Ana Carolina de Oliveira Xavier
Carolina Shypelenko Kowaluk
Érico Bis dos Santos
Deborah Citrin Fernandes
Pietra Ferreira Renner

Fundamentos de Desenvolvimento de Software
Trabalho Final

20 de novembro de 2023

1 Introdução

Este relatório apresenta as explorações realizadas ao longo do semestre de 2023/2 em torno dos fundamentos cruciais de desenvolvimento de software. Focando em conceitos como gerência de configuração e deploy, desenvolvimento em equipe, teste unitário, padrões de projeto e arquitetura de software (com ênfase na arquitetura CLEAN e aplicação dos princípios SOLID), o documento não apenas documenta aprendizagens, mas destaca aplicações práticas derivadas da integração desses elementos

2 Definição do Diagrama de Classes

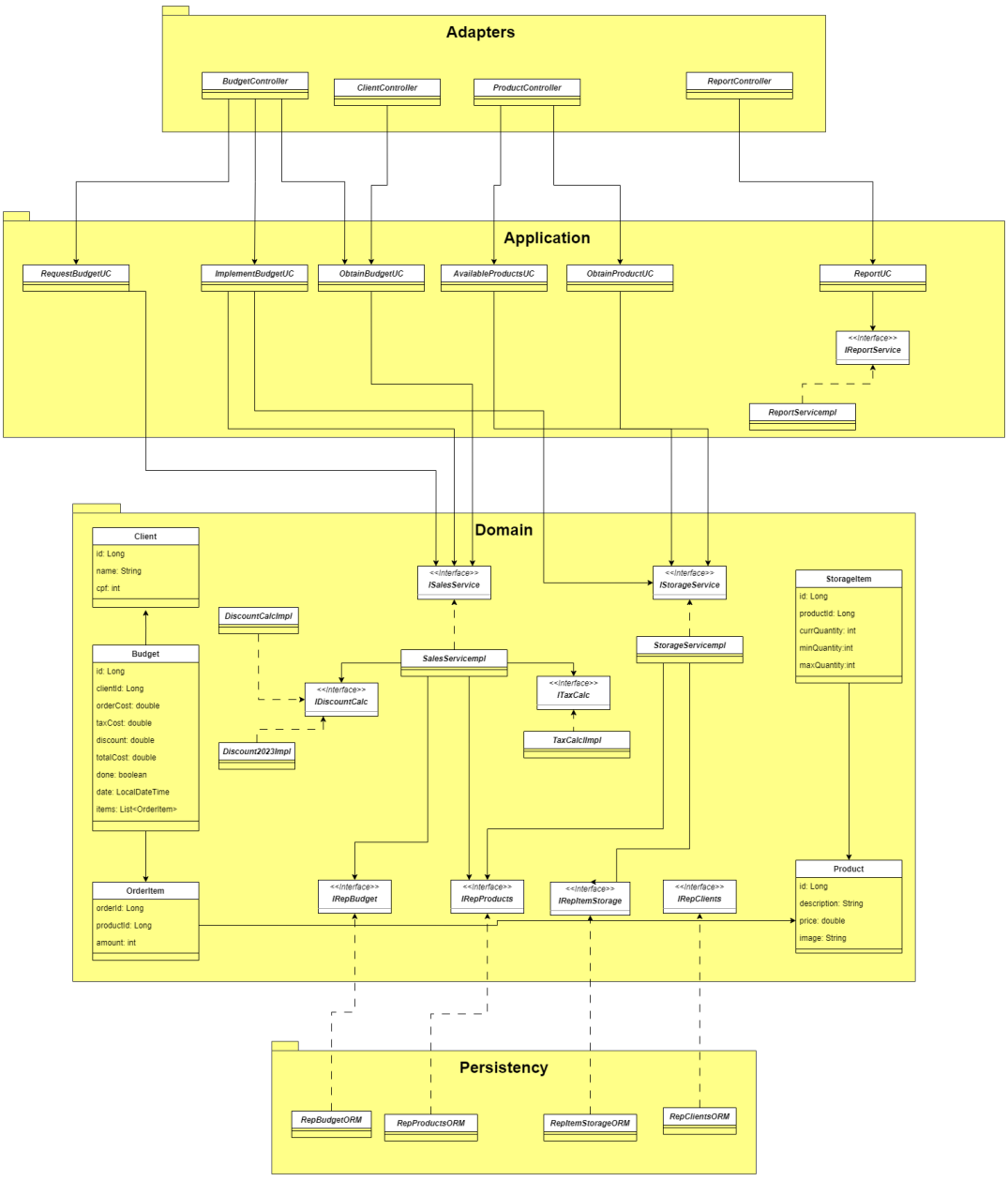

Figura 1: Diagrama de Classes do Projeto

3 Padrões de Arquitetura e Projeto Utilizados

3.1 Princípios da Arquitetura Limpa

O projeto foi concebido com o compromisso de respeitar integralmente os princípios da Arquitetura Limpa, promovendo a separação clara e precisa das responsabilidades do sistema.

3.1.1 Estrutura de Pastas

Foi criada uma estrutura de pastas cuidadosamente planejada para melhor separação e organização do código, assim como para manter os princípios da arquiterua limpas:

- **Domain:** Esta pasta foi reservada para conter as regras de negócio e entidades principais do sistema, representando o núcleo da aplicação de forma independente de frameworks e tecnologias externas.
- **Adapters:** Aqui, foram agrupados elementos que se adaptam às particularidades externas da aplicação, como controllers e mecanismos de persistência.
- **Application:** Nesta seção, foram alocadas as camadas responsáveis pela execução das regras de negócio, sem depender diretamente de detalhes de implementação externos. Isso garantiu a manutenção da lógica de negócios de forma isolada.

3.2 Princípios SOLID

Nosso projeto foi desenvolvido seguindo os princípios SOLID. Seguinos o Princípio de Responsabilidade Única ao definirmos classes que tem apenas uma única função dentro do sistema, o que ajuda a manter o código mais organizado. Também seguimos o Princípio da Segregação de Interfaces, já que cada módulo depende somente das operações que ele necessita, tendo interfaces específicas para a necessidade de cada classe. Outro princío SOLID que respeitamos em nosso projeto é o Princípio de Inversão de Dependências, já que somente referenciamos classes concretas estáveis. Se a classe concreta não for estável, usamos interfaces.

