Marketplace de itens usados

Natã Cezer bordignon, Gabriel Martins

Departamento de Sistemas e Computação
Universidade Regional Integrada (URI) – Erechim, RS, Brasil

103574@aluno.uricer.edu.br, 103799@aluno.uricer.edu.br

Abstract. *This paper presents the architecture and implementation of a fullstack system for a second-hand item marketplace. The backend, built with Node.js, Express, TypeORM, and PostgreSQL, offers endpoints for user and product management, JWT-based authentication, and integration with Docker services. The frontend, developed separately, consumes these APIs and provides an intuitive interface for user interaction. The project demonstrates a modular and scalable design, covering both server-side and client-side development.*

Resumo. *Este artigo apresenta a arquitetura e a implementação de um sistema fullstack para um marketplace de itens usados. O backend, desenvolvido com Node.js, Express, TypeORM e PostgreSQL, oferece endpoints para gerenciamento de usuários e produtos, autenticação baseada em JWT e integração com serviços via Docker. O frontend, desenvolvido separadamente, consome essas APIs e fornece uma interface intuitiva para interação dos usuários. O projeto demonstra um design modular e escalável, cobrindo tanto o desenvolvimento do lado do servidor quanto do lado do cliente.*

1. Introdução

A venda de produtos usados pela internet se tornou algo comum no dia a dia das pessoas, mas ainda assim muitas plataformas existentes acabam sendo complicadas de usar ou deixam a desejar em termos de organização e segurança. Pensando nisso, este projeto teve como objetivo criar um sistema completo (fullstack) para um marketplace de itens usados, que fosse simples, funcional e fácil de manter.

O sistema foi dividido entre backend e frontend. No backend, foram utilizadas tecnologias como Node.js, Express, TypeORM e PostgreSQL, além de autenticação com JWT e uso de contêineres Docker para facilitar o ambiente de desenvolvimento. Já no frontend, foi construída uma interface independente que se comunica com a API, permitindo o cadastro de usuários, gerenciamento de produtos e outras funcionalidades essenciais para o funcionamento do marketplace.

2. Trabalhos relacionados

Plataformas como OLX e Enjoei são exemplos populares de marketplaces de itens usados no Brasil. Elas permitem que usuários anunciem produtos, conversem com interessados e realizem negociações de forma independente. Apesar de serem amplamente utilizadas, essas plataformas possuem estruturas mais complexas, funcionalidades voltadas para grande escala e muitas vezes exigem integrações com sistemas de pagamento, avaliação e verificação.

Diferente dessas soluções consolidadas, este projeto foi desenvolvido com foco acadêmico, com o objetivo de aplicar na prática conceitos de desenvolvimento web fullstack. O sistema foi pensado para ser simples e funcional, com arquitetura modular, autenticação segura, integração com banco de dados relacional e um ambiente containerizado. Além disso, o controle total do código-fonte permite fácil expansão e personalização, o que torna a aplicação ideal para estudos, testes e evolução contínua em sala de aula ou projetos pessoais.

3. Tecnologias Utilizadas

O sistema foi desenvolvido com uma arquitetura fullstack, utilizando diversas tecnologias no backend, frontend e para suporte e testes. No backend, foram utilizadas as seguintes tecnologias: Node.js, ambiente de execução JavaScript para o servidor; Express, framework minimalista para criação de APIs e gerenciamento de rotas; TypeORM, biblioteca ORM para mapeamento objeto-relacional com PostgreSQL; PostgreSQL, sistema gerenciador de banco de dados relacional utilizado para armazenar as informações do sistema; JWT (JSON Web Token), mecanismo para autenticação e autorização de usuários; e Docker, ferramenta de contêinerização utilizada para facilitar o desenvolvimento, testes e deploy do ambiente.

O Node.js permite a execução de código JavaScript fora do navegador, tornando possível o desenvolvimento de aplicações web no lado do servidor. O Express complementa o Node.js ao oferecer recursos para criar rotas, lidar com requisições HTTP e estruturar o backend de forma simples e eficiente. O TypeORM facilita a comunicação com o banco de dados PostgreSQL, permitindo a criação e manipulação de tabelas e registros por meio de objetos e métodos JavaScript/TypeScript, sem a necessidade de escrever consultas SQL manuais. O PostgreSQL é o sistema de banco de dados relacional adotado, escolhido por sua robustez, segurança e suporte a operações complexas de consulta. Para garantir a segurança e o controle de acesso, foi implementada a autenticação baseada em JWT, que permite validar a identidade dos usuários por meio de tokens seguros.

