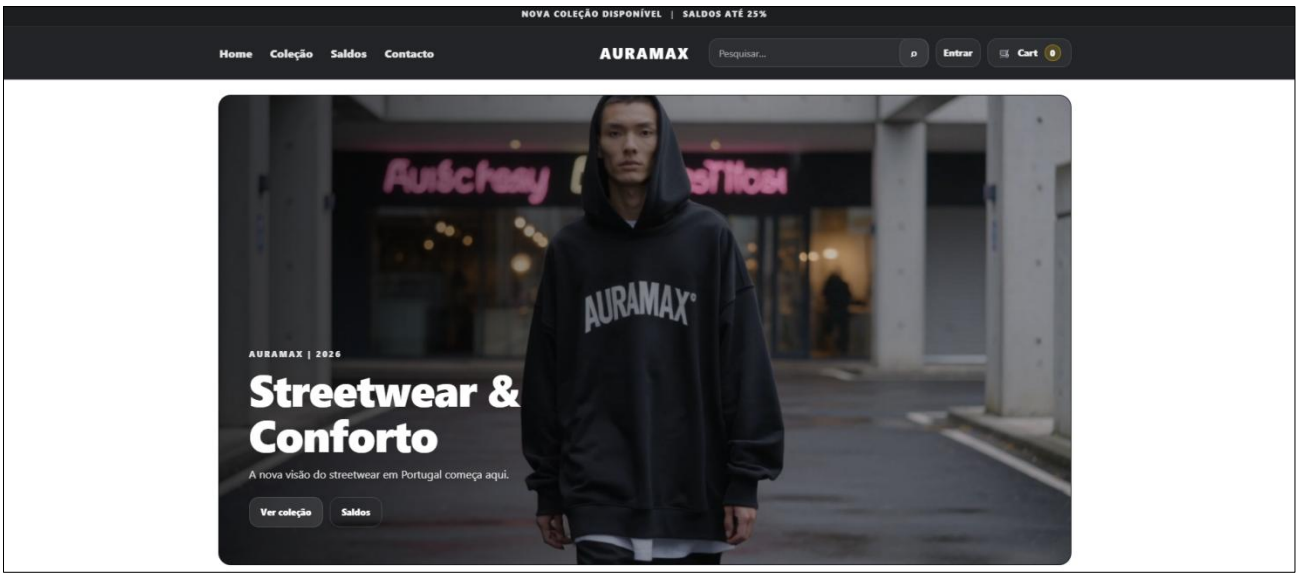
Agrupamento de Escolas
MADEIRA TORRES

TIS

Curso Profissional Técnico/a de Informática-Sistemas

Relatório Final

Prova de Aptidão Profissional



Tomás Inácio

2025/2026

REPÚBLICA PORTUGUESA | EDUCAÇÃO, CIÊNCIA E INOVAÇÃO
ANQEP
SELO DE CONFORMIDADE EQAVET
PESSOAS 2030
PORTUGAL 2030
Cofinanciado pela União Europeia

Índice

Agradecimentos

A realização deste projeto só foi possível graças ao apoio e colaboração de várias pessoas que contribuíram, direta ou indiretamente, para o seu desenvolvimento.

Em primeiro lugar, agradeço aos professores que me acompanharam ao longo do curso, pelo apoio, orientação e conhecimentos transmitidos, fundamentais para a concretização deste trabalho. A sua disponibilidade para esclarecer dúvidas e orientar o projeto foi essencial durante todo o processo.

Agradeço também à escola, por disponibilizar os recursos necessários e por proporcionar as condições adequadas para o desenvolvimento deste Projeto de Aptidão Profissional.

Por fim, deixo um agradecimento especial à minha família e amigos, pelo apoio, incentivo e motivação ao longo de todo o percurso académico, que foram determinantes para a conclusão deste projeto.

Dedicatória

Dedico este Projeto de Aptidão Profissional à minha família, pelo apoio constante e pela motivação ao longo do meu percurso escolar, e a todos aqueles que, de alguma forma, contribuíram para que este objetivo fosse alcançado.

Epígrafe

“O design não é apenas o que parece e o que se sente. O design é como funciona.” (SteveJobs)

“A tecnologia só é verdadeiramente útil quando simplifica a vida das pessoas.” (Bill Gates)

Lista de ilustrações

Figura 86 Ajuda rápida

Lista de acrónimos, abreviaturas e siglas

PAP- Prova de Aptidão Profissional

SOA - Sistemas Operativos Aplicações

FCT - Formação em contexto de trabalho

Introdução

No âmbito da Prova de Aptidão Profissional (PAP), foi desenvolvido um website *front-end* para uma loja de roupa fictícia denominada Auramax. Este projeto tem como objetivo principal aplicar e consolidar os conhecimentos adquiridos ao longo do curso na área do desenvolvimento web, recorrendo a tecnologias atuais e adequadas ao contexto profissional.

O website simula uma loja online de *streetwear*, permitindo ao utilizador navegar entre diferentes páginas, visualizar produtos, gerir um carrinho de compras e efetuar o registo ou início de sessão. O funcionamento do website integra front-end e back-end, com ligação a uma base de dados (SQLite) responsável por armazenar utilizadores, produtos e encomendas.

A marca Auramax foi criada em 2026 com o propósito de inovar o *streetwear* em Portugal, apostando num design moderno, simples e funcional. Essa identidade é refletida no website através de uma interface *clean*, responsiva e intuitiva, proporcionando uma experiência de navegação clara, agradável e acessível.

Este projeto visa demonstrar a capacidade de planear, estruturar e desenvolver uma aplicação web *full-stack*, aplicando boas práticas de organização de código, usabilidade e design, fundamentais no desenvolvimento de soluções digitais eficazes.

Desenvolvimento do projeto

O desenvolvimento da plataforma AURAMAX foi realizado com o objetivo de criar uma loja online moderna, rápida, segura e com uma identidade visual premium. Desde o início do projeto foi definida uma arquitetura *fullstack* separada entre front-end e back-end, permitindo uma melhor organização do código e facilitando futuras expansões da plataforma.
O projeto foi desenvolvido em várias fases, começando pelo planeamento da estrutura do website e pela criação da identidade visual da marca. O principal objetivo foi desenvolver uma plataforma moderna, intuitiva e visualmente apelativa para o utilizador.
Foi criado um design minimalista inspirado em marcas de roupa premium modernas, utilizando cores neutras, contrastes escuros e tipografia simples para transmitir uma imagem elegante e profissional.
Após a definição da parte visual iniciou-se o desenvolvimento técnico do front-end utilizando React e Vite. O React permitiu criar uma aplicação organizada através de componentes reutilizáveis, facilitando a manutenção do projeto e a reutilização de código em diferentes páginas do website.
Foram desenvolvidos vários componentes importantes como a Navbar, Footer, ProductCard, sistema de carrinho, componentes de autenticação e sistema de pesquisa. A estrutura do front-end foi organizada em diferentes pastas, permitindo manter o projeto mais organizado e simples de gerir.
Cada página da plataforma foi desenvolvida individualmente, incluindo página inicial, produtos, detalhe de produto, login, registo, conta do utilizador, contactos, carrinho, administração, saldos e página sobre.

Posteriormente foi desenvolvido o back-end utilizando Python e FastAPI. A FastAPI foi escolhida devido à sua rapidez, simplicidade e excelente integração com APIs modernas. No back-end foram implementadas funcionalidades fundamentais como autenticação JWT, gestão de utilizadores, produtos, encomendas, promoções, upload de imagens, rotas protegidas, sistema de administração e integração com a plataforma Brevo.

A comunicação entre front-end e back-end é realizada através de pedidos HTTP utilizando fetch API, sendo os dados enviados e recebidos em formato JSON. Foi criada uma API REST organizada através de diferentes endpoints responsáveis pelas várias funcionalidades da plataforma.

O back-end utiliza SQLAlchemy para comunicação com a base de dados SQLite. Durante o desenvolvimento foram criadas relações entre tabelas para melhorar a organização da informação, especialmente entre utilizadores e encomendas, produtos e imagens, encomendas e itens de encomenda, bem como promoções e produtos.

Um dos principais focos do projeto foi a segurança. Por esse motivo foram implementados sistemas de *hash* de *passwords*, autenticação *JWT*, verificação de email, rotas protegidas, validação de dados e *middleware* de *CORS*.

Foi também desenvolvido um sistema completo de verificação de email utilizando a plataforma *Brevo*. Quando um utilizador cria conta, é gerado um *token* único que fica guardado na base de dados. Posteriormente, é enviado automaticamente um email de confirmação e apenas após a validação desse email o utilizador consegue iniciar sessão na plataforma.

Além disso, foi criado um painel de administração protegido, acessível apenas a utilizadores com permissões de administrador, onde é possível criar produtos, adicionar e organizar imagens, e gerir o stock disponível.

Durante o desenvolvimento foram realizados vários testes para garantir a estabilidade e o funcionamento correto da plataforma, incluindo testes ao *login*, registo, carrinho, sistema de encomendas, painel de administração, verificação de email, *upload* de imagens e responsividade. Foram também testadas as rotas protegidas, ou seja, páginas que só podem ser acedidas por utilizadores autenticados (e, em alguns casos, apenas por administradores), redirecionando automaticamente quem tenta aceder sem permissão.

A plataforma foi otimizada para funcionar corretamente em computadores, *tablets* e *smartphones*. O design responsivo foi desenvolvido utilizando *CSS* moderno e *media queries*.

Este projeto permitiu aplicar conhecimentos importantes nas áreas de programação web, bases de dados, *APIs*, segurança, desenvolvimento *front-end* e *back-end*, organização de projetos e design de interfaces.

Este projeto permitiu aplicar conhecimentos importantes nas áreas de programação web, bases de dados, *APIs*, segurança, desenvolvimento *frontend* e *backend*, organização de projetos e design de interfaces.

Aplicações Utilizadas no Projeto

Visual Studio Code

O Visual Studio Code foi o editor utilizado para desenvolver todo o projeto AURAMAX. Esta ferramenta permitiu escrever, organizar e testar o código de forma rápida e eficiente.

O VS Code também facilitou a organização do projeto graças às suas extensões para React, Python e GitHub.

React

O React foi utilizado para desenvolver o frontend do website.

Esta biblioteca permitiu criar uma interface moderna, rápida e organizada através de componentes reutilizáveis.

Com React foi possível desenvolver funcionalidades como:

- Sistema de carrinho
- Login e registo
- Pesquisa de produtos
- Navegação dinâmica
- Painel de administração

O React também ajudou a melhorar a organização do código e a experiência do utilizador.

Vite

O Vite foi utilizado para criar e executar o projeto React.

Esta ferramenta permitiu iniciar o website rapidamente e atualizar automaticamente as alterações feitas durante o desenvolvimento.

O Vite ajudou a tornar o desenvolvimento mais rápido e eficiente.

JavaScript

O JavaScript foi utilizado para implementar toda a lógica dinâmica do website.

Foi utilizado para:

- Comunicação com a API
- Gestão do carrinho
- Sistema de autenticação
- Atualização de páginas
- Validação de formulários

O JavaScript tornou o website mais interativo e dinâmico.

Python

O Python foi utilizado no backend através da framework FastAPI.

Foi responsável pela lógica do servidor e pela comunicação com a base de dados.

Com Python foram desenvolvidas funcionalidades como:

- Autenticação
- Gestão de produtos
- Gestão de utilizadores
- Sistema de encomendas
- Verificação de email

FastAPI

A FastAPI foi utilizada para desenvolver a API REST do projeto.

Esta framework permitiu criar endpoints rápidos, seguros e organizados para ligação entre frontend e backend.

Algumas das rotas criadas foram:

- Login
- Registo
- Produtos
- Encomendas
- Utilizadores

A FastAPI também facilitou a validação de dados e a organização do backend.

SQLite

A SQLite foi utilizada como base de dados principal durante o desenvolvimento.

Nesta base de dados foram armazenadas informações como:

- Utilizadores
- Produtos
- Encomendas
- Promoções
- Imagens

A SQLite foi escolhida por ser simples, leve e fácil de integrar com Python.

JWT (JSON Web Token) 

O JWT foi utilizado para autenticação segura dos utilizadores.

Depois do login, o sistema gera um token que identifica o utilizador autenticado.

Esse token é utilizado para aceder a áreas protegidas da plataforma.

O JWT permitiu criar um sistema de login moderno e seguro.

Brevo

A plataforma Brevo foi utilizada para envio de emails automáticos.

Quando um utilizador cria conta, recebe um email de verificação para confirmar o registo.

Isto permitiu aumentar a segurança e tornar o sistema mais profissional.

HTML e CSS 

O HTML e CSS foram utilizados para estruturar e estilizar toda a interface do website.

Foi desenvolvido um design moderno, minimalista e responsivo.

O CSS permitiu criar:

- Layouts organizados
- Responsividade
- Animações
- Efeitos visuais
- Design adaptado a diferentes dispositivos

O objetivo principal foi criar uma experiência visual moderna e agradável para o utilizador.

Dificuldades Encontradas

Durante o desenvolvimento do projeto surgiram várias dificuldades técnicas, principalmente na parte do *backend*. Como este foi o primeiro projeto *fullstack* desenvolvido com *FastAPI*, foi necessário aprender e resolver vários problemas relacionados com autenticação, comunicação entre *frontend* e *backend* e gestão da base de dados.

Uma das primeiras dificuldades encontradas foi a criação da *API* e a organização das rotas. Foi necessário compreender melhor o funcionamento do *FastAPI*, dos *endpoints*, dos pedidos *HTTP* e da comunicação entre o *backend* e o *frontend* desenvolvido em *React*.

Outra dificuldade importante foi a implementação do sistema de autenticação com *JWT*. Inicialmente surgiram vários problemas relacionados com a criação dos tokens, validação dos utilizadores, autenticação de rotas, gestão de sessões e proteção de páginas privadas. Também existiram dificuldades na utilização do *OAuth2PasswordBearer* e na gestão correta dos tokens enviados pelo *frontend*.

O sistema de verificação de email através da plataforma Brevo foi outro dos maiores desafios do projeto. Durante a implementação surgiram problemas relacionados com a configuração da *API key*, autenticação da conta Brevo, autorização de IPs, configuração do ficheiro *.env*, envio automático de emails e validação dos tokens de confirmação. Foi necessário realizar vários testes até conseguir implementar corretamente o sistema de verificação de contas.

Outra dificuldade encontrada foi a configuração do *CORS* entre *frontend* e *backend*. Como o *frontend* e o *backend* estavam a funcionar em portas diferentes, o *browser* bloqueava alguns pedidos por motivos de segurança. Para resolver este problema foi necessário configurar corretamente o middleware *CORSMiddleware* no *FastAPI*.