3.3 Inversão de Controle

O Princípio da Inversão de Controle focaliza a alteração da direção do fluxo de controle em uma aplicação. No contexto do Spring Boot, esse princípio é frequentemente aplicado por meio da técnica de injeção de dependência. Nesse método, as dependências de um componente são providas a ele ao invés de serem criadas internamente. Essa abordagem reduz significativamente o acoplamento entre as classes, resultando em um código mais flexível e facilitando as futuras manutenções.

A Inversão de Controle foi usada na implementação dos repositórios, que atuam como intermediários entre as interfaces dos repositórios e a implementação do Spring Data JPA. Por exemplo, ao empregar a injeção de dependência no construtor da classe *RepBudget*, a dependência da interface *IRepBudgetJPA* é injetada, permitindo à classe *RepBudget* utilizar os métodos definidos na interface sem conhecer os detalhes de implementação específicos. Essa separação viabiliza um desacoplamento entre a classe e a implementação concreta, tornando mais simples a troca de implementações sem afetar o código na classe *RepBudget*.

3.4 Strategy

Fazemos o uso do Padrão de Projeto Strategy quando aplicamos descontos e calculamos o imposto, o que é feito nas classes concretas *Discount2023Impl*, *DiscountCalc* e *TaxCalc*. No nosso caso, a classe SalesService é considerada uma classe contexto, que faz uso dos algoritmos das classes mencionadas anteriormente, através de uma interface. Essa interface possui uma abstração do algoritmo, que é implementado em classes que implementam a interface.

Cada classe concreta implementa uma versão do algoritmo, o que permite que a classe contexto não seja diretamente alterada quando é necessária uma mudança no algoritmo de desconto ou de imposto, que podem vir a ser alterados com alguma frequência. Para definir qual classe contém o algoritmo que deve ser utilizado naquele momento, é necessário incluir a anotação *Primary* na classe.