No frontend, a aplicação foi construída com HTML e CSS para a estruturação e estilização das páginas web, enquanto o React foi utilizado para criar componentes interativos e reativos. O React permite que o frontend consuma a API exposta pelo backend e atualize a interface de forma dinâmica, sem a necessidade de recarregar a página, proporcionando uma melhor experiência para o usuário.

Além disso, foram utilizadas ferramentas de apoio para o desenvolvimento e testes: Docker, para isolar e executar os serviços do projeto em contêineres, garantindo a padronização do ambiente de execução; Adminer, ferramenta web para gerenciamento visual do banco de dados PostgreSQL; e Insomnia, uma ferramenta de testes para requisições HTTP, utilizada para validar os endpoints e a comunicação entre frontend e backend durante o desenvolvimento do sistema.

4. Arquitetura do Sistema: Backend

O backend expõe uma API RESTful, enquanto o frontend, desenvolvido em React, HTML e CSS, consome essa API para fornecer uma interface interativa aos usuários. A comunicação entre as camadas é feita através de requisições HTTP no formato JSON.

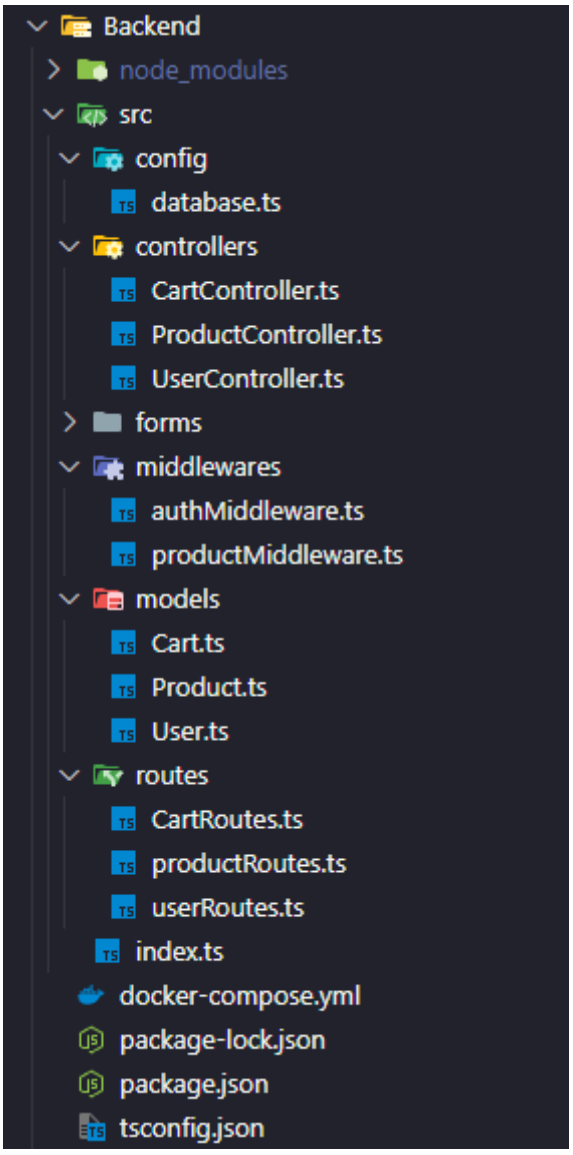

Imagem 1 mostra a estrutura das pastas backend

4.1 Usuário

O módulo de usuários permite o cadastro, login e autenticação. As principais rotas são:

- POST /api/users/register: realiza o cadastro de novos usuários no sistema.

- POST /api/users/login: autentica usuários existentes, retornando um token JWT para autorização de requisições futuras.

- GET /api/users/auth: valida o token JWT e autentica o usuário, garantindo o acesso às funcionalidades protegidas.

4.2 Produto

O módulo de produtos permite o cadastro, consulta, edição e remoção de itens no marketplace. As rotas implementadas são:

- POST /api/products/register: cadastra novos produtos no sistema, incluindo informações como nome, preço, categoria e descrição.