Também surgiram problemas relacionados com a base de dados *SQLite* e *SQLAlchemy*. Em alguns momentos ocorreram erros após alterações aos modelos da base de dados, especialmente quando eram adicionadas novas colunas às tabelas. Em vários casos foi necessário apagar e recriar a base de dados para corrigir incompatibilidades entre os modelos e a estrutura existente.

A gestão das relações entre tabelas também apresentou alguma dificuldade, principalmente nas relações entre utilizadores e encomendas, produtos e imagens e encomendas e itens de encomenda.

Outro desafio importante foi a gestão do stock e do sistema de encomendas, sendo necessário garantir que o stock dos produtos era atualizado corretamente após cada compra realizada pelos utilizadores.

Além disso, também existiram dificuldades na integração entre *frontend* e *backend*, principalmente na comunicação entre *React* e *FastAPI* através de pedidos *fetch*. Durante o desenvolvimento ocorreram erros relacionados com pedidos *HTTP*, respostas *JSON*, autenticação, estados *React*, rotas protegidas e validação de dados.

Apesar de todas as dificuldades encontradas, todos os problemas foram resolvidos através de pesquisa, testes práticos, documentação oficial e *debugging* constante. Estas dificuldades contribuíram bastante para a aprendizagem e evolução técnica ao longo do desenvolvimento do projeto.

Cronograma

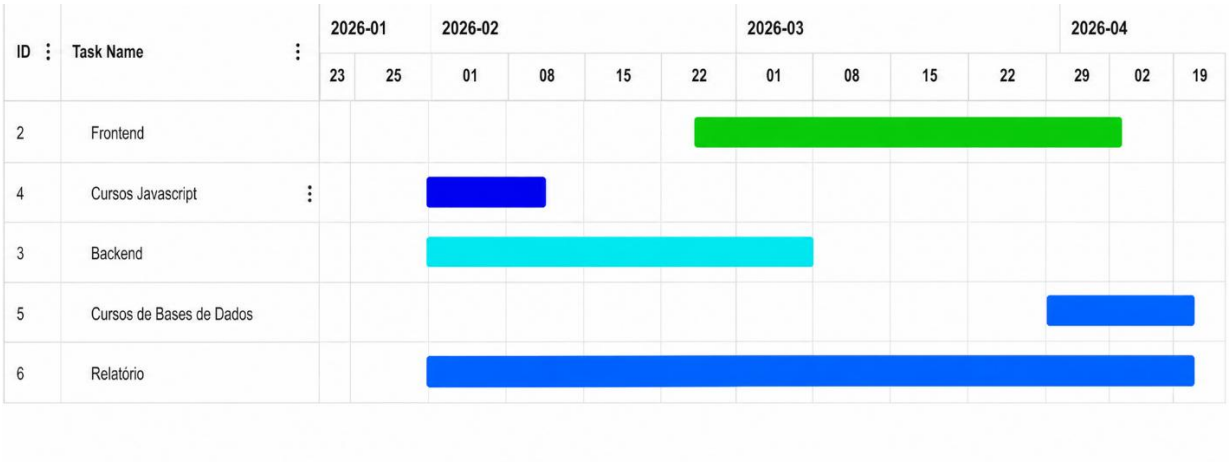


Figura 2 Cronograma

O cronograma está organizado por fases, permitindo acompanhar a evolução do projeto ao longo do tempo. As tarefas de formação em JavaScript e Bases de Dados decorrem em paralelo com o desenvolvimento do *backend* e *frontend*, otimizando o tempo disponível. O relatório é desenvolvido durante grande parte do projeto, acompanhando a implementação das restantes atividades, garantindo uma distribuição equilibrada das tarefas e o cumprimento dos prazos estabelecidos.

Conclusão

A realização deste projeto permitiu aplicar e aprofundar muitos dos conhecimentos adquiridos ao longo do Curso Profissional Técnico de Informática – Sistemas. O desenvolvimento da plataforma AURAMAX foi uma experiência bastante importante, pois permitiu criar uma loja online moderna e funcional, utilizando tecnologias atuais tanto no *frontend* como no *backend*.

Ao longo do desenvolvimento surgiram várias dificuldades, principalmente na parte do *backend*, na autenticação de utilizadores, na ligação entre *frontend* e *backend*, na gestão da base de dados e na implementação do sistema de verificação de email. Apesar disso, todos os problemas foram resolvidos através de pesquisa, testes e aprendizagem contínua, o que contribuiu bastante para a evolução técnica e pessoal durante o projeto.

Este trabalho também permitiu melhorar capacidades importantes como a autonomia, a organização, a resolução de problemas e a capacidade de desenvolver soluções completas de forma mais profissional. Além disso, foi possível ganhar uma melhor compreensão sobre o funcionamento de aplicações web modernas e sobre a importância da segurança, da organização do código e da experiência do utilizador.

Considero que os objetivos definidos no início do projeto foram atingidos com sucesso, uma vez que foi possível desenvolver uma plataforma completa com sistema de autenticação, verificação de email, gestão de produtos, carrinho de compras, encomendas e painel de administração.

Como possível melhoria futura, seria interessante tornar a página de promoções totalmente operacional, permitindo a criação e gestão dinâmica de campanhas promocionais e descontos, enriquecendo a experiência dos utilizadores e aumentando as funcionalidades disponíveis na plataforma.

Em conclusão, este projeto representou uma experiência muito positiva e enriquecedora, permitindo consolidar conhecimentos importantes e preparar-me melhor para futuros desafios na área do desenvolvimento de software e informática.

Referências bibliográficas

React Documentation. Disponível em:

React Official Website

FastAPI Documentation. Disponível em:

FastAPI Official Documentation

Vite Documentation. Disponível em:

Vite Official Documentation

Python Documentation. Disponível em:

Python Official Documentation

SQLAlchemy Documentation. Disponível em:

SQLAlchemy Official Documentation

JWT Official Website. Disponível em:

JWT Official Website

Brevo Developers Documentation. Disponível em:

Brevo Developers Documentation

MDN Web Docs. Disponível em:

MDN Web Docs

Visual Studio Code Documentation. Disponível em:

Visual Studio Code Official Website

PassLib Documentation. Disponível em:

PassLib Documentation

Glossário

API (Application Programming Interface) – Conjunto de regras que permite a comunicação entre diferentes aplicações ou sistemas através de pedidos e respostas.

API REST – Tipo de API baseada no protocolo HTTP que organiza a comunicação através de endpoints e métodos como GET, POST, PUT e DELETE.

Autenticação – Processo de verificação da identidade de um utilizador antes de permitir o acesso a determinadas funcionalidades.

Backend – Parte da aplicação responsável pela lógica de negócio, processamento de dados, autenticação e comunicação com a base de dados.

Base de Dados – Sistema utilizado para armazenar, organizar e gerir informações de forma estruturada.

Brevo – Plataforma utilizada para o envio automático de emails, incluindo mensagens de verificação de conta e confirmação de encomendas.

CORS (Cross-Origin Resource Sharing) – Mecanismo de segurança que permite controlar a comunicação entre aplicações executadas em origens diferentes.

CRUD (Create, Read, Update, Delete) – Conjunto das operações básicas realizadas sobre dados numa aplicação.

Endpoint – URL específica de uma API que executa uma determinada funcionalidade ou fornece acesso a determinados dados.

FastAPI – Framework desenvolvida em Python utilizada para criar APIs rápidas, modernas e seguras.

Fetch API – Funcionalidade do JavaScript utilizada para efetuar pedidos HTTP entre o frontend e o backend.

Frontend – Parte visual da aplicação com a qual o utilizador interage diretamente através do navegador.

Hashing – Processo de transformação de dados, como passwords, em códigos irreversíveis para aumentar a segurança.

HTML (HyperText Markup Language) – Linguagem utilizada para estruturar o conteúdo das páginas web.

HTTP (HyperText Transfer Protocol) – Protocolo utilizado para a comunicação entre clientes e servidores na web.

JSON (JavaScript Object Notation) – Formato leve utilizado para a troca e armazenamento de dados estruturados.

JWT (JSON Web Token) – Token utilizado para autenticar utilizadores e proteger o acesso a recursos da aplicação.

Middleware – Componente intermédio que processa pedidos e respostas antes de chegarem ao destino final.

OAuth2 – Protocolo utilizado para autenticação e autorização segura em aplicações web.

React – Biblioteca JavaScript utilizada para desenvolver interfaces de utilizador modernas através de componentes reutilizáveis.

React Router – Biblioteca utilizada para gerir a navegação entre páginas numa aplicação React.

Responsividade – Capacidade de uma aplicação adaptar automaticamente a sua interface a diferentes tamanhos de ecrã.

Route (Rota) – Caminho que define uma página ou funcionalidade acessível através de um endereço específico da aplicação.

SQLite – Sistema de gestão de bases de dados relacional leve que armazena toda a informação num único ficheiro.

SQLAlchemy – Biblioteca Python utilizada para facilitar a comunicação entre o backend e a base de dados.

Stock – Quantidade disponível de um produto para venda.

Token – Código único utilizado para identificar ou validar um utilizador ou ação específica.

UI (User Interface) – Interface gráfica através da qual o utilizador interage com a aplicação.

Upload – Processo de envio de ficheiros do dispositivo do utilizador para o servidor.

URL (Uniform Resource Locator) – Endereço utilizado para localizar recursos na internet.

UX (User Experience) – Experiência geral do utilizador durante a utilização da aplicação.

Vite – Ferramenta de desenvolvimento utilizada para criar e executar aplicações React de forma rápida.

Visual Studio Code (VS Code) – Editor de código utilizado no desenvolvimento do projeto.

Verificação de Email – Processo que confirma a validade do endereço de email de um utilizador através de um link enviado por correio eletrónico.

