transfer Application Programming Interface \(REST API\)](#) em *Node.js/Express* com *PostgreSQL/Prisma* no *backend* e uma [Single Page Application \(SPA\)](#) em *React* (com *Material UI*) no *frontend*. A autenticação baseia-se em [JSON Web Token \(JWT\)](#), com perfis *Super-Admin*, *Admin* e *Operador*. Integrações com *WooCommerce* (produtos/encomendas) e com transportadoras (CTT Expresso e FedEx) fecham o ciclo operacional, permitindo trabalhar num único ponto de controlo.

O modelo de dados suporta de forma explícita os fluxos do comércio: utilizadores e papéis, *sites* (canais de venda), produtos e parametrização por loja, encomendas e suas transações, entradas de *stock* (faturas de fornecedor), clientes, moradas e histórico de movimentos (*productTransactions*). No *backend*, as operações de negócio encadeiam passos críticos de forma previsível — *generateInvoice* → *updateProductsStock* → *createShipment* → *updateOrderStatus* — garantindo consistência. Etiquetas de envio e faturas são persistidas em *Base64* para simplificar a operação (serviço *stateless*). No *frontend*, a navegação em *desktop* é direta, com páginas de lista e detalhe, cartões de acesso rápido e reutilização de componentes do *Material UI*. Em produção, a aplicação corre num *Droplet* (Nginx como *reverse proxy*, [Transport Layer Security \(TLS\)](#) via Let's Encrypt, pm2 para o processo Node). A análise técnica automatizada (*Lighthouse*) em *desktop* mostrou indicadores sólidos na *Home* (por exemplo, *Desempenho* 99%, *Boas práticas* 100%), com margem conhecida para pequenas melhorias de *Acessibilidade*.

6.1 Conquistas

A solução desenvolvida atingiu os objetivos propostos, consolidando num único sistema as operações essenciais para a gestão eficiente de uma loja *online*. Desde a autenticação segmentada por perfis até à integração com plataformas externas e mecanismos automatizados de expedição, o sistema cobre com solidez os principais fluxos operacionais. Para além da componente técnica, privilegiou-se uma arquitetura extensível e um desenho orientado à experiência do utilizador. De seguida, apresentam-se as principais funcionalidades implementadas:

- **Aplicação *fullstack* pronta a operar:** [REST API](#) (Express) sobre PostgreSQL/*Prisma*; [SPA](#) em *React* com *Material UI*; entrega estática via *Nginx*.
- **Fluxo operacional de ponta a ponta:** da encomenda à fatura e expedição, num único ecrã de detalhe, com encadeamento idempotente e prevenção de duplicações (deteta fatura existente e avança no fluxo).
- **Gestão de *stock* robusta:** *picking* antes do fecho evita *stock* negativo; unicidades e índices asseguram integridade (por exemplo, [Stock Keeping Unit \(SKU\)](#) único).
- **Integrações essenciais:** *WooCommerce* (produtos/encomendas), CTT Expresso e FedEx (etiquetas e *tracking*); atualizações em massa por [Comma-Separated Values \(CSV\)](#) disponíveis apenas na área de *Administração*.
- **Segurança prática:** [JWT](#) e *bcrypt*, *allowlist* de *origins* ([Cross-Origin Resource Sharing \(CORS\)](#)), variáveis de ambiente para segredos; proteção básica por *rate limit* na *API*.
- **Desempenho e manutenção:** resultados *Lighthouse* elevados em *desktop*; código organizado por serviços no *frontend* e camadas claras no *backend*, facilitando evolução incremental.

6.2 Trabalho futuro

Embora o sistema esteja funcional e pronto para operar em contexto real, há espaço para evoluções que podem torná-lo mais robusto, automatizado e adaptável a diferentes contextos de negócio. Algumas dessas melhorias visam reforçar a rastreabilidade, reduzir a intervenção manual e aumentar a inteligência operacional. Apresentam-se abaixo propostas de desenvolvimento futuro, com potencial para alargar o alcance e a fiabilidade da plataforma.