- PUT /api/products/:id: edita as informações de um produto existente, identificado pelo seu ID.

- DELETE /api/products/:id: remove um produto do sistema.

- GET /api/products/all: retorna a lista de todos os produtos cadastrados no marketplace.

- GET /api/products/:id: consulta as informações detalhadas de um produto específico.

4.3 Carrinho

O módulo de carrinho permite a adição, consulta, edição e remoção de itens no carrinho de compras do usuário. As rotas disponíveis são:

- POST /api/cart/add: adiciona produtos ao carrinho.
- GET /api/cart/all: retorna todos os itens adicionados ao carrinho pelo usuário.
- PUT /api/cart/:id: edita a quantidade ou os detalhes de um item específico no carrinho.
- GET /api/cart/:id: consulta as informações de um item específico no carrinho.
- DELETE /api/cart/:id: remove um item específico do carrinho.

O sistema segue o padrão MVC (Model-View-Controller), no qual os controllers são responsáveis pela lógica de negócio, os models representam as tabelas no banco de dados e as rotas direcionam as requisições para os controllers correspondentes. A camada de persistência foi implementada com o TypeORM, mapeando as entidades para tabelas no PostgreSQL, enquanto a autenticação é gerenciada via JWT, garantindo a segurança das operações.

5. Detalhamento do Frontend

5.1 Estrutura do Projeto em React

O frontend do sistema foi construído utilizando a biblioteca React, uma ferramenta baseada em componentes reutilizáveis que facilitam a construção de interfaces interativas e escaláveis. React opera por meio de um DOM virtual, que compara versões da interface para aplicar apenas as alterações necessárias no DOM real, garantindo alta performance. Toda a estrutura foi organizada em diretórios específicos, separando páginas (views), componentes reutilizáveis e arquivos de estilo. Além disso, foi utilizado o recurso de estado (via useState) e manipulação de eventos para capturar, validar e processar dados do usuário em tempo real. A arquitetura em React também facilita a manutenção do sistema e a evolução de novas funcionalidades, uma vez que cada componente funciona de forma isolada, mas integrada ao todo.