4 Testes Realizados

4.1 Classe SalesService

Foi desenvolvido um teste unitário para a funcionalidade de criar um orçamento na classe SaleService utilizando JUnit e Mockito. Os mocks, simuladores de comportamento de dependências externas, são

configurados no método setUp usando Mockito. Eles simulam o comportamento de classes externas como repositórios e serviços. O método de teste, executa a criação de um pedido com um item, usa o SaleSevice para criar um orçamento a partir desse pedido e então verifica se o valor total do orçamento corresponde ao valor esperado.

4.2 Classe Discount2023Impl

Os testes unitários desenvolvidos para a funcionalidade de calcular o desconto na classe Discount2023Impl também utilizam JUnit e Mockito. Para criar os Budgets necessários para efetuar os testes, foi criado um método que retorna uma lista de Budgets com custos definidos na chamada do método.

Como o serviço testado utiliza a média dos últimos Budgets do cliente como base para o cálculo do desconto, e todos os casos testam com um Budget de valor 1000, os casos de teste foram escolhidos como segue:

Média	Resultado Esperado
< 10000	0
10000 < m < 20000	100
20000 < m	150
m > 50000	300

Figura 2: Tabela de casos de teste da classe Discount2023Impl

Além desses, também foi testado o caso de haver mais de 10 Budgets registrados ao cliente nos últimos 6 meses, em que o resultado esperado é 250, considerando, também, o custo de 1000.

5 Análises da Aplicação

Foram conduzidas três análises essenciais na aplicação. A primeira análise focou no cliente com o maior volume de pedidos. A segunda análise explorou a média de vendas. Já a terceira análise concentrou-se nas compras acima de 2 mil reais. Para facilitar a visualização desses resultados, será disponibilizado um retorno HTTP no Swagger, proporcionando uma interface intuitiva e amigável para melhor compreenção dos insights obtidos a partir dessas análises.

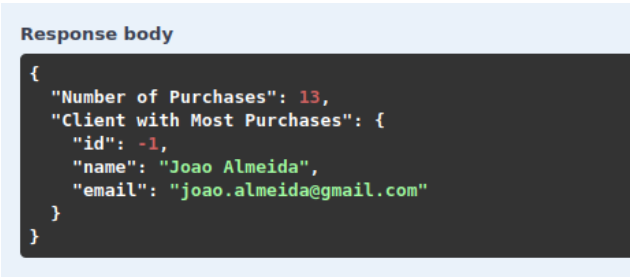

Figura 3: Cliente com mais volume de pedidos

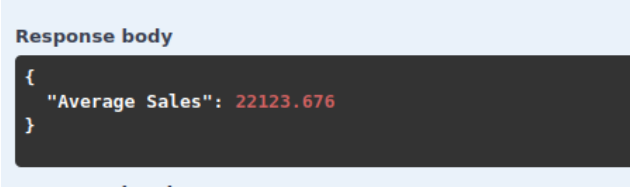

Figura 4: Média de vendas

Response body


```
{
  "Expensive Purchases (> 2000 reais)": [
    {
      "id": 28,
      "clientId": -4,
      "orderCost": 20000,
      "taxCost": 15,
      "discount": 0,
      "totalCost": 20015,
      "done": true,
      "date": "2023-10-02T15:30:00",
      "expirationDate": "2023-12-23T15:30:00",
      "items": []
    },
    {
      "id": 29,
      "clientId": -4,
      "orderCost": 20000,
      "taxCost": 15,
      "discount": 0,
      "totalCost": 20015,
      "done": true,
      "date": "2023-09-05T17:45:05",
      "expirationDate": "2023-12-23T15:30:00",
      "items": []
```


Figura 5: Compras acima de 2 mil Reais

6 Desenvolvimento da Interface

O desenvolvimento da parte Front-end do projeto foi realizado com o uso do Next.js 14.0.2, um framework baseado em React. Este ambiente foi explorado para criar uma interface web dinâmica e responsiva, fornecendo ao usuário as principais funcinalidades da api Back-end, como Catálogo de Produtos, Histórico de Pedidos, Gerenciamento de Carrinho e Orçamentos, além de páginas para confirmação e feedback de pedidos. A autenticação de usuários foi implementada através do Next Auth, simplificando o processo de login e registro. As interações com o back-end foram realizadas por meio de requisições HTTP usando Axios, direcionadas para localhost:8080.