- **Auditoria e logs de ação:** introduzir um trilho de auditoria ao nível do *backend*, registando *quem*, *quando* e *o quê* (antes/depois) em operações sensíveis (encomendas, *stock*, preços, utilizadores). Disponibilizar um painel reservado a *Admin/Super-Admin* com filtros por entidade, utilizador, intervalo temporal e tipo de evento, para controlo e diagnóstico rápido.
- **Sincronização automática de *stock*:** substituir a importação por [CSV](#) por um processo automatizado que atualize o *stock* do(s) *site(s)* com base no inventário local. Opcionalmente, incorporar consulta de *stock* dos fornecedores (quando disponível), com salvaguardas: *buffer* mínimo, limites de frequência, *fallback* para valores locais e bloqueio de *backorders* abaixo de mínimos. O objetivo é reduzir ruturas e eliminar passos manuais.
- **Cálculo automático de preços:** parametrizar regras de preço a partir do custo (margem por categoria/marca, arredondamentos comerciais, inclusão de IVA), com simulações antes de aplicar e registo de alterações para auditoria. As regras devem suportar exceções (produtos «fixados») e campanhas temporárias.

- **Faturação certificada:** integrar um sistema de faturação real (por exemplo, Moloni) via [Application Programming Interface \(API\)](#) para emissão/numeração legal de documentos, comunicação fiscal, gestão de séries e [Standard Audit File for Tax \(SAFT\)](#). A plataforma passaria a delegar a emissão oficial, mantendo apenas metadados e reconciliação entre encomendas e documentos fiscais.

Em síntese, foi construída uma base pragmática, no sentido de orientada a resultados práticos, de baixo custo operacional e sem *vendor lock-in*, que cobre o essencial do dia a dia de uma loja *online*. A arquitetura do sistema e a respetiva base de código permitem evoluções incrementais — desde melhorias de [User Interface \(UI\)](#) e [Developer Experience \(DX\)](#) até reforços de segurança e fiabilidade — acompanhando o crescimento do negócio e a integração com novos canais e operadores.

Bibliografia

- [1] J. M. Lourenço. *The NOVAtesis $\LaTeX$ Template User's Manual*. NOVA University Lisbon. 2021. url: <https://github.com/joaomlourenco/novathesis/raw/main/template.pdf> (ver p. ii).
- [2] *Shipping Delays: What Causes Them & How to Mitigate Delays*. Shopify. Section: Retail Insights. url: <https://www.shopify.com/retail/shipping-delays> (acedido em 2025-02-22) (ver p. 2).
- [3] *The Role of Logistics in eCommerce Customer Satisfaction*. Metrobi. Section: eCommerce Strategy. url: <https://metrobi.com/blog/role-of-logistics-in-ecommerce-customer-satisfaction/> (acedido em 2025-02-22) (ver p. 2).
- [4] *Understanding the Impacts of Inefficient Fulfillment Logistics on Customer Satisfaction*. HulkApps. Section: eCommerce Hub. url: <https://www.hulkapps.com/blogs/ecommerce-hub/understanding-the-impacts-of-inefficient-fulfillment-logistics-on-customer-satisfaction> (acedido em 2025-03-02) (ver p. 2).
- [5] *Automated Order Processing: Revolutionize E-commerce*. AutoStore. Section: Insights. url: <https://www.autostoresystem.com/insights/automated-order-processing-revolutionize-ecommerce> (acedido em 2025-02-26) (ver p. 4).
- [6] *How customer experience drives digital transformation*. SuperOffice. Section: Blog. url: <https://www.superoffice.com/blog/digital-transformation/> (acedido em 2025-04-10) (ver p. 4).
- [7] *E-Commerce Order Management*. Brightpearl. Section: Solutions. url: <https://www.brightpearl.com/ecommerce-order-management> (acedido em 2025-03-15) (ver p. 4).
- [8] *7 Best Ecommerce Inventory Management Software in 2025*. Webgility. Section: Blog. url: <https://www.webgility.com/blog/ecommerce-inventory-management-software> (acedido em 2024-04-10) (ver p. 4).

