Vizzy.

Vizzy - Plataforma Comunitária

Projeto de Desenvolvimento de Software

Enrique Rodrigues Nº28602
José Alves Nº27967
Diogo Machado Nº26042
Diogo Abreu Nº27975
André Silva Nº27965

Instituto Politécnico do Cávado e do Ave

24 de fevereiro de 2025

1 Introdução

No mundo atual, a economia colaborativa tem ganhado cada vez mais destaque, incentivando práticas como a troca, venda e empréstimo de produtos entre indivíduos de uma mesma comunidade. Com base nessa tendência, este projeto propõe o desenvolvimento de uma plataforma que facilite essas transações de forma organizada e segura. Através da marcação de encontros dentro da comunidade, os utilizadores poderão partilhar recursos, reduzindo desperdícios e promovendo um consumo mais sustentável. Este relatório apresenta os principais elementos do projeto, incluindo diagramas UML e mockups, que ilustram a estrutura e funcionalidades da plataforma.

2 Problema e Motivação

2.1 O Problema

2.2 A Solução Proposta

Em muitas comunidades, o consumo desenfreado e a dificuldade de acesso a determinados produtos criam desafios tanto econômicos quanto ambientais.

Muitas vezes, itens de valor são descartados ou ficam inutilizados por falta de uso, enquanto outras pessoas poderiam se beneficiar deles. Paralelamente, a aquisição de novos produtos pode ser um obstáculo devido a limitações financeiras ou dificuldade de disponibilidade.

Nesse contexto, um sistema de troca, venda e empréstimo de produtos dentro de uma comunidade pode se tornar uma solução eficiente para otimizar o uso dos recursos, reduzir desperdícios e fortalecer os laços entre os membros da comunidade.

Esse modelo permite que os indivíduos tenham acesso a itens de que necessitam sem necessariamente precisar comprá-los novos, ao mesmo tempo em que promovem a circularidade dos bens.

2.3 Descrição do Problema e Motivação para a Solução

O problema central reside na falta de um sistema organizado e acessível para facilitar a troca, venda e empréstimo de produtos dentro de uma comunidade.

Atualmente, muitas pessoas não sabem onde ou como oferecer seus itens para troca ou venda, resultando em desperdício e em um ciclo de consumo ineficiente.

Além disso, a aquisição de produtos novos pode ser um desafio financeiro para algumas famílias, tornando a economia colaborativa uma alternativa atraente.

A solução proposta é a criação de um sistema onde os membros de uma comunidade possam marcar encontros para trocar, vender ou emprestar produtos de forma organizada e segura.

Além disso, o sistema permitiria que um membro da comunidade fizesse uma marcação antecipada de quando necessitará de determinado produto, garantindo que itens parados sejam utilizados mais vezes e de forma mais eficiente.

Esse sistema permitiria uma gestão eficiente dos recursos, promovendo a reutilização de itens e reduzindo a necessidade de compras desnecessárias. Além do impacto econômico positivo, a iniciativa também traria benefícios ambientais ao reduzir o descarte de produtos ainda em boas condições.

Dessa forma, ao estabelecer uma plataforma de fácil acesso para troca, venda e empréstimo, seria possível fomentar a cooperação entre os membros da comunidade, fortalecer o senso de pertencimento e promover um consumo mais consciente e sustentável.

3 Processos de Negócio

3.1 Processo de Empréstimo/Aluguer

3.1.1 Etapas

- **Criar um produto**: O utilizador adiciona um produto à plataforma, fornecendo detalhes como nome, descrição, estado e fotos.
- **Definir datas bloqueadas**: O utilizador indica as datas em que o produto não estará disponível para empréstimo/aluguer.
- **Confirmar datas livres**: O sistema verifica e confirma as datas disponíveis para empréstimo/aluguer com base nas datas bloqueadas.
- **Definir valor associado**: O utilizador define o valor do aluguer (ou 0€ em caso de empréstimo).
- **O produto não desaparece**: Após o empréstimo/aluguer, o produto permanece na plataforma para futuras transações.

3.1.2 Fluxo de Trabalho

O utilizador cria o produto e define as datas disponíveis. Outros utilizadores podem solicitar o empréstimo/aluguer nas datas livres. O proprietário do produto confirma ou rejeita a solicitação. Após o uso, o produto volta a estar disponível na plataforma.