Anexos

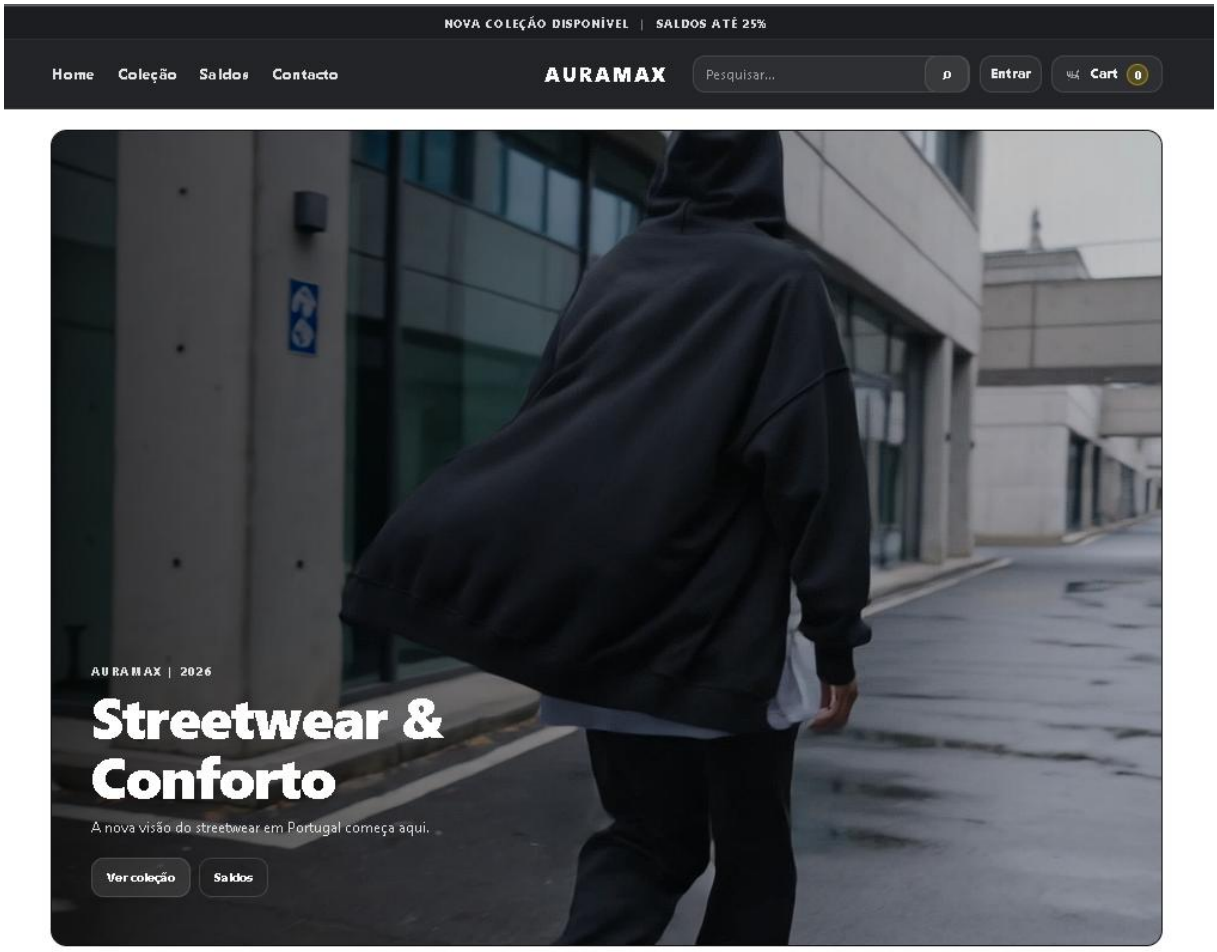


Figura 34 Página Inicial

Figura 3 Categorias

Figura 4 Navbar

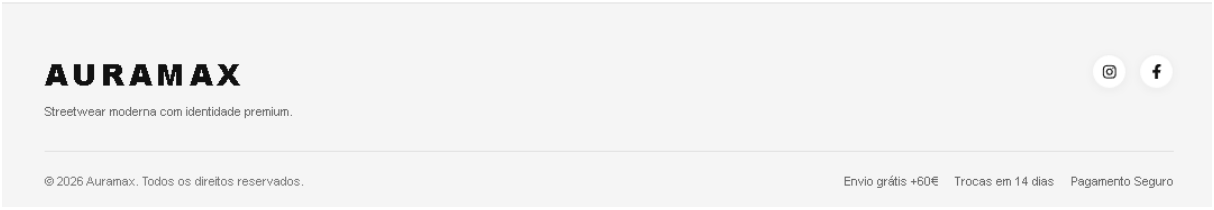


Figura 5 Footer

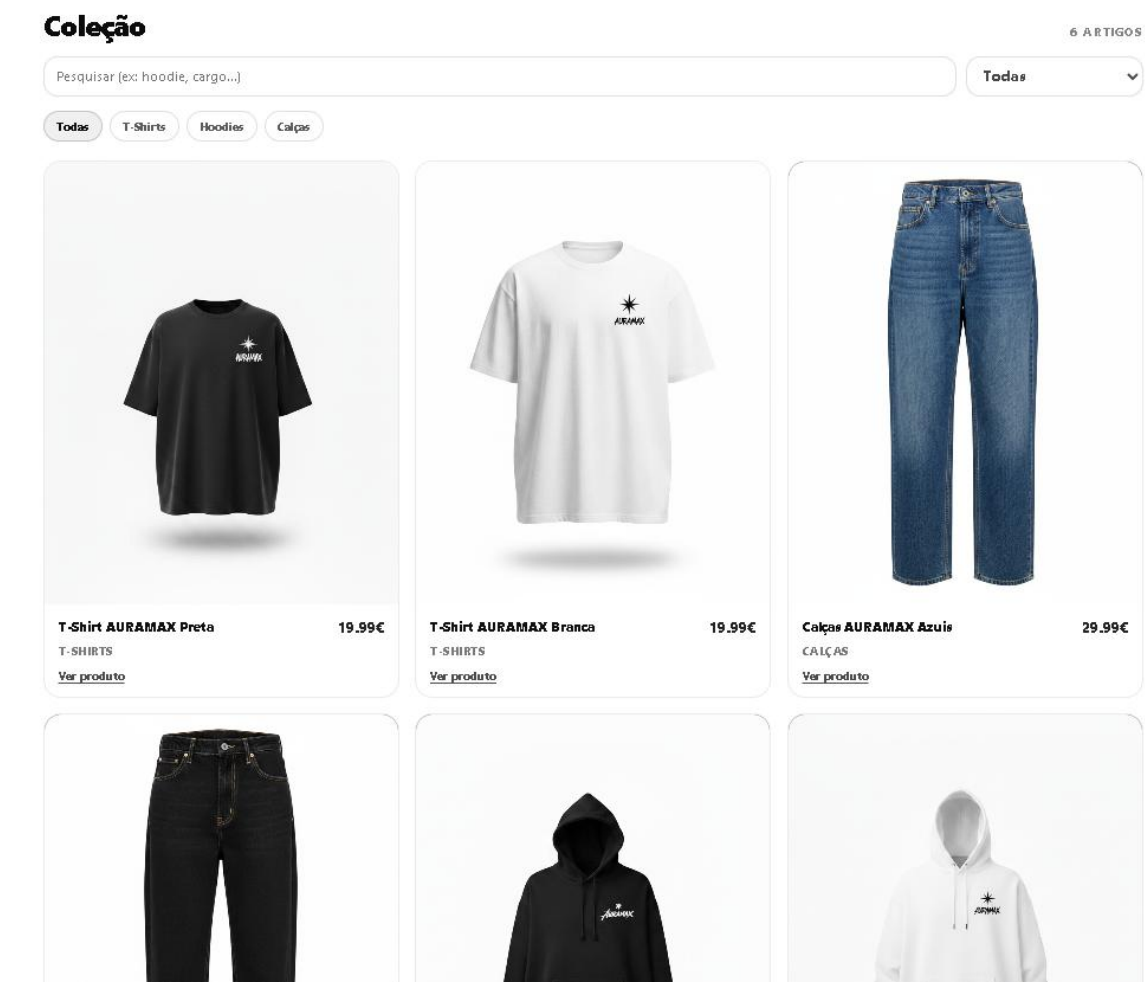


Figura 6 Coleção (catalgo)