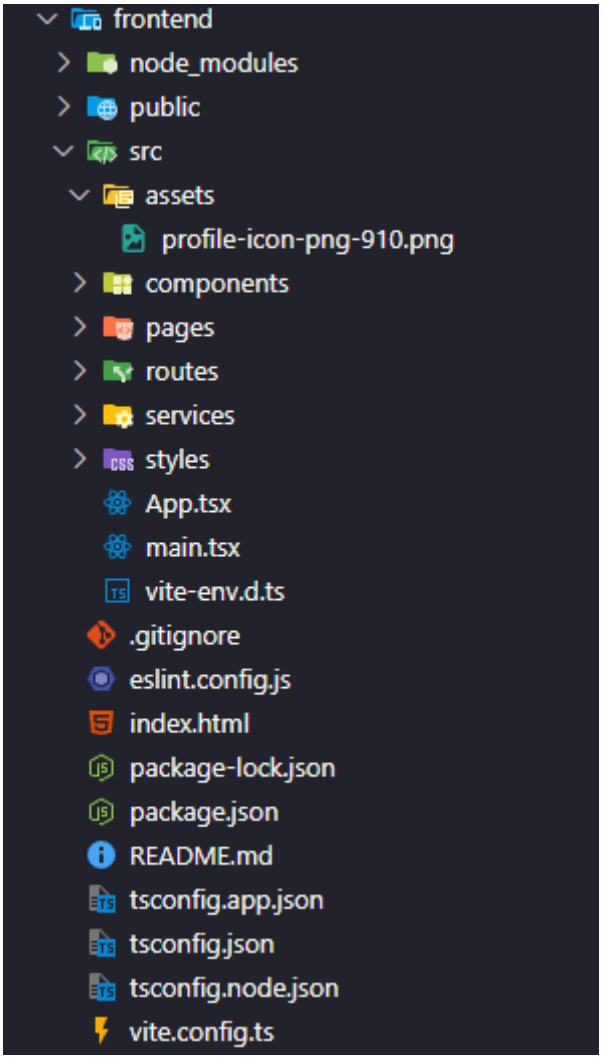

Imagem 2 mostra a estrutura das pastas do frontend

5.2 Tecnologias utilizadas

Para além do React, a aplicação faz uso de TypeScript como linguagem principal, substituindo o JavaScript padrão e oferecendo vantagens como tipagem estática, autocompletar inteligente, prevenção de erros em tempo de desenvolvimento e documentação automática de interfaces. O roteamento entre páginas foi realizado com a biblioteca React Router DOM, que possibilita criar rotas declarativas, incluindo rotas parametrizadas, privadas e de fallback, sem recarregar a página (SPA). A ferramenta de build escolhida foi o Vite, que se destaca por sua velocidade de inicialização, suporte nativo a ES Modules e atualização instantânea de componentes durante o desenvolvimento (hot reload). Para a estilização, optou-se por CSS puro, permitindo controle total sobre o layout e maior liberdade de design. As folhas de estilo foram organizadas por componente, promovendo separação de preocupações e clareza na manutenção.

5.3 Estrutura de componentes

A estrutura de componentes da aplicação foi pensada para garantir modularidade e reutilização de elementos visuais. Cada funcionalidade foi dividida em páginas separadas, com destaque para o componente de registro de novos usuários, implementado em Register.tsx, que contém um formulário com validações básicas; a tela de login, desenvolvida em Login.tsx, com campos de e-mail e senha; e a Home.tsx, que representa a página inicial do marketplace com uma mensagem de boas-vindas e barra de busca. A listagem de produtos disponíveis está presente em ProductList.tsx, enquanto o gerenciamento do carrinho foi tratado em Cart.tsx. A visualização das informações do usuário foi construída na página UserPage.tsx.

5.4 Estilização com CSS Modular

A estilização foi pensada para garantir clareza, legibilidade e fácil manutenção. Utilizou-se CSS modular, onde cada componente possui sua própria folha de estilos isolada. Essa abordagem evita conflitos entre classes, melhora a organização e permite alterar o estilo de um componente sem afetar os demais. Além disso, foram aplicados conceitos de responsividade utilizando flexbox, margens automáticas e centralizações verticais e horizontais, garantindo boa experiência em diferentes tamanhos de tela. Os botões, links e campos de input foram padronizados com base em uma paleta de cores definida e fontes importadas (Red Hat Display), criando uma identidade visual coesa em todo o sistema.

5.5 Integração com a API

A troca de dados entre frontend e backend foi feita através de requisições HTTP. Em alguns pontos foi utilizada a API nativa do JavaScript (fetch) e em outros, a biblioteca axios, devido à sua facilidade de configuração e melhor tratamento de erros. As operações principais realizadas incluem: envio de dados de cadastro, autenticação via login, obtenção da lista de produtos, inclusão de produtos no carrinho e exclusão de itens. O sistema foi preparado para lidar com respostas de erro, redirecionamentos automáticos e validações baseadas nas respostas da API. As requisições foram organizadas em chamadas assíncronas (async/await), permitindo controle de fluxo e melhor leitura do código.

5.6 Experiência do Usuário na Interface

Durante o desenvolvimento da interface, priorizou-se a experiência do usuário (UX). Foram implementadas interações visuais como hover sobre botões, ícones de perfil com menus que surgem ao passar o mouse, placeholders personalizados nos campos de entrada e botões centralizados para facilitar a navegação. Os formulários foram validados com mensagens de alerta e bloqueio de envio quando os dados estavam incorretos. A navegação entre páginas foi projetada de forma fluida, com carregamento rápido e sem recarregamento completo da página. Também foram aplicadas boas práticas de acessibilidade, como textos alternativos em imagens, contraste de cores legível e estrutura semântica correta com tags como <main>, <header> e <footer>, contribuindo para uma aplicação mais inclusiva.

7. Melhorias e Diferenciais Implementados

7.1 Menu flutuante no perfil do usuário

Uma das funcionalidades adicionadas foi a exibição de um menu de opções ao passar o mouse sobre o ícone do perfil do usuário. Esse menu dinâmico oferece atalhos para seções importantes, como o acesso ao perfil, ao carrinho e à página de anúncios, além da opção de logout. Essa interação foi construída utilizando apenas HTML, CSS e React, com manipulação de visibilidade via :hover, e garante uma navegação rápida e intuitiva sem a necessidade de ocupar espaço constante na tela.