3.1.3 Interações Essenciais

- Possibilidade de elogiar o vendedor após o empréstimo/aluguer para garantir a qualidade do serviço.

4 Processo de Venda

4.0.1 Etapas

- **Criar um produto**: O utilizador adiciona um produto à plataforma com detalhes como nome, descrição, estado e fotos.
- **Definir um valor associado**: O utilizador define o preço de venda do produto.

- **Permitir ou não receber contrapropostas**: O utilizador decide se aceita ou não contrapropostas de outros utilizadores.
- **Aceitar/Rejeitar contrapropostas**: Se permitido, o vendedor pode aceitar ou rejeitar contrapropostas recebidas.
- **Concretizada a venda, o produto desaparece**: Após a venda, o produto é marcado como vendido.

4.0.2 Fluxo de Trabalho

O utilizador cria o produto e define o preço. Outros utilizadores podem comprar diretamente ou fazer contrapropostas (se permitido). O vendedor aceita ou rejeita as propostas. Após a venda, o produto é removido da plataforma.

4.1 Processo de Troca

4.1.1 Etapas

- **Criar um produto**: O utilizador adiciona um produto à plataforma com detalhes como nome, descrição, estado e fotos.
- **Associar um produto para troca**: O utilizador indica qual produto deseja receber em troca.
- **Fazer contrapropostas**: Outros utilizadores podem fazer contrapropostas com produtos diferentes.
- **Aceitar/Rejeitar contrapropostas**: O utilizador pode aceitar ou rejeitar as contrapropostas recebidas.

4.1.2 Fluxo de Trabalho

O utilizador cria o produto e indica o produto desejado em troca.O utilizador aceita ou rejeita as propostas. Após a troca, ambos os produtos são removidos da plataforma.

4.2 Integração dos Processos

4.2.1 Interações entre Processos

- **Empréstimo/Aluguer e Venda**: Um produto pode estar disponível para empréstimo ou venda,após a venda ele é marcado como vendido.

- **Troca e Venda**: Um produto pode ser oferecido para troca ou venda, mas o utilizador deve escolher uma das opções.
- **Avaliações**: Todos os usuários vão ter a possibilidade de colocar um "elogio"no vendedor após-transação para garantir a qualidade e confiança na plataforma.

4.2.2 Fluxo Geral

- O utilizador escolhe o tipo de transação (empréstimo/aluguer, venda ou troca).
- O utilizador cria o produto e define os detalhes específicos para cada processo.
- Outros utilizadores interagem com o produto (solicitam empréstimo, compram ou propõem trocas).
- O proprietário do produto confirma ou rejeita as solicitações.
- Após a transação, o produto é marcado como vendido ou permanece disponível (empréstimo/aluguer).

5 Requisitos Funcionais

6 Requisitos Não Funcionais

Os requisitos não funcionais (RNFs) definem as características e restrições de um sistema que não estão diretamente relacionadas às funcionalidades oferecidas, mas sim à qualidade, desempenho, segurança e usabilidade da aplicação. Eles garantem que o sistema seja eficiente, confiável e utilizável dentro de determinados padrões.

Código	**Requisito Não Funcional**
RNF 01	A interface do utilizador deve ser intuitiva e acessível para todas as faixas etárias.
RNF 02	Todos os dados dos utilizadores devem ser armazenados de forma segura.
RNF 03	A aplicação deve cumprir com o Regulamento Geral de Proteção de Dados (RGPD) da União Europeia, garantindo a privacidade e o consentimento para o uso de dados pessoais.
RNF 04	O tempo de carregamento do feed da aplicação não deve ultrapassar os 3 segundos.
RNF 05	A pesquisa de produtos deve devolver os resultados em menos de 5 segundos.
RNF 06	O sistema deve permitir a implementação de novas funcionalidades com um impacto mínimo no desempenho.
RNF 07	A aplicação deve exigir uma palavra-passe com pelo menos 8 caracteres, incluindo letras e números.
RNF 08	O sistema deve utilizar armazenamento em cache para reduzir o número de pedidos ao servidor e melhorar a eficiência.
RNF 09	O sistema deve limitar o tamanho máximo de imagens carregadas para 5MB, evitando uso excessivo de espaço.