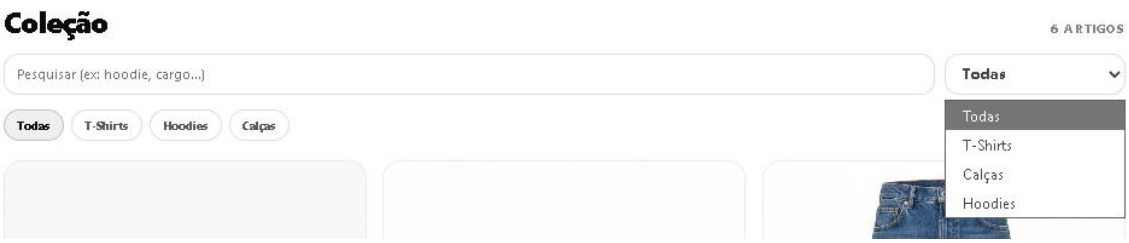


Figura 7 Categorias

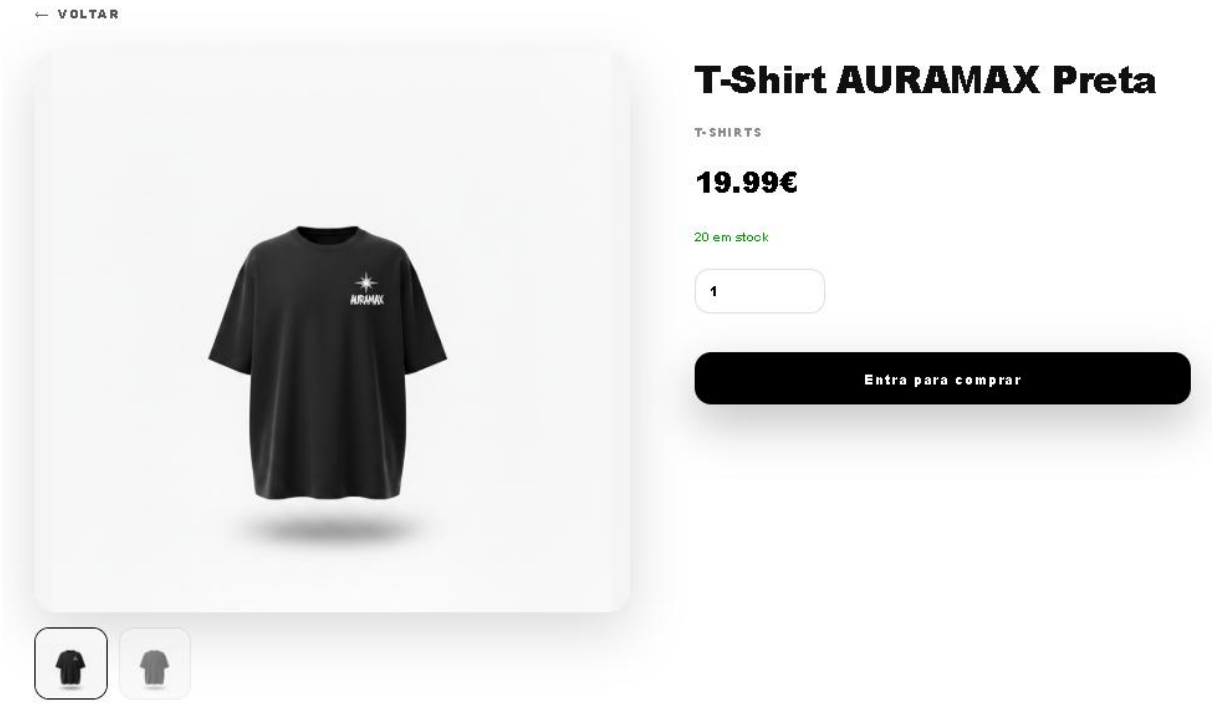


Figura 8 Detalhes do produto

Figura 9 Página saldos

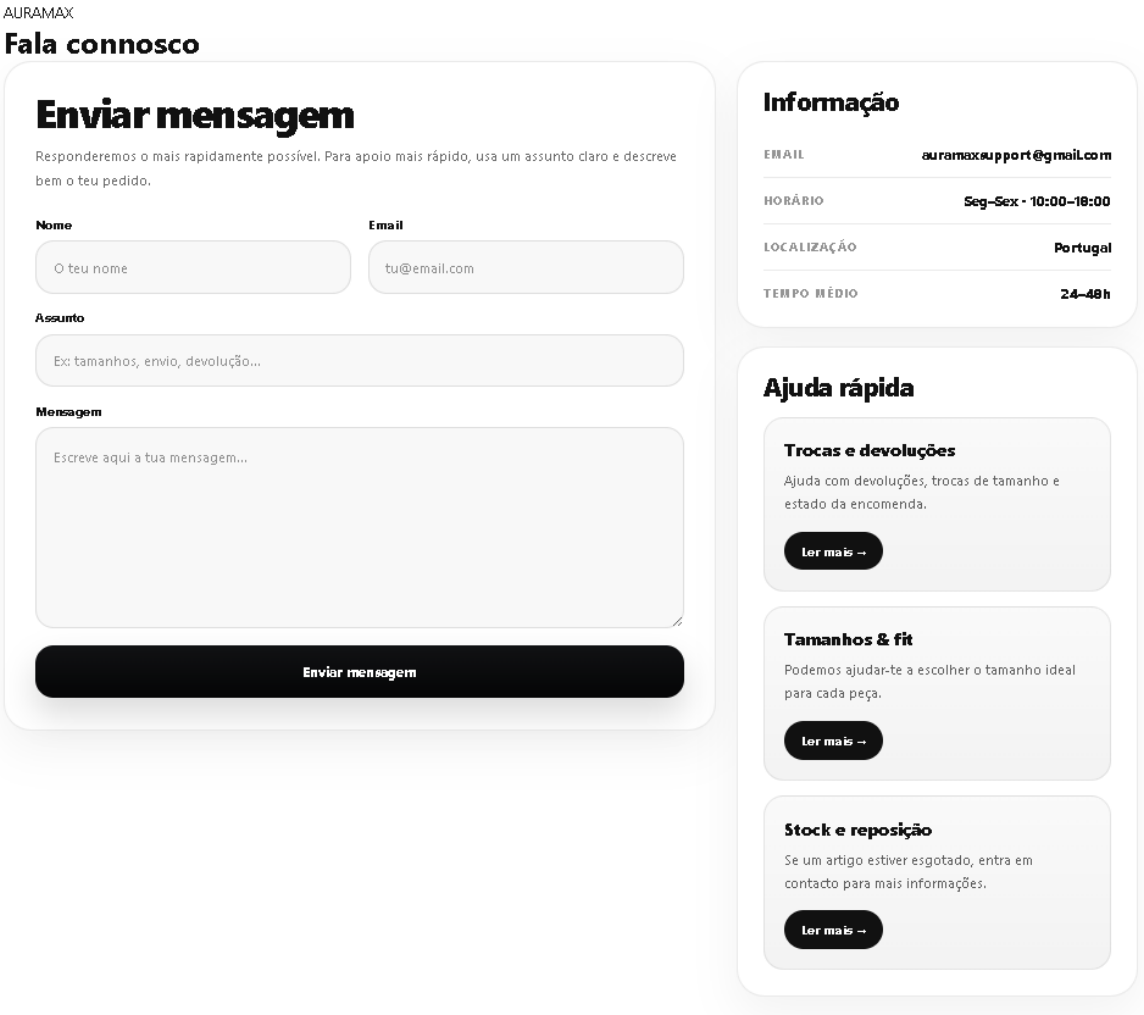


Figura 10 Página contacto

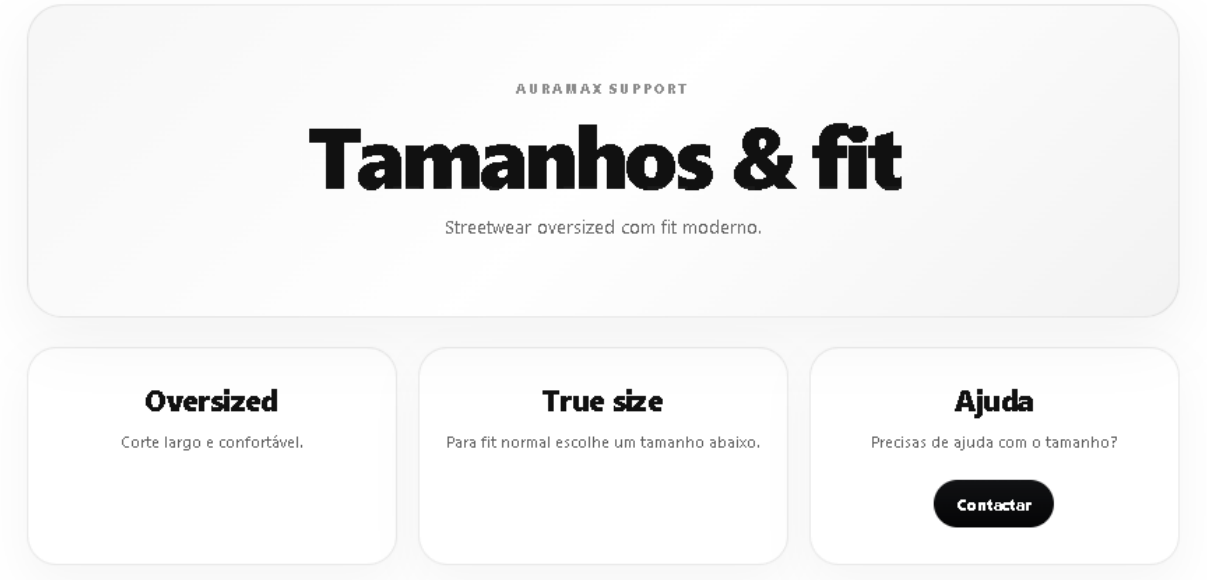


Figura 11 Ajuda rápida

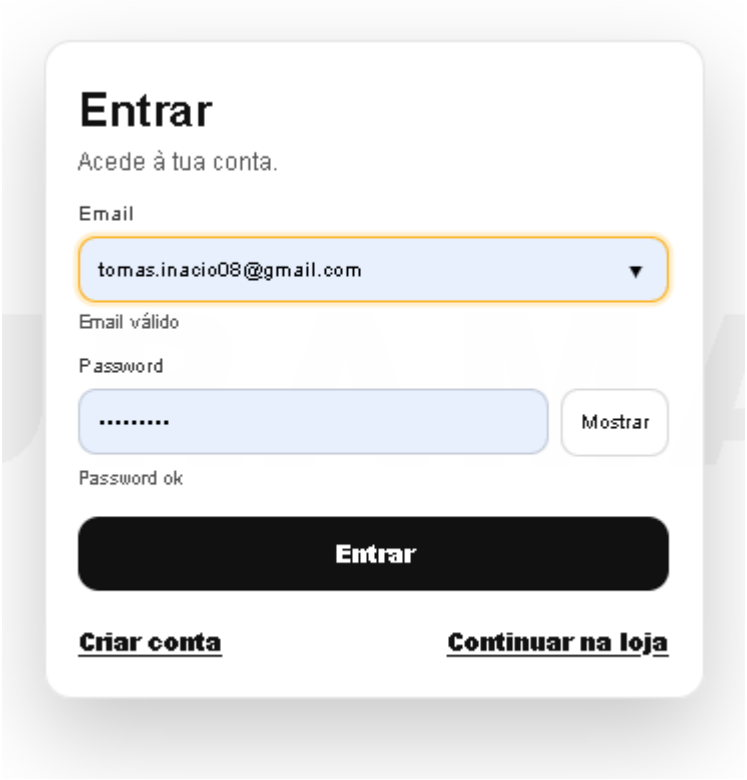


Figura 12 Login

Figura 13 Criar conta

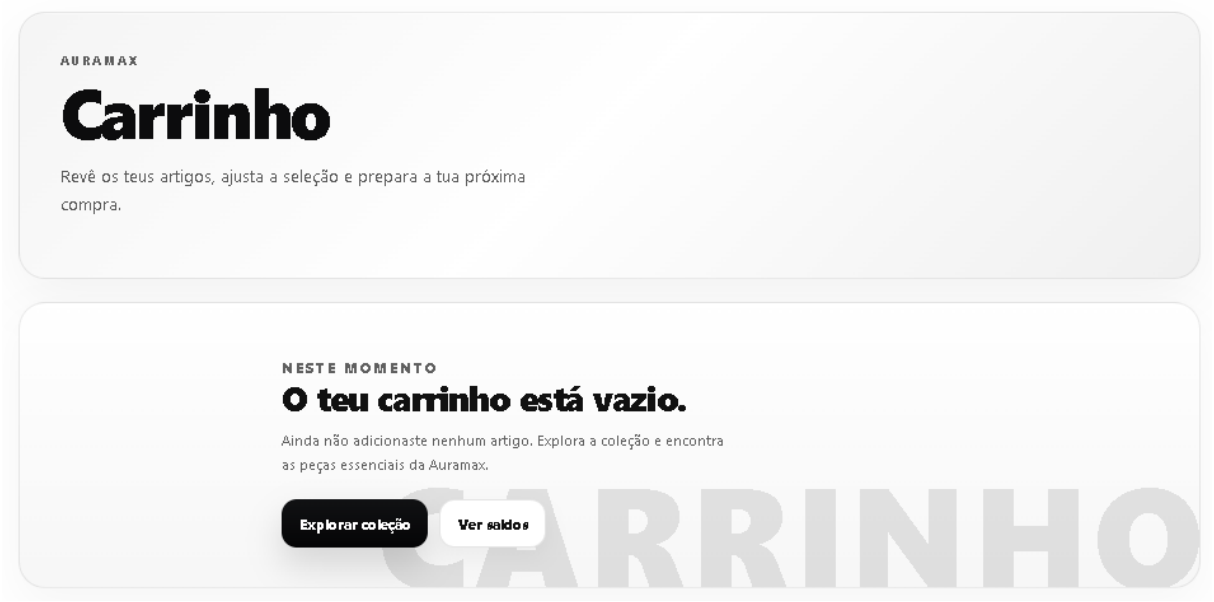


Figura 14 Carrinho

Manual do Programador

Desenvolvimento de Código

1. Enquadramento Técnico

O projeto Auramax é uma aplicação web para uma loja de roupa (*streetwear*) desenvolvida com o objetivo de simular uma loja online moderna e funcional. O sistema é *full-stack*, incluindo:

- **Front-end**: interface e experiência do utilizador, desenvolvida em React
- **Back-end**: lógica de servidor, autenticação, gestão de produtos e encomendas, desenvolvido em *Python* com *FastAPI*
- **Base de dados**: armazenamento persistente de utilizadores, produtos, imagens e encomendas, utilizando *SQLite*

Nesta fase do desenvolvimento, encontram-se implementadas as três componentes (*front-end*, *back-end* e base de dados), com comunicação real entre o cliente e a *API* através de fetch.

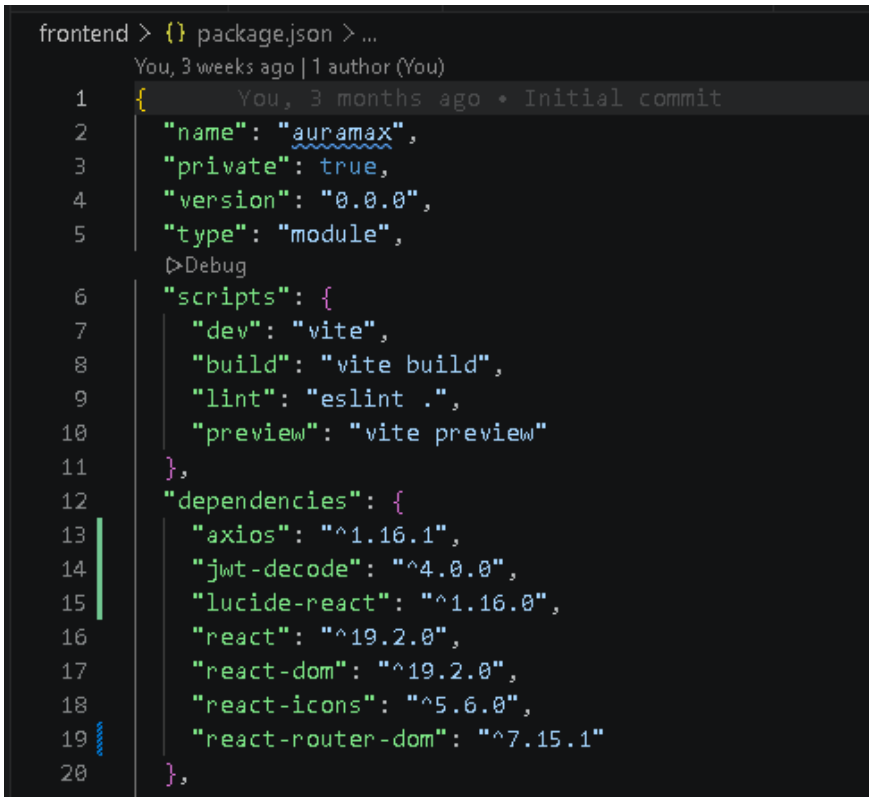


Figura 15package.json (front-end)

Este ficheiro lista as bibliotecas usadas no front-end da aplicação. Destacam-se o react e react-dom (para construir a interface), o react-router-dom (para a navegação entre páginas), o axios (para fazer pedidos à API), o jwt-decode (para ler o token de autenticação) e o react-icons/lucide-react (ícones usados na interface). Também mostra os comandos disponíveis (dev, build, lint, preview) para correr e construir o projeto.

Figura 16 main.py (back-end)

Este é o início do ficheiro principal do back-end, onde são importadas as ferramentas necessárias. O FastAPI é o framework usado para criar a API; o CORSMiddleware permite que o front-end (React) comunique com o back-end sem bloqueios de segurança; o OAuth2PasswordBearer/OAuth2PasswordRequestForm suportam o sistema de autenticação por token; o UploadFile/File permitem o upload de imagens dos produtos; e o SQLAlchemy (create_engine, Column, Integer, String, etc.) é usado para definir as tabelas e ligar à base de dados SQLite.

2. Estrutura do Projeto

A aplicação está organizada por módulos, para facilitar manutenção e evolução do código:

- App.jsx – estrutura geral e rotas
- pages/ – páginas principais (Home, Produtos, Login, Registar, Conta, Admin, etc.)
- context/ – estados globais (AuthContext para autenticação e CartContext para o carrinho)
- main.py – ponto de entrada do *back-end*, contendo modelos da base de dados, rotas da *API* e lógica de autenticação
- CSS separado por componente/página – organização visual

Esta estrutura modular permite manter o *front-end* e o *back-end* desacoplados, comunicando exclusivamente através de pedidos *HTTP* à *API*.

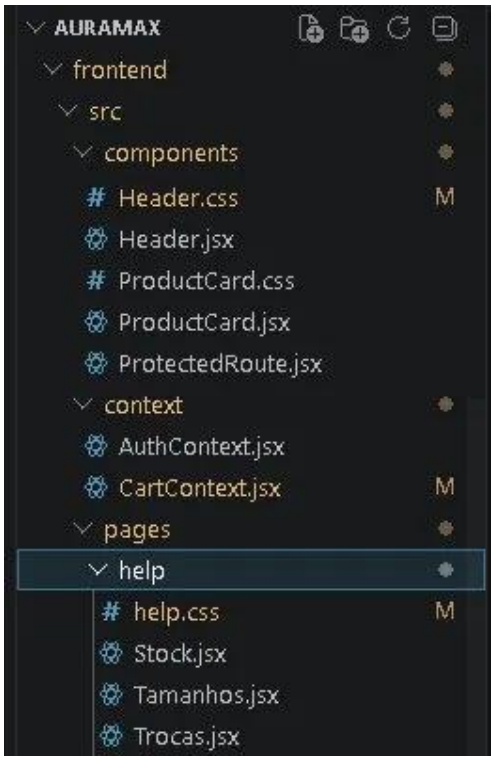


Figura 17 Páginas do front-end

Frontend/src com components/ (Header, ProductCard, ProtectedRoute), context/ (AuthContext, CartContext) e o início de pages/help/.

Figura 18 Páginas do front-end

Pasta pages/ completa, com todas as páginas (Account, AdminPanel, Cart, Contacto, Home, Login, ProductDetails, Products, Registar, Saldos, Sobre, VerifyEmail, VerifyNotice).

3. Navegação e Rotas

A navegação é feita com *React Router*, permitindo mudança de páginas sem recarregar o site.

Rotas implementadas:

- / – Home
- /produtos – Coleção de produtos
- /produtos/:id – Detalhe do produto
- /carrinho – Carrinho de compras
- /login – Início de sessão
- /registar – Criação de conta
- /verify-notice – Aviso de verificação de email
- /verify-email/:token – Confirmação de email
- /conta – Área pessoal (rota protegida)
- /admin – Painel de administração (rota protegida, apenas para administradores)
- /saldos – Promoções
- /contacto – Contacto
- /sobre – Sobre nós

- /ajuda/trocas, /ajuda/tamanhos, /ajuda/stock – Páginas de suporte

A página /conta está protegida, impedindo o acesso de utilizadores não autenticados. A página /admin tem uma camada adicional de proteção, estando disponível apenas a utilizadores com permissões de administrador (is_admin).

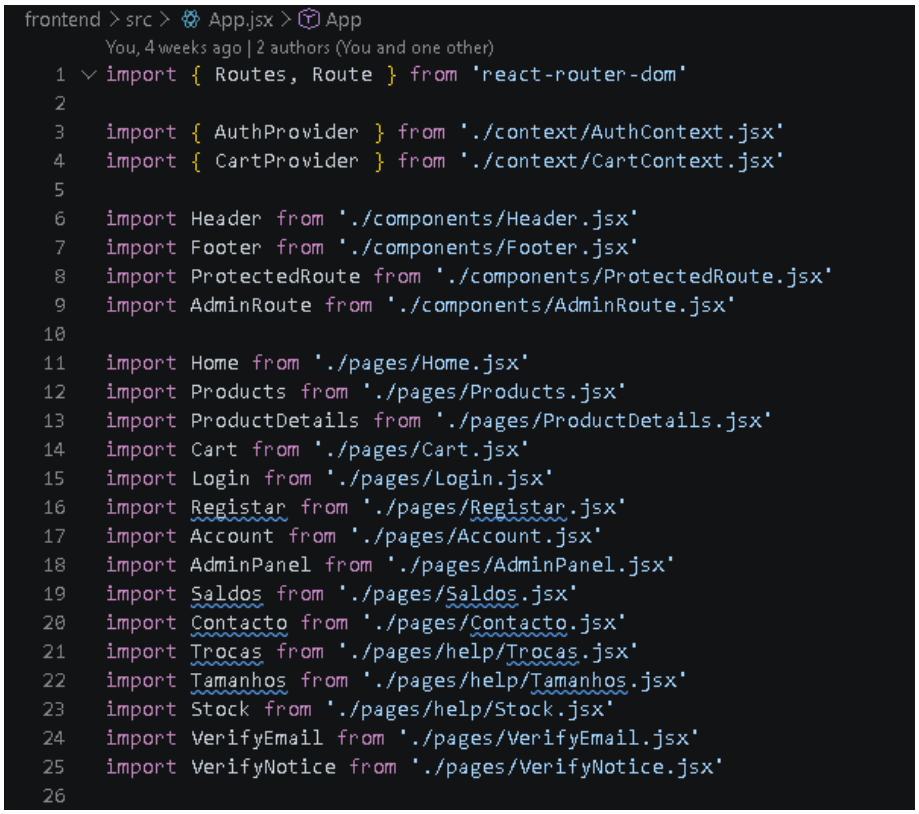


Figura 19 Imports do `App.jsx`

Esta secção do App.jsx importa todas as dependências e componentes necessários: o Routes/Route do React Router, os contextos AuthProvider e CartProvider, os componentes Header, Footer, ProtectedRoute e AdminRoute, e todas as páginas da aplicação (Home, Products, ProductDetails, Cart, Login, Registar, Account, AdminPanel, Saldos, Contacto, páginas de ajuda, VerifyEmail e VerifyNotice).

```jsx
import './App.css'

export default function App() {
  return (
    <AuthProvider>
      <CartProvider>

        <div className="app">

          <Header />

          <main className="main-content">
            <Routes>

              <Route path="/" element={<Home />} />
              <Route path="/produtos" element={<Products />} />
              <Route path="/produtos/:id" element={<ProductDetails />} />
              <Route path="/carrinho" element={<Cart />} />
              <Route path="/login" element={<Login />} />
              <Route path="/registar" element={<Registar />} />
              <Route path="/saldos" element={<Saldos />} />
              <Route path="/contacto" element={<Contacto />} />
              <Route path="/verify-email/:token" element={<VerifyEmail />} />
              <Route path="/verify-notice" element={<VerifyNotice />} />

              <Route path="/ajuda/trocas" element={<Trocas />} />
              <Route path="/ajuda/tamanhos" element={<Tamanhos />} />
              <Route path="/ajuda/stock" element={<Stock />} />

              <Route
                path="/conta"
                element={
                  <ProtectedRoute>
                    <Account />
                  </ProtectedRoute>
                }
              />
```

Figura 20 Definição das rotas no `App.jsx`

Este trecho mostra a estrutura principal da aplicação. Dentro de AuthProvider e CartProvider, o Header é mostrado em todas as páginas, e o conteúdo principal é controlado pelo Routes, que define cada rota da aplicação (Home, Coleção, Detalhe do produto, Carrinho, Login, Registo, Saldos, Contacto, páginas de verificação de email, páginas de ajuda). A rota /conta está protegida pelo componente ProtectedRoute, garantindo que só utilizadores autenticados acedem à página de Conta.

Figura 21 Rota `/admin` e fecho da estrutura do `App.jsx`

Explicação: Este trecho mostra a rota /admin, protegida pelo componente AdminRoute, que garante que apenas utilizadores com permissões de administrador acedem ao AdminPanel. Fecha-se aqui a estrutura geral da aplicação: o bloco Routes dentro de main, seguido do Footer, e o encerramento dos providers CartProvider e AuthProvider, que envolvem toda a aplicação para disponibilizar o estado do carrinho e da autenticação a todas as páginas.

Esta imagem complementa a anterior (Imagem 2 com as rotas /conta, etc.) e ilustra a frase do texto sobre a rota /admin ter "uma camada adicional de proteção, estando disponível apenas a utilizadores com permissões de administrador".