-
- [9] *The Evolution of E-Commerce: Tech Innovations Driving Retail*. Forbes Technology Council. Section: Retail Innovation. url: <https://councils.forbes.com/blog/evolution-of-e-commerce-in-retail> (acedido em 2025-04-10) (ver p. 4).
- [10] *The trends driving the evolution of e-commerce and AI*. RSM Global. Section: Insights. url: <https://www.rsm.global/insights/trends-driving-evolution-e-commerce-and-ai> (acedido em 2025-04-11) (ver p. 4).
- [11] *Ecommerce Frontend vs Backend: What's the Difference?* Fabric. Architecture Insights. url: <https://fabric.inc/blog/commerce/ecommerce-frontend-backend> (acedido em 2025-03-23) (ver pp. 4, 7).
- [12] *Headless commerce*. Wikipedia. Accessed online. url: https://en.wikipedia.org/wiki/Headless_commerce (acedido em 2025-04-11) (ver p. 5).
- [13] *From Click to Sale: Understanding E-Commerce Frontend*. Naturaily. UX Conversion. url: <https://naturaily.com/blog/from-click-to-sale%3A-understanding-e-commerce-frontend> (acedido em 2025-03-10) (ver p. 5).
- [14] *Why React JS Powers the Fastest and Most Scalable E-commerce Platforms*. Optimity Logics. Frontend Technologies. url: <https://optimitylogics.com/blog/why-react-js-powers-the-fastest-and-most-scalable-ecommerce-platforms/> (acedido em 2025-04-20) (ver p. 5).
- [15] *Next.js*. Wikipedia. Web Framework Overview. url: <https://en.wikipedia.org/wiki/Next.js> (acedido em 2025-03-10) (ver p. 5).
- [16] *What Is Headless Architecture? The Benefits of Headless Commerce Solutions*. Clarity Ventures. eCommerce Architecture. url: <https://www.clarity-ventures.com/headless-ecommerce-overview> (acedido em 2025-03-10) (ver p. 5).
- [17] *Headless Architecture: Benefits, Best Practices, Challenges, and Use Cases*. Crystallize. Architecture Insights. url: <https://crystallize.com/blog/headless-architecture> (acedido em 2025-03-11) (ver p. 5).
- [18] *What is Ecommerce ERP Integration?* BigCommerce. ERP Integration Overview. url: <https://www.bigcommerce.com/articles/ecommerce/erp/> (acedido em 2025-03-11) (ver p. 6).
- [19] *ERP Software for Ecommerce*. ERP Research. Research Comparison. url: <https://www.erpresearch.com/en-us/ecommerce-erp> (acedido em 2025-03-11) (ver p. 6).
- [20] *ERP for Ecommerce: The Ultimate Guide to Scaling Your Business in 2025*. Finaloop. Blog Article. url: <https://www.finaloop.com/blog/erp-for-ecommerce-the-ultimate-guide-to-scaling-your-business-in-2025> (acedido em 2025-03-23) (ver p. 6).
- [21] *7 Splendid Reasons to Keep Frontend and Backend Separate*. IT Chronicles. Software Development. url: <https://itchronicles.com/software-development/7-splendid-reasons-to-keep-frontend-and-backend-separate/> (acedido em 2025-03-23) (ver p. 7).

- [22] *eCommerce and Accounting Software Integration*. BlueLink ERP. Operational Automation. url: <https://www.bluelinkerp.com/blog/e-commerce-integration-with-inventory-accounting-erp-software/> (acedido em 2025-03-23) (ver pp. 7, 8).
- [23] Q. Zhang e H. Li. «Separation of frontend and backend in web systems». Em: *Journal of Web Engineering* 16.3 (2017), pp. 245–260 (ver p. 8).
- [24] L. Moreno e F. Andrade. «Scalable Architecture in E-commerce Systems». Em: *ACM Digital Commerce* (2020) (ver p. 8).
- [25] S. Planner. *Shelf Planner: WooCommerce Inventory Forecasting*. 2023. url: <https://shelfplanner.com/plugins/woocommerce> (ver p. 8).
- [26] Z. Corporation. *Zoho Inventory - Features and Integrations*. 2025. url: <https://www.zoho.com/inventory/> (ver p. 8).
- [27] Fishbowl. *Fishbowl Inventory Software Overview*. 2025. url: <https://www.fishbowlinventory.com/> (ver p. 8).
- [28] W. C. Studies. *Cabin Street Improves Operations with WooPayments*. 2023. url: <https://woocommerce.com/posts/woocommerce-inventory-case-study-cabin-street/> (ver p. 8).
- [29] *Product CSV Importer and Exporter Documentation – WooCommerce*. Ferramenta nativa WooCommerce para importação e exportação de produtos via CSV. url: <https://woocommerce.com/document/product-csv-importer-exporter/> (acedido em 2025-03-18) (ver p. 8).
- [30] *Introduction to Product CSV Import Suite – WooCommerce*. Extensão premium para importação/exportação de produtos via CSV com suporte a campos personalizados. url: <https://woocommerce.com/document/product-csv-import-suite/> (acedido em 2025-03-18) (ver p. 8).
- [31] *Price and Stock Import – WooCommerce Marketplace*. Plugin para atualização de preços e stock via CSV. url: <https://woocommerce.com/products/price-stock-import/> (acedido em 2025-03-18) (ver p. 8).
- [32] *WooCommerce Product Sync via API*. Plugin que sincroniza produtos e stock via APIs JSON/XML. url: <https://plugins.miniorange.com/woocommerce-api-product-sync-with-woocommerce-rest-apis> (acedido em 2025-03-18) (ver p. 8).
- [33] *Order Management & Warehouse Management – Stok.ly*. Sistema central ‘hub and spoke’ com atualização via CSV e APIs. url: <https://www.stok.ly/order-management-warehouse-management/> (acedido em 2025-03-18) (ver p. 9).
- [34] *WooCommerce Shipping Tracking Integration*. Plugin para gestão de envios via ShipStation no WooCommerce. url: <https://www.shipstation.com/partners/woocommerce/> (acedido em 2025-03-19) (ver p. 9).