Tabela 1: Requisitos Não Funcionais

7 Arquitetura

A arquitetura da plataforma é composta por um frontend desenvolvido com **React e Next.js**, um backend em **C#**, e uma base de dados gerida com **Supabase (PostgreSQL)**. Esta abordagem garante um sistema eficiente, seguro e escalável.

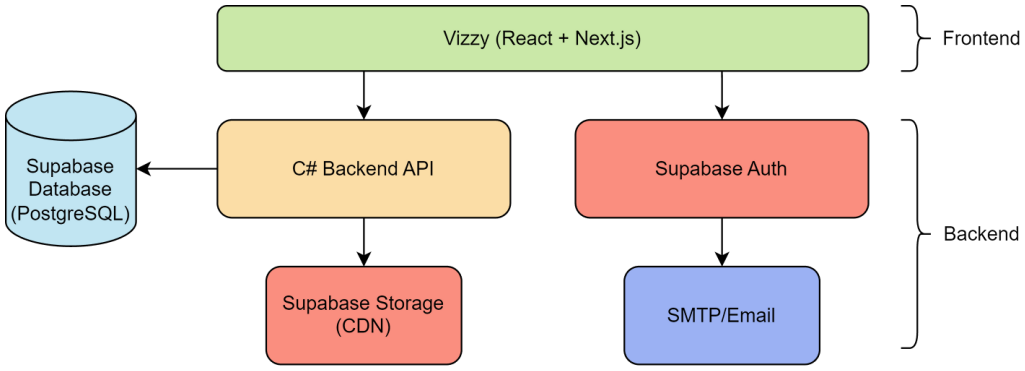

Figura 1: Arquitetura da Plataforma

7.1 Frontend

O frontend foi desenvolvido com **Next.js** para tirar proveito do server-side rendering (SSR) e static site generation (SSG), o que melhora o desempenho e o SEO. Além disso, o sistema utiliza API Routes para comunicação eficiente com o backend e otimização de carregamento de páginas para melhor experiência do utilizador.

7.2 Backend

O backend é composto por diversos serviços que trabalham em conjunto para garantir a funcionalidade da plataforma:

- **API Backend (C#):** Responsável pelo processamento dos pedidos HTTP do frontend, gestão de dados e comunicação com a base de dados.
- **Supabase Auth:** Sistema de autenticação e gestão de utilizadores, garantindo segurança no acesso à plataforma.

- **SMTP/Email:** Serviço de envio de emails para notificações e recuperação de credenciais.
- **Supabase Storage (CDN):** Sistema de armazenamento utilizado para gerir ficheiros e recursos estáticos.

7.3 Base de Dados

A plataforma utiliza o **PostgreSQL** como base de dados relacional. Esta base de dados oferece alta escalabilidade, confiabilidade e desempenho, permitindo operações eficientes de leitura e escrita.

7.4 Fluxo de Funcionamento

1. O utilizador interage com o **frontend (Vizzy)**.
2. Os pedidos são enviados para o **API Backend (C#)**, que os processa e consulta a base de dados.
3. A autenticação é gerida pelo **Supabase Auth**, garantindo a segurança no acesso.
4. Os ficheiros carregados são armazenados no **Supabase Storage (CDN)**.
5. O serviço de email (SMTP) é utilizado principalmente para a recuperação de senhas e outras funcionalidades de gestão de contas, como confirmação de registo e redefinição de credenciais.

Esta arquitetura assegura um sistema robusto, seguro e otimizado, permitindo uma gestão eficiente dos recursos e uma experiência fluida para os utilizadores.

8 Diagrama de Classes

8.1 Visão Geral

O diagrama de classes é uma representação visual da estrutura do sistema, demonstrando como suas classes se relacionam e interagem. É essencial para entender a organização do código, garantindo que o design da aplicação seja modular e bem estruturado.

Para isso, seguimos os princípios da *Clean Architecture*, que ajudam a manter um código organizado, de fácil manutenção e escalável. Esta abordagem permite que cada classe tenha uma responsabilidade bem definida, separando claramente as camadas da aplicação e reduzir o acoplamento entre os componentes.

8.2 Organização do Diagrama

O sistema está dividido em vários pacotes, cada um com uma função específica dentro da arquitetura. Esta estrutura facilita a manutenção, a testabilidade e a evolução do sistema. A seguir, apresentamos os principais pacotes e suas respectivas classes.