6.1 Estrutura de Páginas e Funcionalidades

- **Catálogo de Produtos:** Esta página é dedicada à exibição dos produtos disponíveis para compra, oferecendo uma visualização intuitiva e organizada.

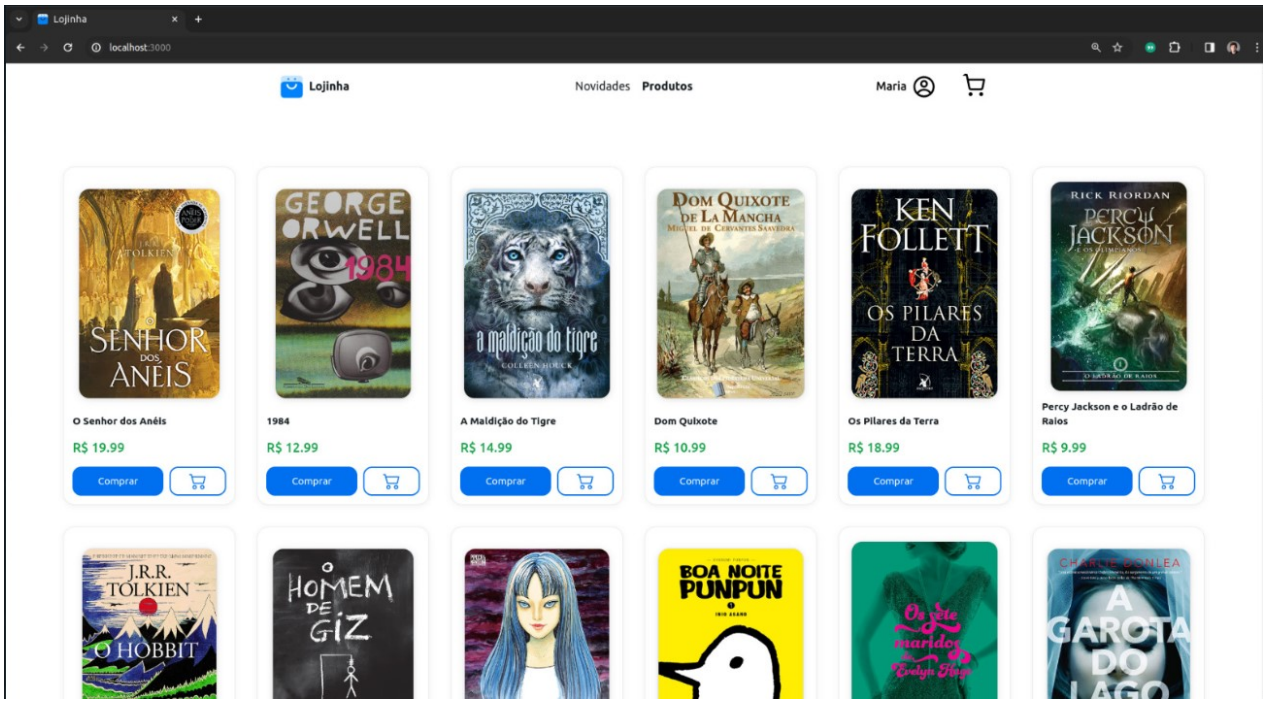

Figura 6: Página inicial da lojinha

- **Histórico de Pedidos do Cliente:** Aqui, é possível visualizar todos os pedidos efetuados pelo usuário, proporcionando um histórico detalhado das transações anteriores.