-
- [35] *WooCommerce Shipping Documentation*. Solução nativa do WooCommerce para geração de etiquetas de envio. url: <https://docs.woocommerce.com/document/woocommerce-shipping/> (acedido em 2025-03-19) (ver p. 9).
- [36] *Sendcloud WooCommerce Integration*. Integração WooCommerce com múltiplas transportadoras e tracking. url: <https://www.sendcloud.com/integrations/woocommerce/> (acedido em 2025-03-19) (ver p. 9).
- [37] *Easyship - Powering Shipping for Woo Stores*. Envio internacional com cotações, etiquetas e tracking. url: <https://www.easyship.com/integrations/woocommerce-shipping> (acedido em 2025-03-19) (ver p. 10).
- [38] *Ship WooCommerce Orders With Shippo*. Gestão de envios com tracking e envios globais. url: <https://goshippo.com/partners/woocommerce-shipping-plugin> (acedido em 2025-03-19) (ver p. 10).
- [39] *powerfulwp. Order Picking For WooCommerce plugin*. Plugin que gere o processo de picking com estados de encomenda. url: <https://wordpress.org/plugins/order-picking-for-woocommerce/> (acedido em 2025-03-19) (ver p. 11).
- [40] C. AS. *Pick List for WooCommerce*. Plugin para listas de picking, scanning e geração de etiquetas. url: <https://woocommerce.com/products/picklist/> (acedido em 2025-03-19) (ver p. 11).
- [41] SkyVerge. *Print Invoices and Packing Lists for WooCommerce*. Permite imprimir listas de packing e picking manual ou automática. url: <https://woocommerce.com/products/print-invoices-packing-lists/> (acedido em 2025-03-20) (ver p. 12).
- [42] F. Marchand. *25 Best Pick And Pack Software In 2025*. Comparação de soluções de pick pack profissionais. url: <https://theretailexec.com/tools/best-pick-and-pack-software/> (acedido em 2025-03-20) (ver p. 12).
- [43] G. Hobbs. *Top Shipping Software for Ecommerce Businesses*. Integração WMS e realtime reporting para picking, packing, ship. url: <https://www.inboxarmy.com/blog/shipping-software-for-ecommerce-businesses/> (acedido em 2025-03-20) (ver p. 12).
- [44] *Inventory Sync for Multichannel Ecommerce*. Importância de sincronização de stock em tempo real. url: <https://feedonomics.com/blog/inventory-sync/> (acedido em 2025-03-21) (ver pp. 12, 13).
- [45] *Ensure product data accuracy with Event Driven Sync*. Sincronização orientada a eventos para prevenção de overselling. url: <https://feedonomics.com/event-driven-real-time-sync/> (acedido em 2025-03-21) (ver p. 13).
- [46] *WooCommerce Product Sync via API*. Sincronização de produtos e stock entre WooCommerce via API. url: <https://plugins.miniorange.com/woocommerce-api-product-sync-with-woocommerce-rest-apis> (acedido em 2025-03-21) (ver p. 13).