7.2 Animações e interações visuais

O projeto apresenta diversas animações e interações visuais que contribuem para uma interface mais moderna e agradável. Botões com efeitos de hover, campos com foco visualizado, e feedbacks em ações como envio de formulários foram cuidadosamente implementados. Esses detalhes tornam a navegação mais fluida e fornecem ao usuário um retorno imediato sobre suas ações, favorecendo a usabilidade geral da aplicação.

7.3 Organização por categorias

A aplicação oferece uma visualização organizada dos produtos por categorias, facilitando a filtragem e a localização de itens de interesse. As categorias são exibidas no layout lateral em forma de botões simples, agrupados verticalmente, permitindo ao usuário selecionar o grupo de produtos desejado com apenas um clique. Essa funcionalidade é essencial em aplicações de e-commerce, por melhorar significativamente a experiência de navegação.

7.4 Campo de busca com botão

Foi implementado um campo de busca com um placeholder personalizado, posicionado na tela inicial do sistema. Esse campo orienta o usuário sobre o tipo de conteúdo que pode ser pesquisado, e o botão posicionado ao lado executa a busca de forma direta e intuitiva. Essa funcionalidade simula o comportamento de sistemas reais de pesquisa, como aqueles encontrados em grandes marketplaces.

7.5 Validação de formulários com useState

Os formulários da aplicação utilizam o hook useState do React para o controle de seus campos. Cada entrada do usuário é armazenada em tempo real, permitindo a validação dinâmica dos dados antes do envio. Um exemplo prático dessa funcionalidade está na verificação de igualdade entre a senha e a confirmação de senha no momento do registro, bloqueando o envio do formulário caso os campos não coincidam. Esse modelo de controle reativo melhora a segurança e confiabilidade das ações do usuário.

8. Possibilidades de expansão

Apesar das funcionalidades implementadas já permitirem o uso completo da aplicação como um marketplace funcional, o sistema apresenta múltiplas possibilidades de expansão para versões futuras. Um dos caminhos mais promissores seria a implementação de um sistema de mensagens privadas entre usuários, permitindo contato direto entre comprador e vendedor. Outro recurso importante seria a integração com serviços de pagamento, como o Stripe ou o PayPal, viabilizando transações reais dentro da própria plataforma. Além disso, a funcionalidade de upload de imagens reais para os produtos, com pré-visualização e redimensionamento automático, tornaria o sistema ainda mais próximo de marketplaces profissionais.

Outras melhorias relevantes incluem a utilização de autenticação com tokens JWT no frontend, o que permitiria o controle de acesso a rotas privadas, aumentando a segurança da navegação. O uso de bibliotecas de UI como Material UI ou Tailwind CSS também poderia padronizar e acelerar o desenvolvimento visual de novos componentes, além de tornar a interface mais responsiva e moderna. A criação de um painel administrativo, com estatísticas e gerenciamento de usuários, produtos e anúncios, também figura como uma possibilidade interessante para transformar o sistema em uma solução completa e adaptável a diferentes contextos.

9. Conclusão

O processo de desenvolvimento deste sistema representou um exercício completo de integração entre frontend e backend, destacando a importância de um planejamento inicial bem definido e de uma arquitetura modular. Desde as primeiras decisões sobre as tecnologias a serem utilizadas até a organização dos diretórios e componentes, cada etapa demandou atenção ao detalhe e à manutenção de boas práticas de engenharia de software. A aplicação prática dos conhecimentos adquiridos ao longo do curso, em especial na construção de rotas, formulários, componentes reutilizáveis e controle de estado, consolidou a base técnica para o desenvolvimento de sistemas mais complexos.

Além disso, o projeto exigiu um olhar atento para a usabilidade e estética da aplicação, promovendo uma abordagem centrada no usuário. A preocupação com a clareza visual, com a interação fluida e com o comportamento reativo das interfaces demonstrou o quanto uma boa experiência de usuário pode impactar diretamente na percepção de qualidade do sistema. Essa etapa foi também marcada por um amadurecimento na forma de lidar com erros, testes e simulações de uso real, preparando o sistema para cenários variados e contribuindo para uma entrega final mais robusta.