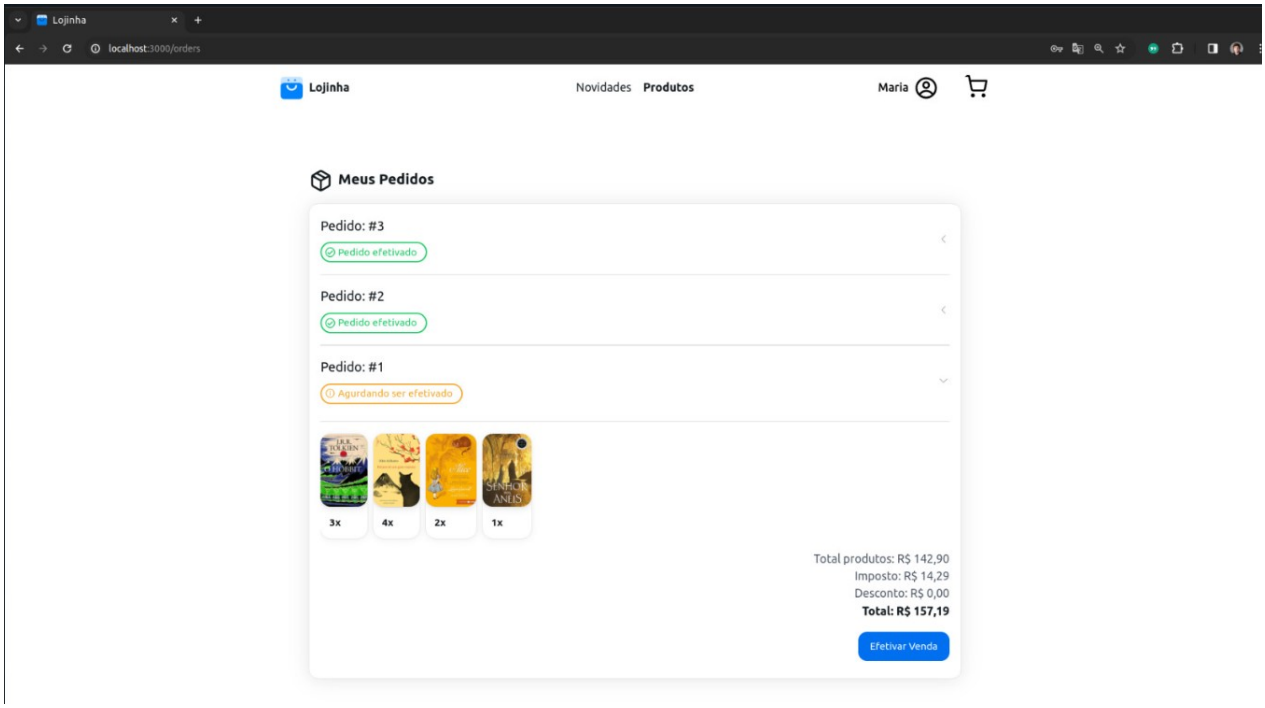

Figura 7: Histórico de pedidos

- **Carrinho de Compras e Orçamento:** Essa página permite ao usuário visualizar e gerenciar o conteúdo do carrinho de compras, bem como gerar um orçamento com informações detalhadas, incluindo impostos, descontos e o valor total do pedido.