- [47] E. A. /. V. OMNI. *Case Study: Zippo - Shopify Plus ERP Integration*. Integração bidirecional de Shopify Plus com ERP via VL OMNI. url: <https://ecommerceaccelerators.com/pages/case-study/zippo> (acedido em 2025-03-21) (ver p. 14).
- [48] Kensium. *3DXTECH Shopify Plus Acumatica Integration Case Study*. Migração para Shopify Plus, integração bidirecional e PWA. url: <https://www.kensium.com/case-study/3dxtech> (acedido em 2025-02-22) (ver p. 15).
- [49] S. Ravooof. *Next.js vs React? It's a Partnership, Not a Competition*. Comparação entre React.js e Next.js no desenvolvimento frontend moderno. url: <https://kinsta.com/blog/nextjs-vs-react/> (acedido em 2025-02-22) (ver p. 16).
- [50] J. D. Scarsbrook, M. Utting e R. K. L. Ko. *TypeScript's Evolution: An Analysis of Feature Adoption Over Time*. arXiv. Estudo acadêmico sobre a evolução de funcionalidades no TypeScript. 2023. (Acedido em 2025-02-22) (ver p. 16).
- [51] Cloudinary. *Node.js Ecommerce: Benefits, Architecture, and Best Practices*. Benefícios do Node.js para e-commerce: escalabilidade, performance e ecossistema. url: <https://cloudinary.com/guides/e-commerce-platform/node-js-ecommerce-benefits-architecture-and-best-practices> (acedido em 2025-02-22) (ver p. 16).
- [52] ccpythonprogramming. *Automating E-commerce Operations with n8n and Python*. Automação de operações e-commerce usando Python e n8n. url: <https://medium.com/@ccpythonprogramming/automating-e-commerce-operations-with-n8n-and-python-a-real-world-application-fd8444d0ca11> (acedido em 2025-02-22) (ver p. 16).
- [53] Cloudways. *WooCommerce REST API: Detailed Integration Guide*. Guia sobre uso da API REST do WooCommerce para integração avançada. url: <https://www.cloudways.com/blog/woocommerce-rest-api/> (acedido em 2025-02-22) (ver p. 16).
- [54] DroidSonRoids. *Flutter vs React Native: Complete 2025 Framework Comparison Guide*. Comparativo detalhado entre Flutter e React Native em 2025. url: <https://www.thedroidsonroids.com/blog/flutter-vs-react-native-comparison> (acedido em 2025-02-22) (ver p. 16).
- [55] Nomtek. *Flutter vs. React Native in 2025: Detailed Analysis*. Dados estatísticos sobre popularidade de Flutter e React Native. url: <https://www.nomtek.com/blog/flutter-vs-react-native> (acedido em 2025-02-25) (ver p. 16).
- [56] I. C. Alexandra Mendes. *Best Types of Software Architecture Patterns Explained*. Explicação e comparação de padrões como monolítico, camadas and microserviços. url: <https://www.imaginarycloud.com/blog/types-of-software-architecture-patterns> (acedido em 2025-02-25) (ver pp. 16, 17).