4. Componentes Essenciais

4.1 Header

O *Header* inclui:

- Nome da marca
- Menu de navegação com animações (*hover*/ativo)
- Pesquisa
- Carrinho com contador
- Secção de *login*/*logout*
- Menu *mobile* responsivo

Foi dada prioridade à aparência *clean* e profissional, com tipografia simples, cores consistentes e responsividade.

Figura 22 Estrutura do menu de navegação no Header.jsx

Este trecho mostra o menu de navegação principal (hdrNavLeft), com os links para as páginas Home, Coleção, Saldos e Contacto, usando o componente NavLink do React Router. A função linkClass aplica automaticamente um estilo diferente ao link da página atual, criando o efeito de estado ativo/hover mencionado no texto. Mostra também o nome da marca (AURAMAX) e a caixa de pesquisa (hdrSearch), que envia o termo procurado através da função onSearch.

Figura 23 Media queries de responsividade no Header.css

Media que adaptam o Header a ecrãs mais pequenos. Na primeira (até 1100px), os espaçamentos do menu e da pesquisa são reduzidos. Na segunda (até 980px), o menu de navegação, a pesquisa e parte do conteúdo do carrinho são escondidos (`display:none`), o layout do cabeçalho é reorganizado em três colunas (`grid-template-columns: auto 1fr auto`) e a marca AURAMAX é centrada — preparando o espaço para o menu hambúrguer e o menu mobile, garantindo a responsividade referida no texto.

4.2 Autenticação

Foi implementado um sistema de autenticação real, com ligação ao *back-end* e à base de dados:

- Página de *login* e de registo, com validações de email e *password*
- Registo de utilizadores via POST /auth/register, com encriptação da *password* através de *hash* (*bcrypt*)
- Verificação de email obrigatória, utilizando a plataforma *Brevo*: ao criar conta, é gerado um *token* único guardado na base de dados e enviado por email; apenas após a confirmação desse *token* o utilizador consegue iniciar sessão
- *Login* via POST /auth/login, com geração de um *token JWT* utilizado para autenticar pedidos seguintes
- AuthContext no *front-end*, responsável por guardar o *token*, o utilizador autenticado e o estado de administrador (isAdmin)

As validações implementadas no formulário incluem:

- Email obrigatório com @ e .com
- *Password* mínima de 6 caracteres

```
const login = async (email, password) => {
  if (!email || !password || password.length < 4) {
    throw new Error('Email e password (mín. 4) são obrigatórios.')
  }

  const formData = new FormData()
  formData.append('username', email)
  formData.append('password', password)

  const res = await fetch(`${API_URL}/auth/login`, {
    method: 'POST',
    body: formData,
  })

  if (!res.ok) {
    throw new Error('Credenciais inválidas')
  }

  const data = await res.json()
  const newToken = data.access_token

  // Buscar dados do utilizador
  const userRes = await fetch(`${API_URL}/users/me`, {
    headers: { 'Authorization': `Bearer ${newToken}` },
  })

  if (!userRes.ok) {
    throw new Error('Não foi possível carregar dados do utilizador')
  }

  const userData = await userRes.json()
  const newUser = {
    ...userData,
    is_admin: userData.is_admin || false
  }
```

Figura 24 Função login no AuthContext.jsx

Esta função, no front-end, é responsável por autenticar o utilizador. Valida primeiro se o email e a password foram preenchidos e têm o tamanho mínimo, depois envia um pedido POST para /auth/login com as credenciais. Se a resposta for válida, guarda o token de acesso (access_token) recebido. Em seguida, usa esse token para fazer um segundo pedido a /users/me, obtendo os dados do utilizador autenticado, incluindo o campo is_admin, que é guardado no estado como isAdmin.

```python
@app.post("/auth/register", status_code=201)      You, last month • backend
def register(
    user: UserCreate,
    db: Session = Depends(get_db)
):

    existing = db.query(User).filter(
        User.email == user.email
    ).first()

    if existing:

        raise HTTPException(
            status_code=400,
            detail="Email já registado"
        )

    verification_token = secrets.token_urlsafe(64)

    new_user = User(
        nome_user=user.nome_user,
        email=user.email,
        password_hash=hash_password(user.password),

        is_admin=False,
        is_verified=False,

        verification_token=verification_token
    )

    db.add(new_user)
    db.commit()
    db.refresh(new_user)

    print("TOKEN GERADO:", verification_token)

    send_verification_email(
        new_user.email,
        verification_token
    )
```

Figura 25 Endpoint POST /auth/register no main.py

Este endpoint do back-end trata o registo de novos utilizadores. Primeiro verifica se já existe uma conta com o mesmo email, recusando o registo caso exista. Depois gera um token de verificação único (verification_token), cria o novo utilizador na base de dados com a password encriptada (hash_password) e os campos is_admin/is_verified a False. Por fim, envia um email de confirmação ao utilizador através da função send_verification_email, contendo esse token, conforme descrito no texto sobre a verificação obrigatória de email via Brevo.

```python
@app.post("/auth/login", response_model=Token)      You, last month • backend
def login(
    form: OAuth2PasswordRequestForm = Depends(),
    db: Session = Depends(get_db)
):

    user = db.query(User).filter(
        User.email == form.username
    ).first()

    if not user:

        raise HTTPException(
            status_code=401,
            detail="Credenciais inválidas"
        )

    if not verify_password(
        form.password,
        user.password_hash
    ):

        raise HTTPException(
            status_code=401,
            detail="Credenciais inválidas"
        )

    if not user.is_verified:

        raise HTTPException(
            status_code=401,
            detail="Confirma primeiro o teu email"
        )

    token = create_access_token({
        "sub": user.email
    })

    return {
        "access_token": token,
        "token_type": "bearer"
    }
```

Figura 26 Endpoint POST /auth/login no main.py

Este endpoint trata o início de sessão. Verifica se o email existe na base de dados e se a password corresponde ao hash guardado, devolvendo erro "Credenciais inválidas" caso contrário. Verifica também se o email do utilizador já foi confirmado (is_verified), bloqueando o login com a mensagem "Confirma primeiro o teu email" caso não tenha sido. Se tudo estiver correto, gera um token JWT (create_access_token) e devolve-o ao front-end, que o usará para autenticar pedidos seguintes.

4.3 Carrinho

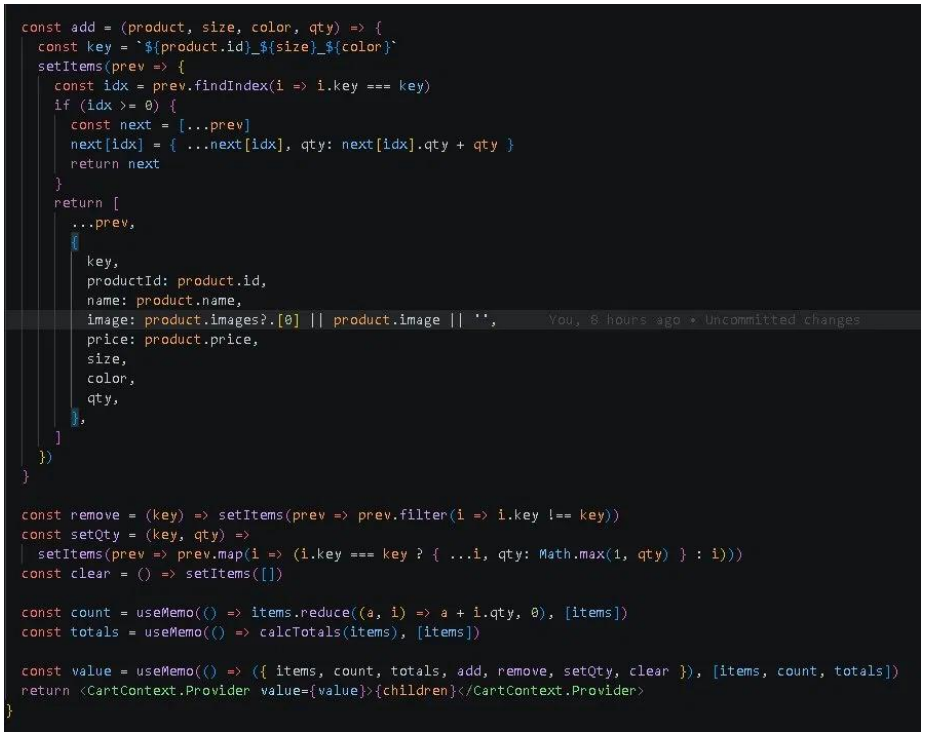
O carrinho é gerido através de CartContext, permitindo manter o estado do carrinho disponível em toda a aplicação e atualizar o contador no *Header*.

No processo de *checkout*, o carrinho está ligado à *API* real:

- É criada uma encomenda através de POST /orders, associada ao utilizador autenticado

- O *stock* dos produtos é automaticamente descontado na base de dados
- É enviado um email de confirmação da encomenda para o utilizador, através de POST /send-order-email

Atualmente, os itens do carrinho mantêm-se apenas em memória durante a sessão (através do CartContext); a persistência do carrinho por utilizador na base de dados, antes do *checkout*, é um desenvolvimento futuro.



```
const add = (product, size, color, qty) => {
  const key = `${product.id}_${size}_${color}`
  setItems(prev => {
    const idx = prev.findIndex(i => i.key === key)
    if (idx >= 0) {
      const next = [...prev]
      next[idx] = { ...next[idx], qty: next[idx].qty + qty }
      return next
    }
    return [
      ...prev,
      {
        key,
        productId: product.id,
        name: product.name,
        image: product.images?.[0] || product.image || '',      You, 8 hours ago • Uncommitted changes
        price: product.price,
        size,
        color,
        qty,
      },
    ]
  })
}

const remove = (key) => setItems(prev => prev.filter(i => i.key !== key))
const setQty = (key, qty) =>
  setItems(prev => prev.map(i => (i.key === key ? { ...i, qty: Math.max(1, qty) } : i)))
const clear = () => setItems([])

const count = useMemo(() => items.reduce((a, i) => a + i.qty, 0), [items])
const totals = useMemo(() => calcTotals(items), [items])

const value = useMemo(() => ({ items, count, totals, add, remove, setQty, clear }), [items, count, totals])
return <CartContext.Provider value={value}>{children}</CartContext.Provider>
}
```

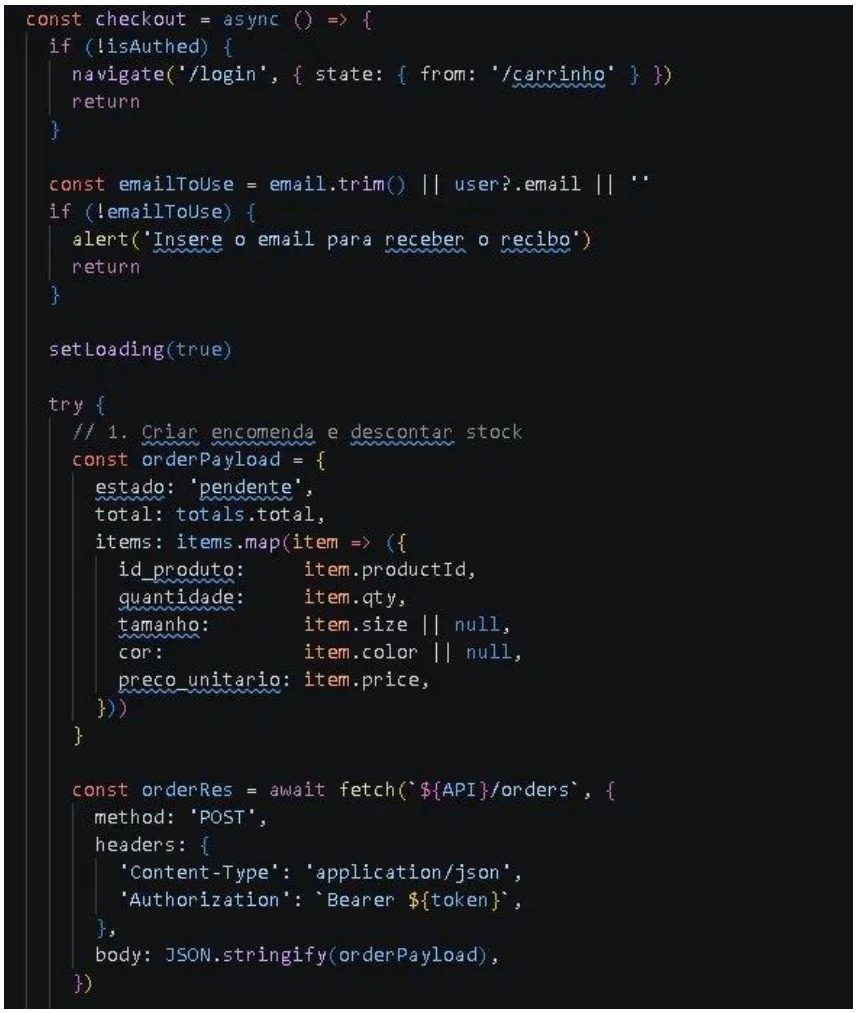
Figura 27 Funções de gestão do carrinho no CartContext.jsx

Este trecho define as principais operações do carrinho. A função add adiciona um produto (combinando id, tamanho e cor numa chave única); se o mesmo artigo já existir, soma a quantidade em vez de duplicar a linha. Existem ainda remove (retira um item), setQty (altera a quantidade), e clear (esvazia o carrinho). Os valores count e totals são calculados automaticamente com useMemo, sendo count o número total de artigos — usado para o contador no Header — e totals o subtotal/total calculado pela função calcTotals.

```jsx
import React, { createContext, useContext, useEffect, useMemo, useState } from 'react'

const CartContext = createContext(null)
const KEY = 'auramax_cart'

function calcTotals(items) {
  const subtotal = items.reduce((a, i) => a + i.price * i.qty, 0)
  const shipping = subtotal >= 60 || subtotal === 0 ? 0 : 4.9
  return { subtotal, shipping, total: subtotal + shipping }
}

export function CartProvider({ children }) {
  const [items, setItems] = useState([])

  useEffect(() => {
    const raw = localStorage.getItem(KEY)
    if (raw) setItems(JSON.parse(raw))
  }, [])

  useEffect(() => {
    localStorage.setItem(KEY, JSON.stringify(items))
  }, [items])
```

Figura 28 Configuração do CartContext.jsx e persistência local

Esta parte mostra a criação do contexto (CartContext) e a função calcTotals, que calcula o subtotal, o custo de envio (gratuito a partir de 60€) e o total. O CartProvider guarda o estado dos itens (items) e usa dois useEffect: um para carregar o carrinho guardado no localStorage ao iniciar, e outro para o atualizar sempre que os itens mudam — garantindo que o estado do carrinho fica disponível em toda a aplicação.



```
const checkout = async () => {
  if (!isAuthed) {
    navigate('/login', { state: { from: '/carrinho' } })
    return
  }

  const emailToUse = email.trim() || user?.email || ''
  if (!emailToUse) {
    alert('Insere o email para receber o recibo')
    return
  }

  setLoading(true)

  try {
    // 1. Criar encomenda e descontar stock
    const orderPayload = {
      estado: 'pendente',
      total: totals.total,
      items: items.map(item => ({
        id_produto:     item.productId,
        quantidade:     item.qty,
        tamanho:        item.size || null,
        cor:            item.color || null,
        preco_unitario: item.price,
      }))
    }

    const orderRes = await fetch(`${API}/orders`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`,
      },
      body: JSON.stringify(orderPayload),
    })
```

Figura 29 Início da função checkout no Cart.jsx

Esta função inicia o processo de finalização de compra. Primeiro verifica se o utilizador está autenticado (caso contrário, redireciona para o login) e se existe um email para receção do recibo. Depois monta o orderPayload com os dados da encomenda (estado, total e lista de itens com produto, quantidade, tamanho, cor e preço) e envia-o por POST para /orders, com o token de autenticação no cabeçalho.