8.3 Gestão de Utilizadores

O pacote **User Management** trata da gestão dos utilizadores e das suas interações no sistema. Inclui as seguintes classes:

- **User**: Representa um utilizador do sistema, contendo atributos como nome, email e localização. Além disso, um utilizador pode estar associado a vários **anúncios** (*listings*) e **transações** (*transactions*), permitindo rastrear os itens que publicou e as operações que realizou.
- **Contacts**: Gere a lista de contactos de um utilizador e fornece operações para adicionar e remover contactos.
- **BlockedUsers**: Permite bloquear e desbloquear utilizadores, garantindo maior controlo sobre interações indesejadas.
- **Favorites**: Mantém uma lista de itens favoritos do utilizador, facilitando o acesso rápido a listagens de interesse.
- **Location**: Representa a localização geográfica de um utilizador ou de um anúncio.

- **Company**: Atributo opcional associado a um utilizador, que indica a empresa à qual ele pertence.

Além destas classes, o pacote inclui várias **enumerações** para representar os resultados das operações, como por exemplo:

- `BlockUserResult` – Indica o resultado de uma tentativa de bloquear um utilizador (sucesso, já bloqueado, não encontrado, erro).
- `AddContactResult` – Define os possíveis resultados ao adicionar um contacto (sucesso, já existe, contacto inválido, erro).

O uso de enumerações para representar estes resultados segue um princípio de **design limpo**, garantindo que as respostas das operações são bem definidas e evitam o uso de mensagens genéricas ou estados indefinidos. Com esta abordagem, o sistema mantém uma estrutura clara e consistente, facilitando a deteção e tratamento de erros através de códigos específicos.

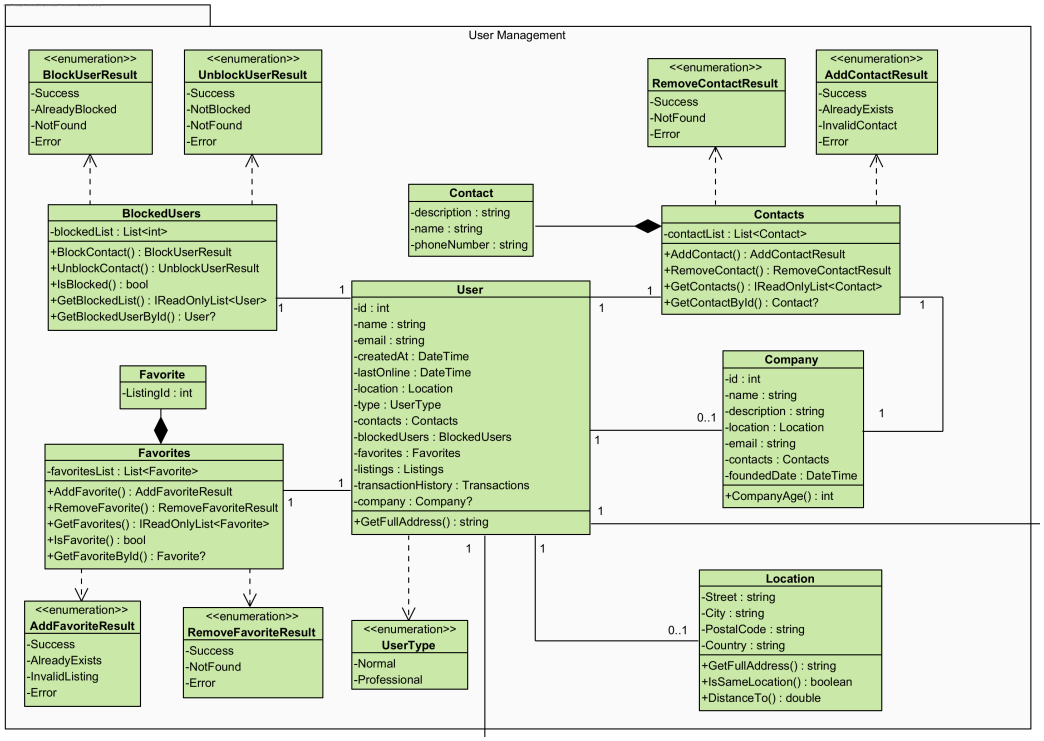

Figura 2: Pacote de Gestão de Utilizadores

8.4 Anúncios e Itens

O pacote **Listings & Items** é responsável pela gestão de anúncios e categorização de itens. Contém as seguintes classes:

- **Listings**: Classe que contém uma lista de anúncios, oferecendo funcionalidades para adicionar, remover e pesquisar itens dentro dessa lista.
- **BaseListing**: Classe abstrata que serve de base para diferentes tipos de anúncios.
- **SwapListing**: Extende a classe *BaseListing* e representa um item disponível para troca.
- **SaleListing**: Extende a classe *BaseListing* e representa um item disponível para venda.
- **GiveawayListing**: Extende a classe *BaseListing* e utilizado para itens oferecidos gratuitamente.
- **RentalListing**: Extende a classe *BaseListing* e representa itens disponíveis para aluguer.

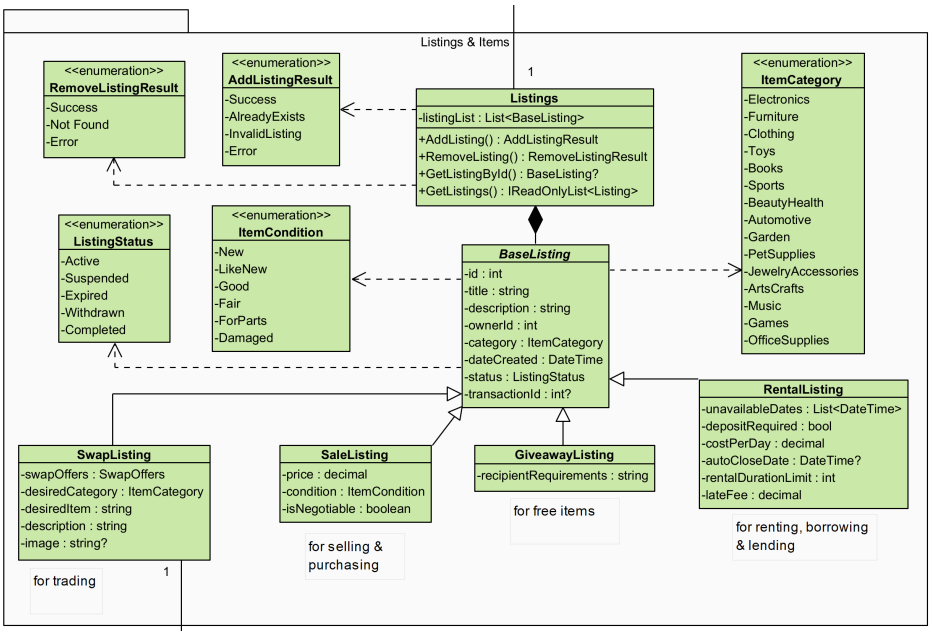

Figura 3: Pacote de Anúncios e Itens

8.5 Transações

O pacote **Transactions** gere as operações de compra, venda, troca e oferta de itens. Inclui:

- **Transactions**: Classe que contém uma lista de todas as transações registadas no sistema.
- **BaseTransaction**: Classe abstrata que define a estrutura comum a todas as transações, com atributos como utilizador, item, data e status.
- **RentalTransaction**: Extende a classe *BaseTransaction* e representa um aluguer de item entre utilizadores.
- **SaleTransaction**: Extende a classe *BaseTransaction* e representa uma transação de venda.
- **SwapTransaction**: Extende a classe *BaseTransaction* e permite a troca de itens entre utilizadores.
- **GiveawayTransaction**: Extende a classe *BaseTransaction* e gere a oferta de um item sem custo.