Durante o desenvolvimento do projeto, diversos aprendizados técnicos e conceituais foram absorvidos. A utilização de React com TypeScript reforçou o domínio sobre tipagem estática, componentização e gerenciamento de estado local, especialmente com o uso de hooks como useState. Foi possível compreender, na prática, como o React cria uma aplicação do tipo SPA (Single Page Application), otimizando o carregamento e a experiência do usuário ao evitar recarregamentos completos da página. Aprendizados importantes também surgiram com a manipulação de eventos, integração de formulários e a organização dos arquivos de estilo por componente.

Do lado do backend, houve uma evolução significativa na criação de rotas RESTful, integração com banco de dados e na forma de tratar requisições assíncronas entre cliente e servidor. A prática com fetch e axios, além da simulação de fluxos de autenticação e validação, agregou conhecimento sobre segurança, performance e clareza na comunicação entre camadas. Os testes realizados ao longo da construção da aplicação ajudaram a identificar gargalos e pontos de melhoria, estimulando a prática de refatoração constante e planejamento incremental de novas funcionalidades.

Referencias

VIEIRA, Rosângela da Silva; AMARAL, Hellen Fernanda. *Plataformas de marketplace e o impacto no comportamento do consumidor: estudo de caso da OLX*. Revista de Gestão e Negócios, v. 13, n. 1, p. 78-95, 2021. Disponível em: https://revistadegestaoenegocios.com.br. Acesso em: 02 jun. 2025.

KOTLER, Philip; KELLER, Kevin Lane. *Administração de marketing*. 15. ed. São Paulo: Pearson, 2016.

CHRISTENSEN, Clayton. *O Dilema da Inovação: quando as novas tecnologias levam as grandes empresas à falência*. São Paulo: MBooks, 2011.

POSTGRESQL. *Documentação oficial do PostgreSQL*. Disponível em: https://www.postgresql.org/docs/. Acesso em: 02 jun. 2025.

DOCKER. *Documentação oficial do Docker*. Disponível em: https://docs.docker.com/. Acesso em: 02 jun. 2025.

REACT. *Documentação oficial do React*. Disponível em: https://react.dev/. Acesso em: 02 jun. 2025.

TYPEORM. *Documentação oficial do TypeORM*. Disponível em: https://typeorm.io/. Acesso em: 02 jun. 2025.

EKPOBIMI, Harrison Oke; KANDEKERE, Regina Coelis; FASANMADE, Adebamigbe Alex. *Conceptual framework for enhancing front-end web performance: Strategies and best practices*. Global Journal of Advanced Research and Reviews, v. 2, n. 1, p. 99-107, set. 2024. DOI: 10.58175/gjarr.2024.2.1.0032. Acesso em: 02 jun. 2025

BALLAMUDI, Venkata Koteswara Rao; et al. *Front-End Development in React: An Overview*. Engineering International, v. 7, n. 2, p. 117-124, set. 2023. Disponível em: plataforma ResearchGate. Acesso em: 02 jun. 2025

PREVITE, Joe. *React with TypeScript: Best Practices*. SitePoint, set. 2020 (atualizado nov. 2024). Disponível em: sitepoint.com. Acesso em: 02 jun. 2025

VIKTORSLLO, Víctor. *Build a React component library with TypeScript and Vite*. [blog]. VíctorLillo.dev, mar. 2024 (atualizado fev. 2025). Disponível em: victorlillo.dev. Acesso em: 05 jun. 2025

CHOWDHURY, Tamal; KURUP, Manikandan. *How To Set Up a React Project with Vite for Fast Development*. DigitalOcean Community, atualizado 14 mai. 2025. Disponível em: digitalocean.com. Acesso em: 05 jun. 2025

TESMA; et al. *Research and analysis of the frontend development frameworks for e-business*. IJEAST, 2021. Disponível em: ijeast.com. Acesso em: 05 jun. 2025

PELTONEN, Severi; MEZZALIRA, Luca; TAIBI, Davide. *Motivations, Benefits, and Issues for Adopting Micro-Frontends: A Multivocal Literature Review*. arXiv, jul. 2020. Disponível em: arxiv.org. Acesso em: 05 jun. 2025