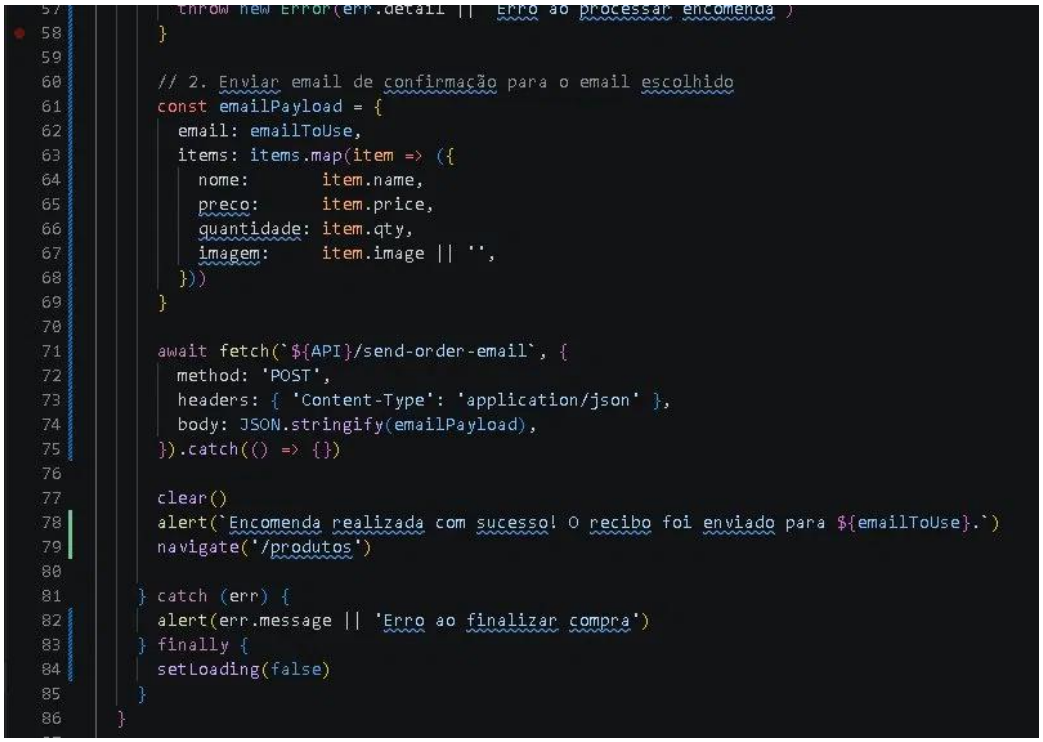


Figura 30 Conclusão da função checkout

Após a encomenda ser criada com sucesso, esta parte monta o emailPayload com os itens comprados e envia-o por POST para /send-order-email, para que o utilizador receba a confirmação por email. Em seguida, o carrinho é esvaziado (clear()), é mostrada uma mensagem de sucesso e o utilizador é redirecionado para a página de produtos. Em caso de erro em qualquer passo, é mostrada uma mensagem de erro ao utilizador.

```python
@app.post("/orders", status_code=201)
def create_order(
    data: EncomendaCreate,
    current_user: User = Depends(get_current_user),
    db: Session = Depends(get_db)
):
    for item in data.items:
        produto = db.query(Produto).filter(
            Produto.id_produto == item.id_produto
        ).first()
        if not produto:
            raise HTTPException(status_code=404, detail=f"Produto {item.id_produto} não encontrado")
        if produto.stock < item.quantidade:
            raise HTTPException(status_code=400, detail=f"Stock insuficiente para '{produto.nome_produto}'")

    new_order = Encomenda(
        id_user=current_user.id_user,
        estado=data.estado,
        total=data.total,
        data_encomenda=datetime.utcnow(),
    )
    db.add(new_order)
    db.commit()
    db.refresh(new_order)

    for item in data.items:
        produto = db.query(Produto).filter(Produto.id_produto == item.id_produto).first()
        produto.stock -= item.quantidade

    db.commit()
    return {"msg": "Encomenda criada", "id_encomenda": new_order.id_encomenda}
```

Figura 31 Endpoint POST /orders no main.py

Este endpoint do back-end cria a encomenda associada ao utilizador autenticado (current_user). Primeiro percorre os itens recebidos, verificando se cada produto existe e se há stock suficiente, devolvendo erro caso contrário. Depois cria o registo da encomenda (Encomenda) na base de dados. Por fim, percorre novamente os itens para descontar automaticamente o stock de cada produto (produto.stock -= item.quantidade), confirmando as alterações na base de dados.

4.4 Painel de Administração

Foi desenvolvido um painel de administração (/admin), acessível apenas a utilizadores com permissões de administrador:

- Criação de novos produtos, com nome, categoria, *stock* e preço
- *Upload* e organização de imagens associadas a cada produto, incluindo reordenação e remoção
- Remoção de produtos diretamente na página de Coleção, visível apenas a administradores

```
if (!isAdmin) {
  return (
    <div className="admin-panel">
      <div className="admin-error">
        <h2>Acesso Negado</h2>
        <p>Apenas administradores podem aceder a esta página.</p>
        <button onClick={() => navigate('/')}>Voltar à Home</button>
      </div>
    </div>
  )
}
```

Figura 32 Restrição de acesso ao painel de administração (AdminPanel.jsx)

Este trecho garante que apenas administradores podem aceder ao painel. Se o utilizador autenticado não tiver permissões de administrador (!isAdmin), é mostrada uma mensagem de "Acesso Negado" com um botão para voltar à Home, em vez do formulário de gestão de produtos.

```
const handleSubmit = async (e) => {

e.preventDefault()

setError('')
setMessage('')
setLoading(true)
    You, last month • updates …
try {

  // 1. Criar produto
  const payload = {

    nome_produto: formData.nome_produto,

    categoria: formData.categoria || null,

    stock: parseInt(formData.stock) || 0,

    preco: parseFloat(formData.preco),
  }

  const res = await fetch(`${API_URL}/products`, {

    method: 'POST',

    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${token}`,
    },

    body: JSON.stringify(payload),
  })

  if (!res.ok) {

    const data = await res.json()

    throw new Error(
      data.detail || 'Erro ao criar produto'
    )
  }
```

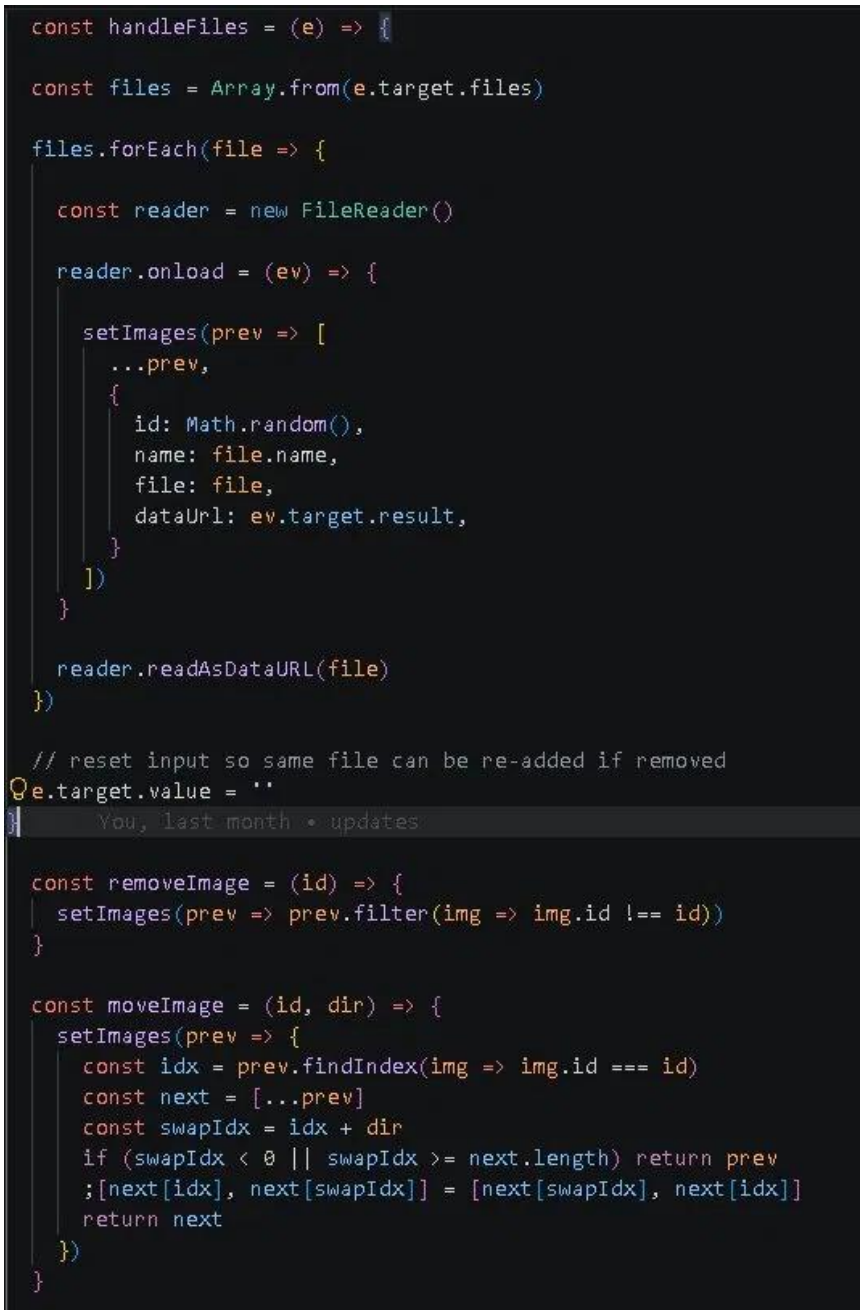
Figura 33 Criação de produto — handleSubmit (AdminPanel.jsx)

Esta função trata a submissão do formulário de novo produto. Monta o objeto payload com o nome, categoria, stock e preço introduzidos, e envia-o por POST para /products, com o token de autenticação no cabeçalho. Se a resposta não for bem-sucedida, é lançado um erro com a mensagem devolvida pela API.

```
if (!res.ok) {

  const data = await res.json()

  throw new Error(
    data.detail || 'Erro ao criar produto'
  )
}

const data = await res.json()

const id_produto = data.id_produto

// 2. Upload imagens
if (images.length > 0) {

  const uploadData = new FormData()

  images.forEach((img) => {
    uploadData.append('files', img.file)
  })

  const imgRes = await fetch(
    `${API_URL}/products/${id_produto}/images`,
    {

      method: 'POST',

      headers: {
        'Authorization': `Bearer ${token}`,
      },

      body: uploadData,
    }
  )
```

Figura 34 Upload de imagens do produto — continuação do handleSubmit (AdminPanel.jsx)

Após a criação do produto, obtém-se o id_produto devolvido pela API. Se houverem imagens selecionadas, são adicionadas a um FormData e enviadas por POST para /products/{id_produto}/images, associando-as ao produto recém-criado.



```
const handleFiles = (e) => {

const files = Array.from(e.target.files)

files.forEach(file => {

  const reader = new FileReader()

  reader.onload = (ev) => {

    setImages(prev => [
      ...prev,
      {
        id: Math.random(),
        name: file.name,
        file: file,
        dataUrl: ev.target.result,
      }
    ])
  }

  reader.readAsDataURL(file)
})

// reset input so same file can be re-added if removed
e.target.value = ''
}      You, last month • updates

const removeImage = (id) => {
  setImages(prev => prev.filter(img => img.id !== id))
}

const moveImage = (id, dir) => {
  setImages(prev => {
    const idx = prev.findIndex(img => img.id === id)
    const next = [...prev]
    const swapIdx = idx + dir
    if (swapIdx < 0 || swapIdx >= next.length) return prev
    ;[next[idx], next[swapIdx]] = [next[swapIdx], next[idx]]
    return next
  })
}
```

Figura 35 Gestão de imagens — handleFiles, removeImage e moveImage (AdminPanel.jsx)

A função handleFiles lê cada ficheiro selecionado e adiciona-o à lista de imagens (setImages), guardando o nome, o ficheiro e uma pré-visualização (dataUrl). A função removeImage retira uma imagem da lista pelo seu id, e moveImage permite reordenar as imagens, trocando a posição de duas imagens consecutivas — implementando a organização e reordenação de imagens referida no texto.

```
const deleteProduct = async (id) => {

  const confirmDelete = window.confirm(
    'Tens a certeza que queres apagar este produto?'
  )

  if (!confirmDelete) return

  try {

    const res = await fetch(
      `http://127.0.0.1:8000/products/${id}`,
      {
        method: 'DELETE',
        headers: {
          Authorization: `Bearer ${token}`
        }
      }
    )

    if (!res.ok) {
      throw new Error('Erro ao apagar produto')
    }

    setProducts(prev =>
      prev.filter(p => p.id !== id)
    )

  } catch (err) {

    alert(err.message)
  }
}
```

Figura 36 Remoção de produtos — deleteProduct (Products.jsx)

Esta função permite a um administrador apagar um produto diretamente na página de Coleção. Após confirmação do utilizador, envia um pedido DELETE para /products/{id} com o token de autenticação. Se for bem-sucedido, o produto é removido da lista mostrada na página (setProducts); caso contrário, é mostrado um alerta com o erro.