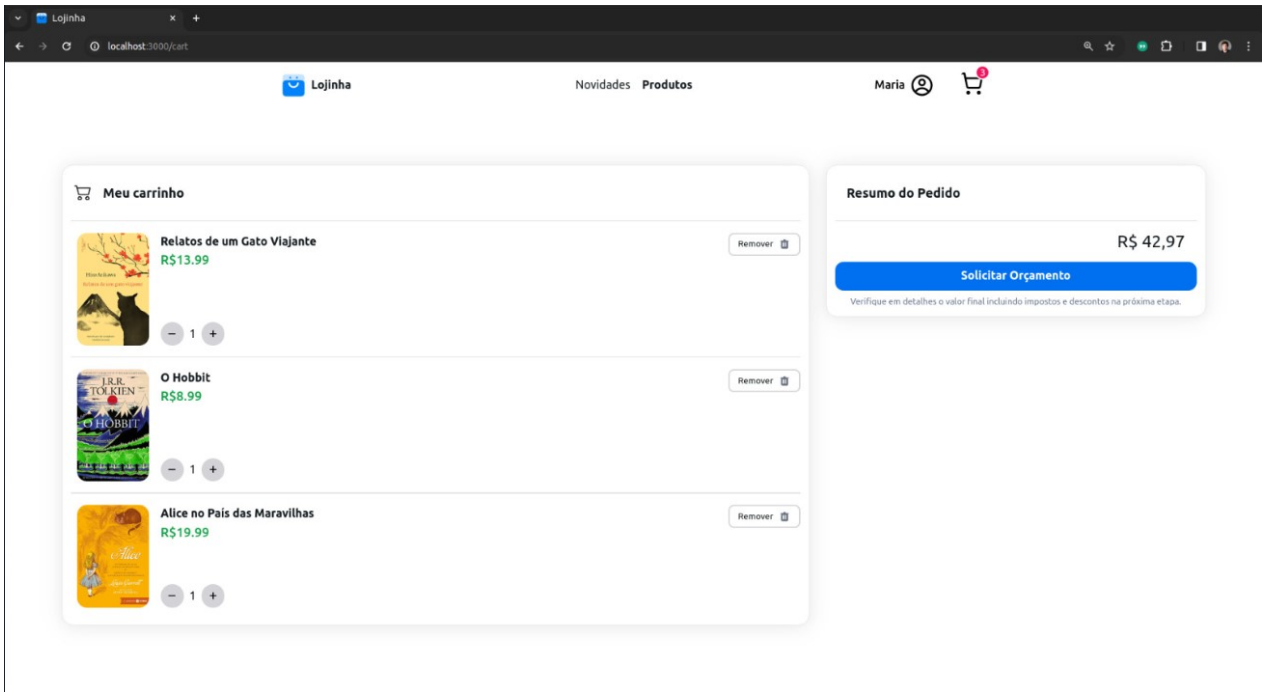

Figura 8: Página de orçamento/carrinho de compras

- **Confirmação de Pedido:** Após a geração do orçamento, esta página exibe os detalhes finais do pedido para confirmação, garantindo uma visão clara e transparente dos valores e condições antes da efetivação da compra.