-
- [57] V. Kumar. *Monolithic vs. Microservices Architecture: A Detailed Comparison*. Vantagens e desafios de cada abordagem. url: https://dev.to/vikram_kumar_2101/-day-2-monolithic-vs-microservices-architecture-a-detailed-comparison-fkc (acedido em 2025-02-25) (ver p. 16).
- [58] A. Selin. *Understanding Space-Based Architecture*. Introdução e análise de Space-Based Architecture. url: <https://medium.com/the-pragmatic-tech-review/understanding-space-based-architecture-15281953c24c> (acedido em 2025-02-25) (ver p. 17).
- [59] Nerdnodes. *Space-Based Architecture Pattern: Understanding Software Architecture Patterns*. Comparativo entre Space-Based e Microservices. url: <https://nerdnodes2023.medium.com/space-based-architecture-pattern-understanding-software-architecture-patterns-5-e4d55f849966> (acedido em 2025-02-25) (ver p. 17).
- [60] S. Koltsov. *Exploring Common Software Architecture Patterns: Monolithic, SOA, Microservices and Layered*. Descrição de padrões como monolítico, SOA, microserviços e em camadas. url: <https://staskoltsov.medium.com/exploring-common-software-architecture-patterns-monolithic-soa-microservices-and-layered-804ed88e6bef> (acedido em 2025-02-25) (ver p. 18).
- [61] D. Blog. *Microservices vs. monolithic architecture: Understanding the difference*. Benefícios, testes e deploy de microserviços em comparação com monólitos. url: <https://www.dynatrace.com/news/blog/microservices-vs-monolithic-architecture/> (acedido em 2025-02-27) (ver p. 18).
- [62] M. Shashisshashis. *Comparing SOA with Event-Driven Architecture*. Comparativo entre SOA e arquitetura orientada a eventos. url: <https://medium.com/@shashisshashis792/comparing-soa-with-event-driven-architecture-66630ef97d56> (acedido em 2025-02-25) (ver p. 18).
- [63] A. K. Luhach, S. K. Dwivedi e C. K. Jha. *Designing and implementing the logical security framework for e-commerce based on service oriented architecture*. Estudo acadêmico sobre arquitetura SOA em e-commerce. url: <https://arxiv.org/abs/1407.2421> (acedido em 2025-02-27) (ver p. 18).
- [64] A. Documentation. *What is an Event-Driven Architecture?* Descrição e benefícios de architectures baseadas em eventos. url: <https://aws.amazon.com/event-driven-architecture/> (acedido em 2025-02-27) (ver p. 19).
- [65] E. Blog. *Event Driven Architecture for E-commerce*. Aplicação de EDA em plataformas de comércio B2B/B2C. url: <https://www.emporix.com/blog/event-driven-architecture-for-e-commerce> (acedido em 2025-02-27) (ver p. 19).
- [66] Confluent. *Event-Driven Architecture (EDA): A Complete Introduction*. Explicação e casos de uso de EDA. url: <https://www.confluent.io/learn/event-driven-architecture/> (acedido em 2025-02-27) (ver p. 19).

- [67] C. Technologies. *Headless Architecture: When Do We Really Need It?* Análise das vantagens e desafios da arquitetura headless no contexto de e-commerce. url: <https://www.codilar.com/blog/headless-architecture-when-do-we-really-need-it/> (acedido em 2025-02-27) (ver p. 19).
- [68] DashDevs. *Headless commerce: What is it and how does it work?* Explicação clara da arquitetura headless aplicada ao comércio eletrônico. url: <https://dashdevs.com/blog/headless-commerce-what-is-it-and-how-does-it-work/> (acedido em 2025-02-27) (ver p. 19).
- [69] N. Foundation. *Node.js Documentation*. Documentação oficial da plataforma Node.js utilizada para construir o backend. url: <https://nodejs.org/en/docs> (acedido em 2025-04-21) (ver p. 27).
- [70] Express.js. *Express Documentation*. Documentação oficial do framework Express, utilizado na estrutura do servidor HTTP. url: <https://expressjs.com/> (acedido em 2025-04-21) (ver p. 27).
- [71] T. P. G. D. Group. *PostgreSQL Documentation*. Documentação oficial do sistema de gestão de base de dados PostgreSQL. url: <https://www.postgresql.org/docs/> (acedido em 2025-04-21) (ver p. 27).
- [72] Prisma. *Prisma ORM Documentation*. Documentação do ORM Prisma, utilizado para a integração entre Node.js e PostgreSQL. url: <https://www.prisma.io/docs> (acedido em 2025-04-21) (ver p. 27).
- [73] I. Meta Platforms. *React Documentation*. Documentação oficial do React, principal biblioteca utilizada no desenvolvimento do frontend. url: <https://react.dev/> (acedido em 2025-04-21) (ver p. 28).
- [74] R. Software. *React Router Documentation*. Documentação da biblioteca React Router, usada para navegação entre páginas da aplicação. url: <https://reactrouter.com/en/main> (acedido em 2025-04-22) (ver p. 28).
- [75] MUI. *Material UI Documentation*. Referência principal para os componentes de interface gráfica da biblioteca Material UI. url: <https://mui.com/material-ui/getting-started/overview/> (acedido em 2025-04-22) (ver p. 28).
- [76] A. Project. *Axios Documentation*. Biblioteca utilizada para realizar chamadas HTTP entre o frontend React e o backend Node.js. url: <https://axios-http.com/docs/intro> (acedido em 2025-04-22) (ver p. 28).