5. Normas e Boas Práticas

Durante a implementação foram seguidas boas práticas:

- Separação entre interface e lógica de negócio
- Componentização e reutilização de código
- Organização modular de ficheiros, tanto no *front-end* como no *back-end*
- Convenções padrão (*PascalCase*, *camelCase*)
- Código legível e consistente
- Responsividade com *media queries*
- Feedback visual (animações suaves, estados de *hover*, validação em tempo real de formulários)

- Configuração de *CORS* no *back-end*, permitindo a comunicação segura entre o *front-end* e a *API*

6. Estado final do projeto

Ao longo da realização deste projeto foi implementado:

- Estrutura completa do *front-end*, com sistema de rotas e páginas principais
- *Header* responsivo com animações
- *Back-end* em *FastAPI*, com base de dados *SQLite*
- Sistema de autenticação real (registo, *login*, *JWT*, *hash* de *passwords*)
- Verificação de email via *Brevo*, com *token* único por utilizador
- *CRUD* de produtos, incluindo *upload* e gestão de imagens
- Painel de administração protegido
- Carrinho funcional, com *checkout* real e desconto automático de *stock*
- Sistema de encomendas, com registo na base de dados e email de confirmação
- Formulário de contacto, com envio de mensagem por email
- Páginas institucionais (*Sobre*, *Saldos*) e de suporte (*Trocas*, *Tamanhos*, *Stock*)
- Rotas protegidas, tanto para utilizadores autenticados como para administradores

7. Desenvolvimentos Futuros

Para completar e melhorar o projeto, ficam identificados os seguintes pontos:

- Tornar a página promoções operacional

Manual de Utilizador

Introdução

Este manual destina-se a orientar os utilizadores da plataforma Auramax na utilização das suas funcionalidades, abrangendo tanto utilizadores finais como administradores. Aqui encontram-se instruções passo a passo sobre como navegar, registar conta, comprar produtos e, no caso dos administradores, gerir o catálogo da loja.

Página inicial (Home)

Ao aceder ao website, o utilizador é recebido pela página inicial, que apresenta um vídeo de destaque com a identidade visual da marca, juntamente com o slogan "A nova visão do streetwear em Portugal começa aqui".

A partir desta página é possível aceder diretamente à coleção completa de produtos através do botão "Ver coleção", consultar a página de promoções ("Saldos") e navegar pelas categorias em destaque (T-Shirts, Hoodies, Calças), que direcionam para a página de produtos já filtrada pela categoria escolhida.

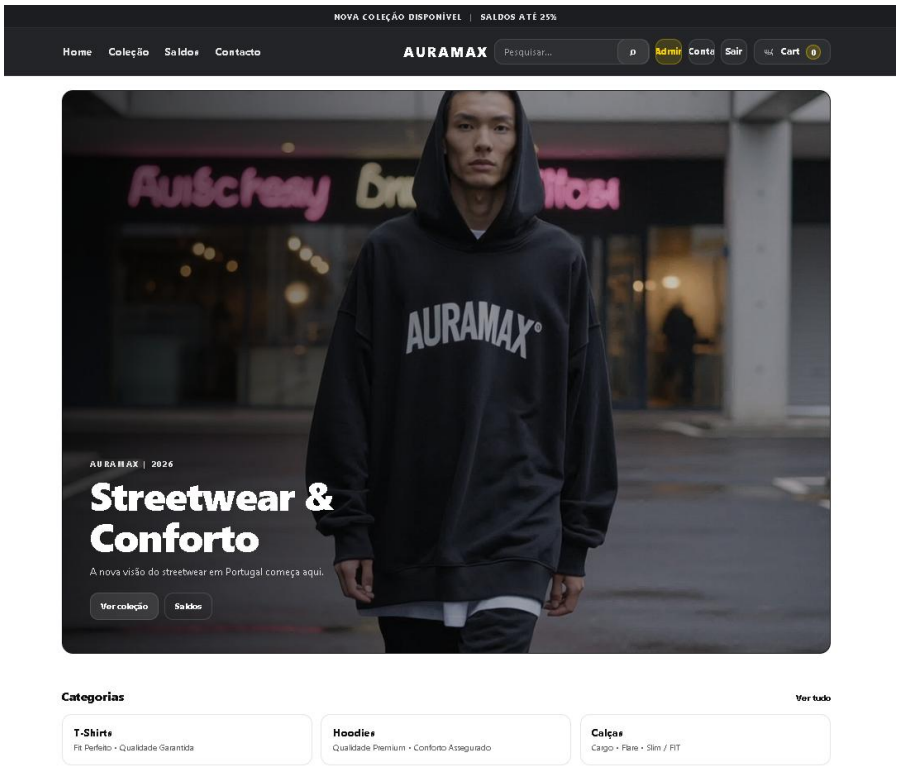


Figura 37 Separador Home

Navbar (barra de navegação)

A barra de navegação está presente em todas as páginas do site e mantém-se fixa no topo, garantindo acesso rápido às principais secções.

O logótipo Auramax, ao ser clicado, redireciona para a página inicial. O menu de navegação inclui links para Home, Coleção, Saldos e Contacto, com efeitos visuais ao passar o cursor e destaque da página ativa. Existe ainda um campo de pesquisa rápida de produtos, um ícone de carrinho que mostra o número de artigos adicionados, e uma secção de sessão: o botão "Entrar" para utilizadores não autenticados, ou o acesso à área pessoal e o botão de logout para utilizadores já autenticados.

Em ecrãs mais pequenos (tablets e smartphones), o menu adapta-se automaticamente a um formato mobile, mantendo todas as funcionalidades acessíveis.

Figura 38 Navbar

Criar conta e iniciar sessão

Registo

Para criar uma conta, o utilizador acede à página /registar e preenche o formulário com nome (mínimo de 2 caracteres), email (deve conter "@" e terminar em ".com") e password (mínimo de 6 caracteres).

Durante o preenchimento, o formulário apresenta validação em tempo real, indicando se o email e a password introduzidos são válidos. O botão "Criar conta" só fica disponível quando todos os campos são válidos.

Após submeter o formulário, o utilizador é redirecionado para a página de aviso de verificação (/verify-notice), onde é informado que foi enviado um email de confirmação para a conta criada.

Footer (rodapé)

O rodapé está presente em todas as páginas do site e localiza-se na parte inferior, fornecendo informações complementares sobre a marca.

O logótipo AURAMAX é apresentado em destaque, acompanhado da frase "Streetwear moderna com identidade premium", reforçando a identidade visual da marca. O Footer inclui ainda ícones de acesso às redes sociais Instagram e Facebook, permitindo aos utilizadores acompanhar conteúdos, novidades e futuras atualizações da marca.

O design foi desenvolvido de forma simples e minimalista, mantendo a consistência visual com o restante website. A sua estrutura adapta-se automaticamente a diferentes tamanhos de

ecrã, garantindo uma apresentação organizada e acessível em computadores, tablets e smartphones.

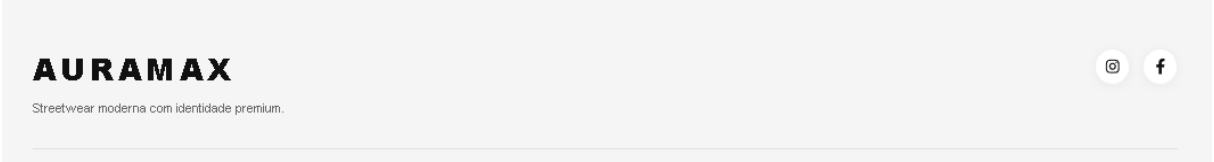


Figura 39 Footer (rodapé)

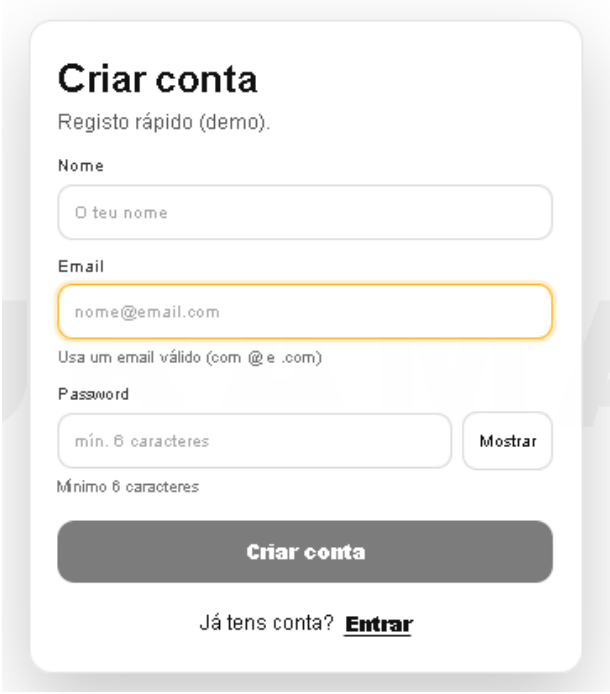


Figura 40 Criar Conta

Verificação de email

O utilizador recebe um email de confirmação enviado através da plataforma Brevo. Ao clicar no link recebido, é direcionado para a página /verify-email/:token, onde o sistema valida automaticamente o token através do back-end.

Se o token for válido, é apresentada a mensagem "Email confirmado" e a conta fica ativada, sendo disponibilizado um botão para aceder à página de login. Se o link for inválido ou tiver expirado, é apresentada a mensagem "Link inválido".

Apenas após a confirmação do email é possível iniciar sessão na plataforma.

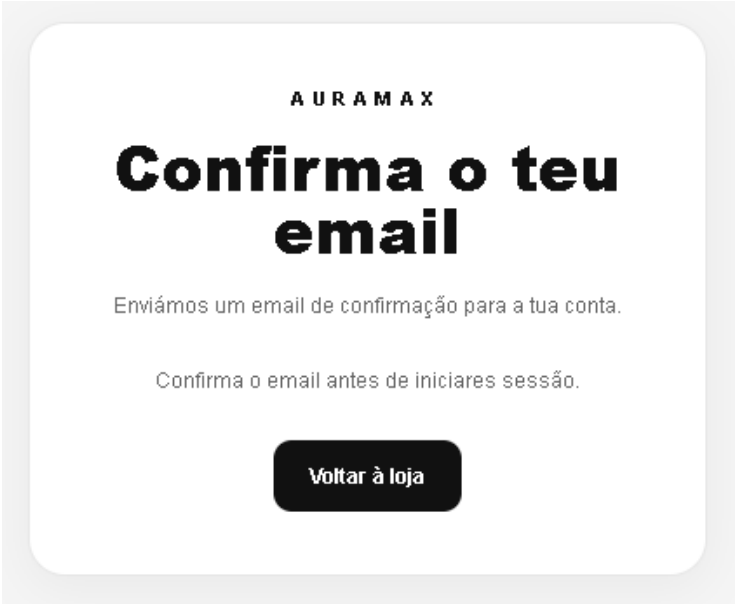


Figura 41 Confirmar Email

Login

Na página /login, o utilizador introduz o seu email e password. O formulário inclui validação em tempo real do email e da password, a opção para mostrar ou ocultar a password, sugestões de emails utilizados anteriormente através de autocompletar, e uma mensagem de erro caso as credenciais estejam incorretas ou a conta ainda não tenha sido verificada.

Após um login bem-sucedido, o utilizador é redirecionado para a área pessoal (/conta) ou para a página onde estava antes de ser solicitado o login, por exemplo ao tentar adicionar um produto ao carrinho sem estar autenticado.

Figura 42 Login

Página de produtos (Coleção)

Na página /produtos, o utilizador encontra a coleção completa de artigos disponíveis. Esta página permite pesquisar produtos pelo nome ou categoria através do campo de procura, filtrar por categoria utilizando o menu suspenso ou os atalhos rápidos (Todas, T-Shirts, Hoodies, Calças), e visualizar o número total de artigos correspondentes à pesquisa ou filtro aplicado.

Cada produto é apresentado num cartão com imagem, nome, categoria e preço. Ao passar o cursor sobre a imagem, é mostrada uma segunda fotografia do produto, caso exista, permitindo ver diferentes ângulos sem sair da página.

Clicar em qualquer produto abre a respetiva página de detalhe.

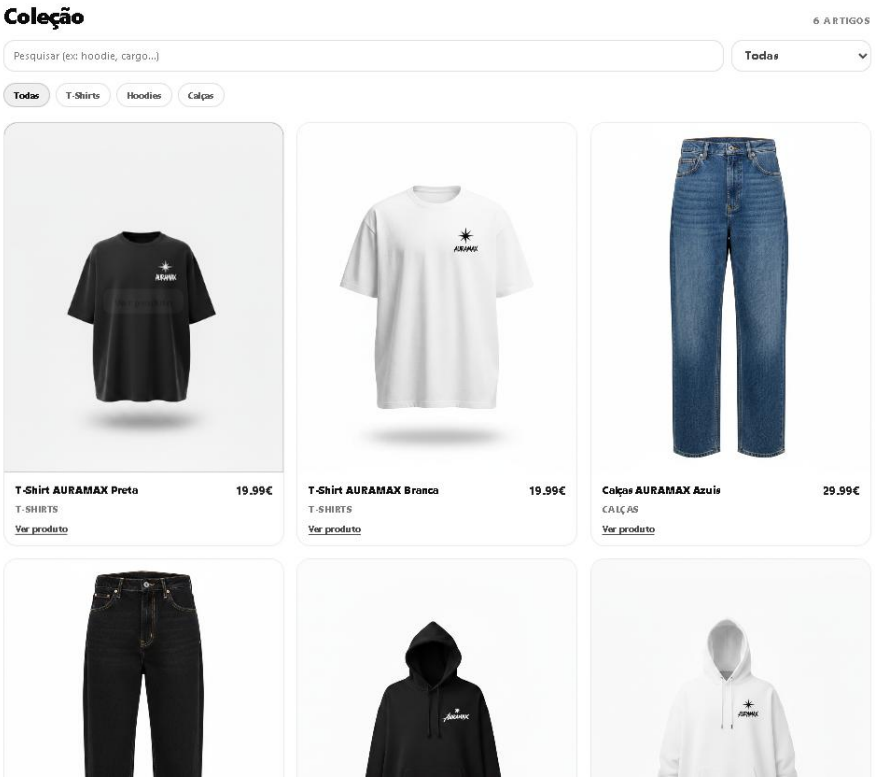


Figura 43 Coleção (catalgo roupa)

Apagar produto (apenas administradores)

Quando um administrador tem sessão iniciada, é apresentado um botão "Apagar" em cada cartão de produto. Ao clicar, é pedida confirmação antes de o produto ser removido permanentemente da base de dados.

Categorias

A secção de categorias encontra-se disponível na página inicial e permite ao utilizador aceder rapidamente aos diferentes tipos de produtos disponíveis na loja.

São apresentadas três categorias principais: T-Shirts, Hoodies e Calças, cada uma acompanhada por uma breve descrição das suas características. Ao clicar numa categoria, o utilizador é automaticamente redirecionado para a página de produtos com o respetivo filtro aplicado, facilitando a navegação e a pesquisa de artigos específicos.