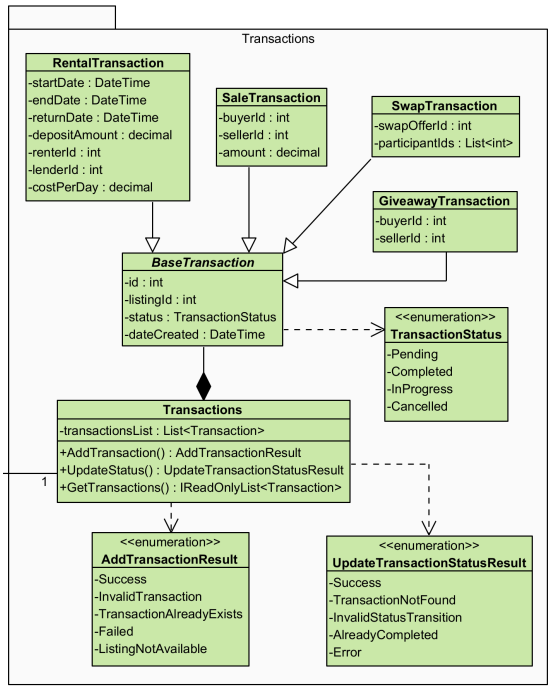

Figura 4: Pacote de Transações

8.6 Trocas

O pacote **Swaps** lida com a gestão de propostas de troca entre utilizadores. Inclui:

- **SwapOffers**: Contém uma lista de todas as ofertas de troca ativas no sistema.
- **SwapOffer**: Representa uma proposta individual de troca entre utilizadores.

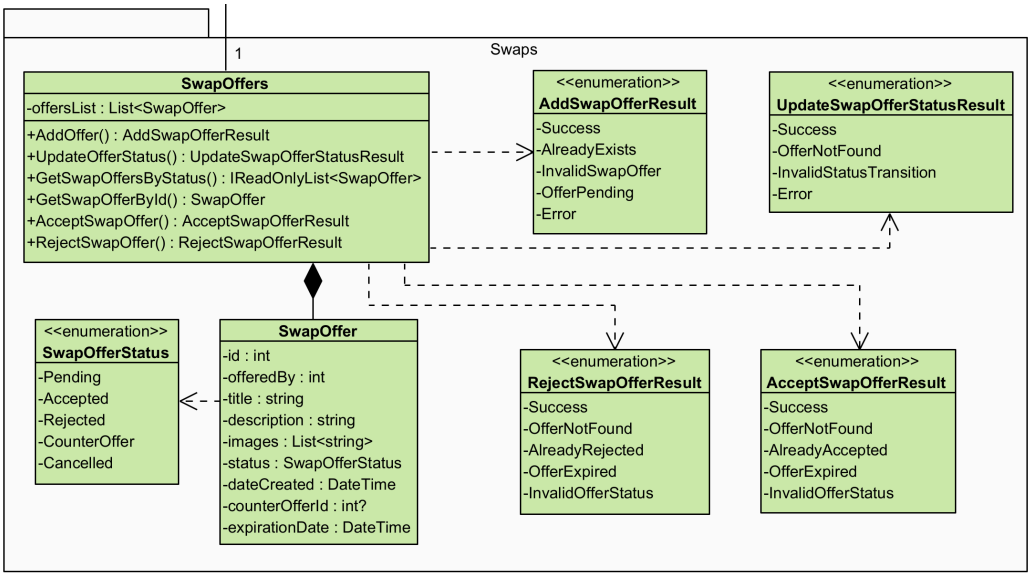

Figura 5: Pacote de Trocas

8.7 Relações entre os Pacotes

Os pacotes interagem de forma a garantir a funcionalidade completa do sistema. A classe `User` está associada às `Transactions`, registando as operações realizadas pelos utilizadores, como compras, vendas, alugueres, trocas e ofertas. As `Listings` estão relacionadas com os utilizadores, representando os itens que estes publicam no sistema, seja para venda, troca, aluguer ou oferta. O pacote `Swaps` é responsável por gerir as propostas de troca entre utilizadores, estando diretamente ligado às `SwapOffer`. Cada pacote tem uma função bem definida, e as ligações entre eles garantem a coesão e eficiência do sistema.

User Stories

A plataforma pr

Processos de Negócio

A plataforma pr

Conclusion

A plataforma proposta tem como objetivo criar um ambiente seguro e eficiente para a troca, venda e empréstimo de produtos dentro de uma comunidade. Ao permitir a marcação de encontros para a realização das transações, o projeto fomenta a interação entre os membros e incentiva práticas de consumo consciente. Com a definição clara dos requisitos e o suporte de diagramas UML e mockups, este relatório serve como um guia para o desenvolvimento da solução, garantindo que todos os aspectos essenciais sejam considerados.

Referências