Existe ainda a opção "Ver tudo", que permite visualizar a coleção completa sem qualquer filtro ativo. O design desta secção segue a identidade visual da plataforma, utilizando cartões simples e organizados que proporcionam uma navegação rápida, intuitiva e agradável.

Figura 44 Categorias

Figura 45 Apagar produto

Detalhe do produto

Ao selecionar um produto, o utilizador acede a uma página com informação detalhada, composta por uma galeria de imagens com imagem principal e miniaturas clicáveis para alternar entre as várias fotografias do produto, o nome, categoria e preço, e uma indicação de stock apresentada com cores diferentes consoante a disponibilidade: stock normal em verde, stock baixo com a mensagem "Apenas X em stock" em laranja quando restam 5 ou menos unidades, ou "Esgotado" em vermelho.

A página inclui também a descrição do produto, a seleção de tamanho e cor quando aplicável, e um campo de quantidade limitado ao stock disponível.

Adicionar ao carrinho

Se o utilizador não tiver sessão iniciada, ao clicar em "Entra para comprar" é redirecionado para a página de login, voltando depois automaticamente à página do produto. Se o utilizador tiver sessão iniciada, ao clicar em "Adicionar ao carrinho" o produto é adicionado e o utilizador é encaminhado diretamente para o carrinho. Se o produto estiver esgotado, o botão fica desativado e indica "Esgotado".

Figura 46 Finalizar compra

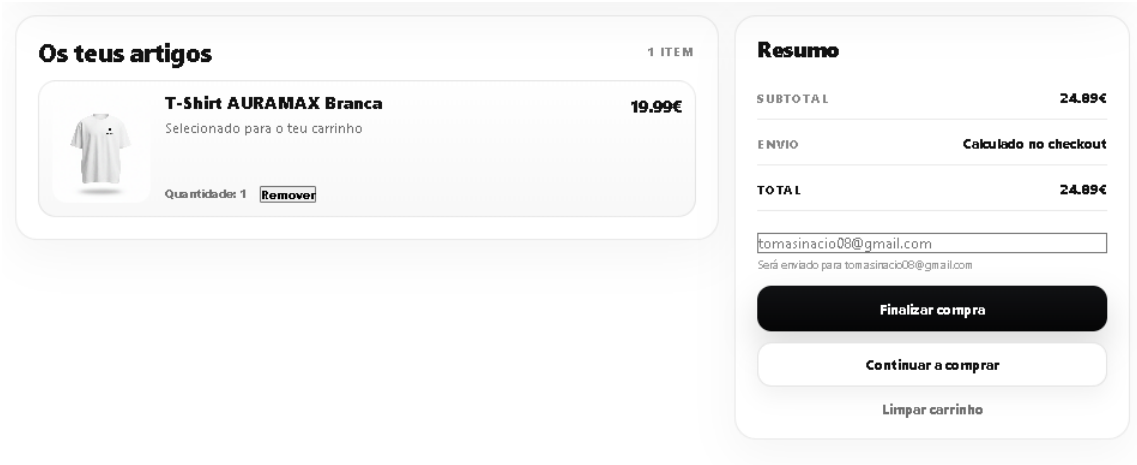


Figura 47 Finalizar Compra

Carrinho de compras

Na página /carrinho, o utilizador pode consultar todos os artigos adicionados, com nome, imagem, preço e quantidade.

Gerir o carrinho

É possível remover um artigo individualmente através do botão "Remover", ou limpar o carrinho por completo de uma vez. O resumo apresenta o subtotal e o total da compra, atualizados automaticamente.

Finalizar compra

Antes de finalizar, o utilizador pode indicar um email para receção do recibo da encomenda, sendo por defeito utilizado o email da conta. Ao clicar em "Finalizar compra", é criada uma encomenda associada ao utilizador autenticado, o stock dos produtos comprados é descontado automaticamente na base de dados, é enviado um email de confirmação com o resumo da compra para o endereço indicado, e o carrinho é esvaziado, sendo o utilizador redirecionado para a página de produtos.

Se o carrinho estiver vazio, é apresentada uma mensagem a convidar o utilizador a explorar a coleção ou as promoções.

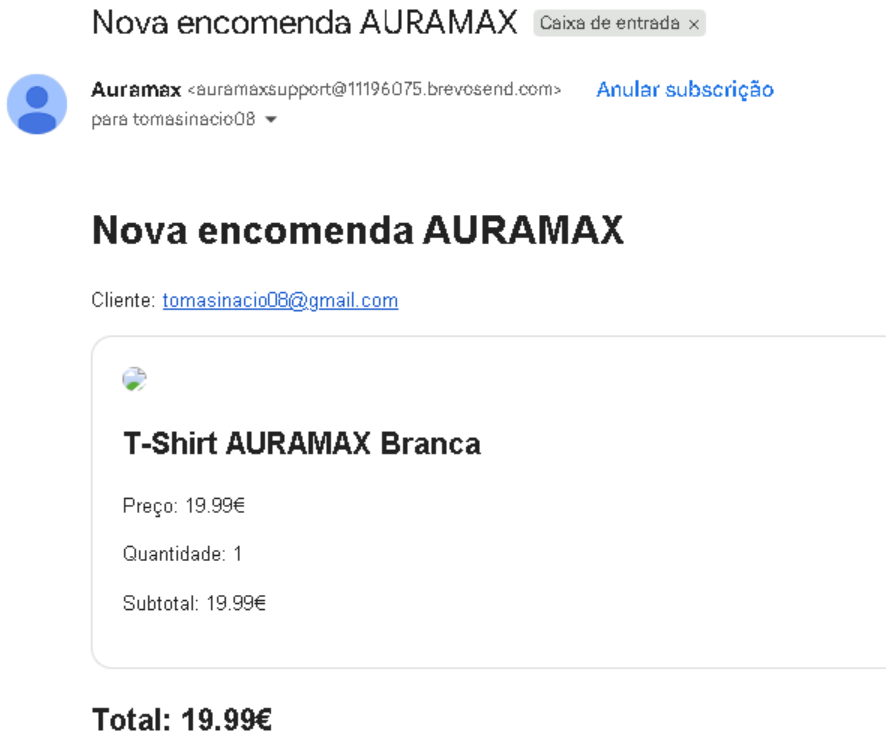


Figura 48 Recibo email

Área pessoal (Conta)

A página /conta está disponível apenas para utilizadores autenticados, sendo uma rota protegida. Caso um utilizador sem sessão iniciada tente aceder, é redirecionado automaticamente para a página de login.

Nesta área, o utilizador encontra a informação da conta (nome e email associados ao perfil), um resumo do carrinho com o número de artigos e o valor total atual, atalhos de acesso rápido ao carrinho e à coleção de produtos, e o botão "Terminar sessão" para efetuar logout.

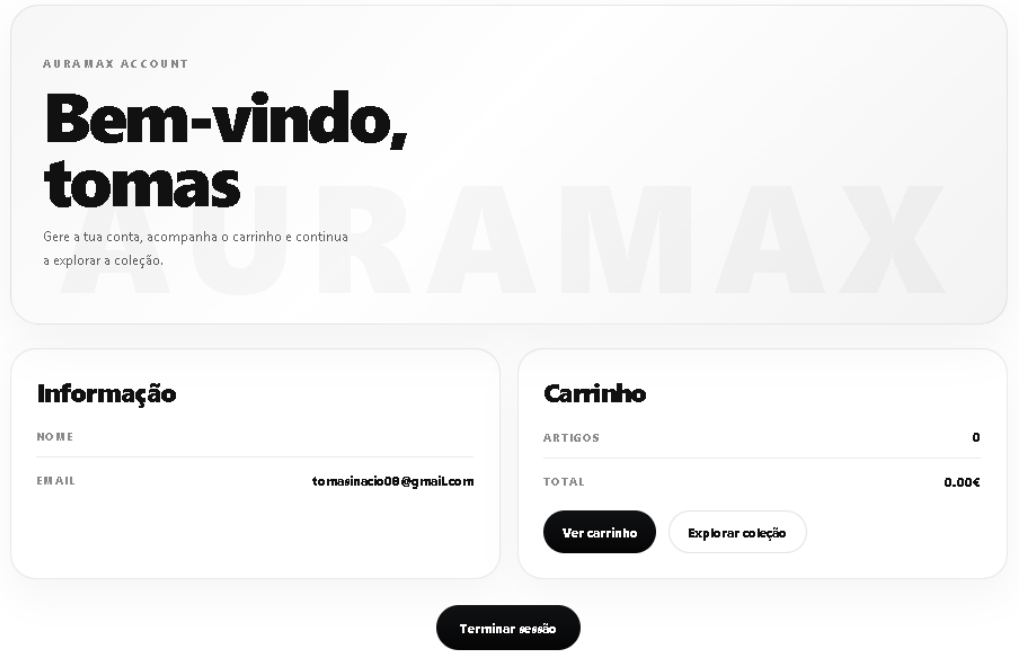


Figura 49 Área pessoal (conta)

Painel de administração

A página /admin está disponível exclusivamente para utilizadores com permissões de administrador. Qualquer outro utilizador que tente aceder vê uma mensagem de "Acesso Negado" com a opção de voltar à página inicial.

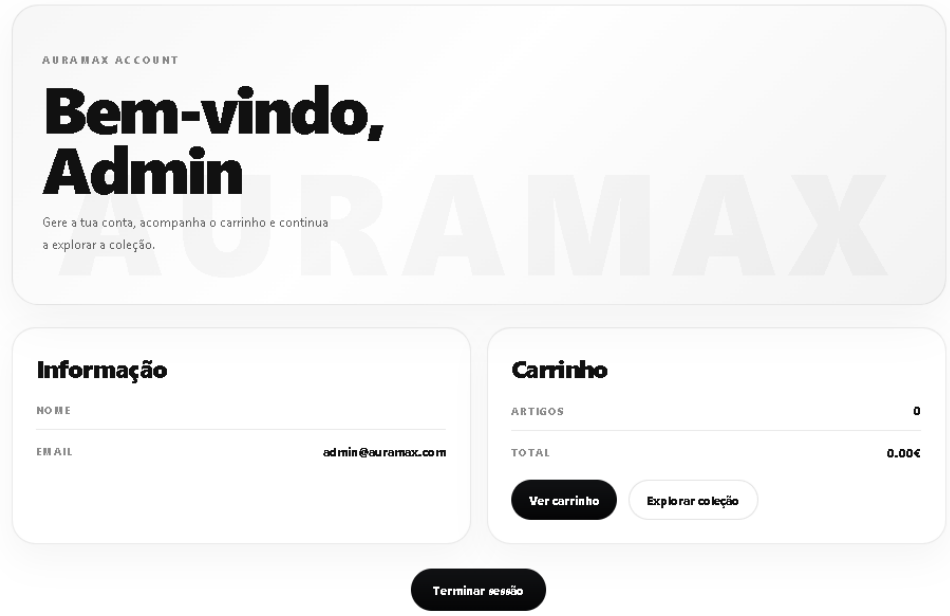


Figura 50 Conta Admin

Adicionar novo produto

O administrador pode criar um novo produto preenchendo o nome do produto, a categoria (selecionada de uma lista com T-Shirts, Hoodies e Calças), o stock inicial disponível e o preço em euros.

Adicionar imagens

É possível arrastar ou selecionar várias imagens, nos formatos PNG, JPG ou WEBP, para associar ao produto. Para cada imagem é possível visualizar uma miniatura e o nome do ficheiro, reordenar a posição subindo ou descendo na lista, e remover a imagem antes de submeter.

Após preencher os dados e adicionar as imagens, o administrador clica em "Adicionar Produto". O sistema confirma a criação do produto com o respetivo identificador e limpa o formulário para uma nova adição.

Figura 51 Painel de Admin

Páginas institucionais

Sobre nós

A página /sobre apresenta a missão, visão e valores da marca Auramax, bem como uma breve história sobre a sua criação.

Saldos

A página /saldos está reservada para a apresentação de promoções e descontos. Atualmente apresenta uma mensagem indicando que "Novas promoções em breve", com atalhos para a coleção completa.

Figura 52 Página promoções

Contacto e apoio ao cliente

Página de contacto

Na página /contacto, o utilizador pode preencher um formulário com nome, email, assunto e mensagem, para entrar em contacto com a equipa Auramax. A página apresenta ainda informações de contacto, como email, horário de funcionamento, localização e tempo médio de resposta, bem como acesso rápido às páginas de ajuda.

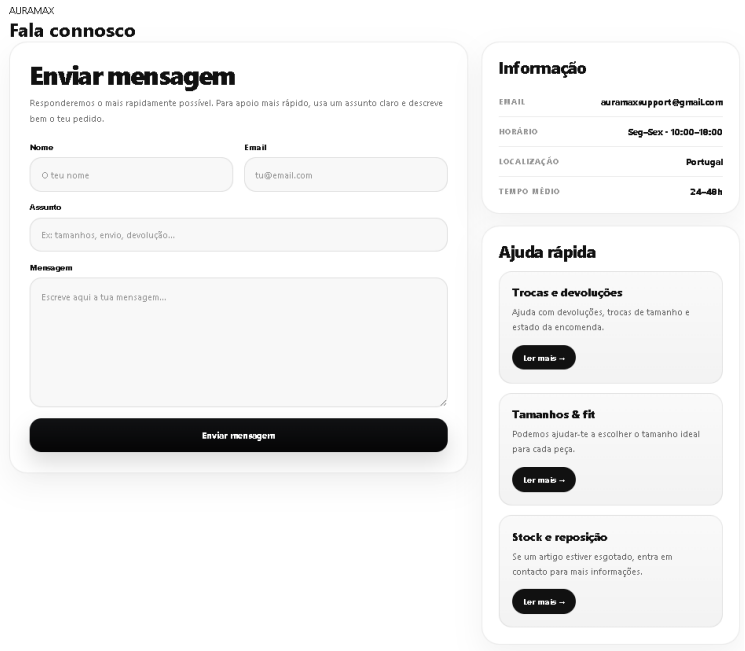


Figura 53 Página Fala conosco

Páginas de ajuda

A plataforma disponibiliza três páginas de apoio ao cliente, acessíveis a partir da página de contacto: trocas e devoluções (/ajuda/trocas), com informação sobre o prazo de 14 dias para devolução e condições do artigo; tamanhos e fit (/ajuda/tamanhos), com esclarecimentos sobre o corte das peças, distinguindo *oversized* de *true size*; e stock e reposição (/ajuda/stock), com informação sobre disponibilidade e reposição de artigos.

Em todas estas páginas é disponibilizado um botão de contacto direto por email para questões específicas.

Figura 54 Ajuda rápida

Notas:

- Conter linguagem que cumpra a sintaxe e semântica da língua portuguesa.
- Ter rigor técnico-científico.
- Conter os elementos apropriados ao enriquecimento do seu conteúdo (imagens, gráficos, fotografias, esquemas, etc.).
- Número máximo de páginas - 30 (sem contabilizar anexos).