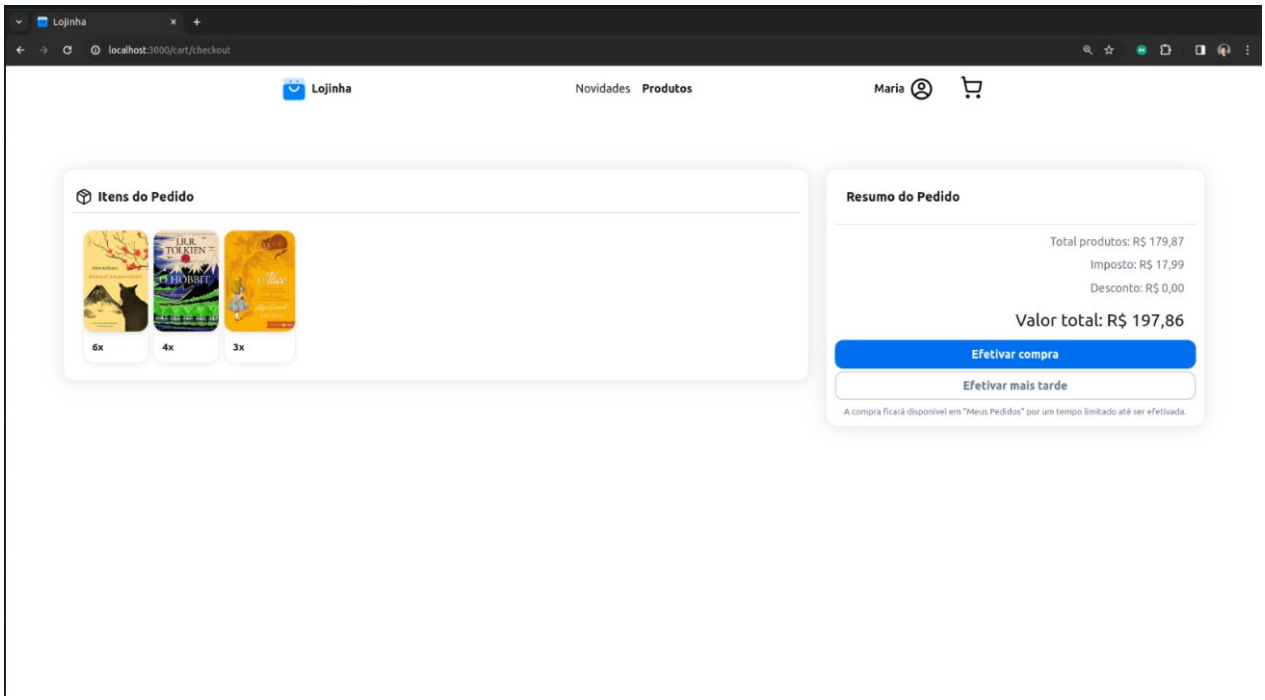

Figura 9: Página para efetivar compra

- **Página de Feedback:** Esta seção é crucial para informar ao usuário se o pedido foi efetivado com sucesso ou se ocorreu algum erro durante o processo, como produtos indisponíveis, expiração do orçamento, entre outros.

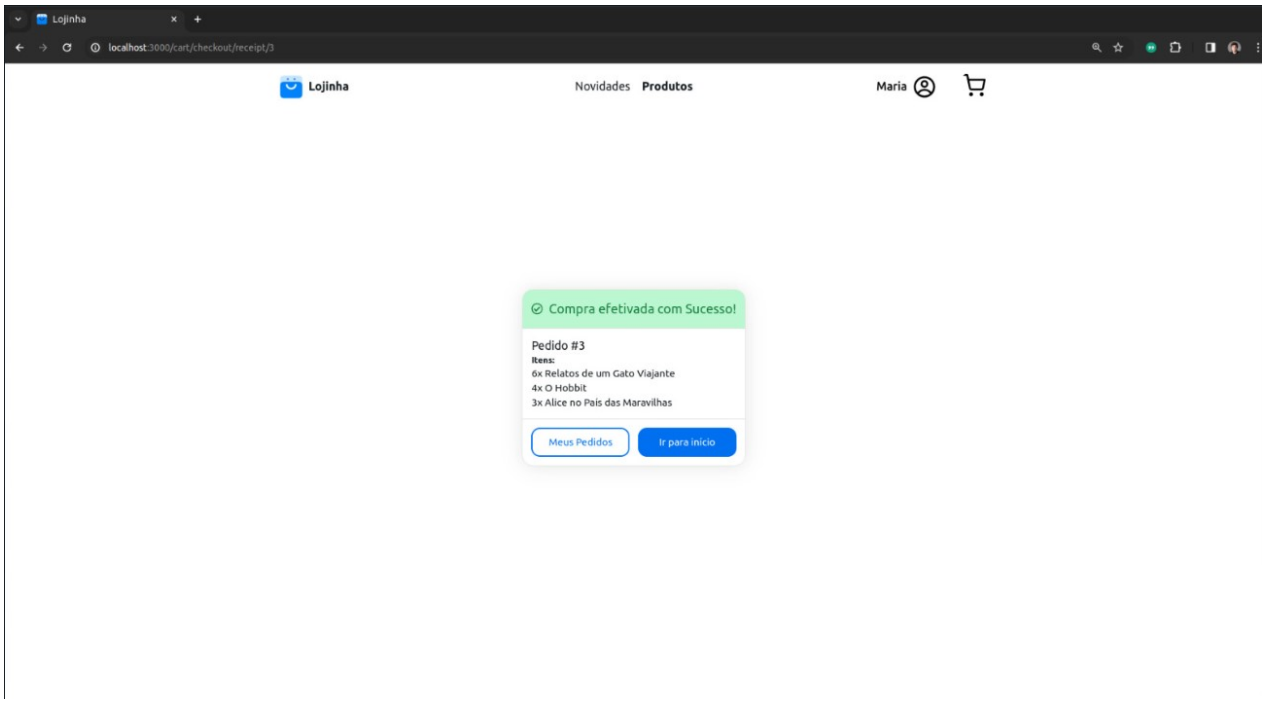

Figura 10: Compra efetuada

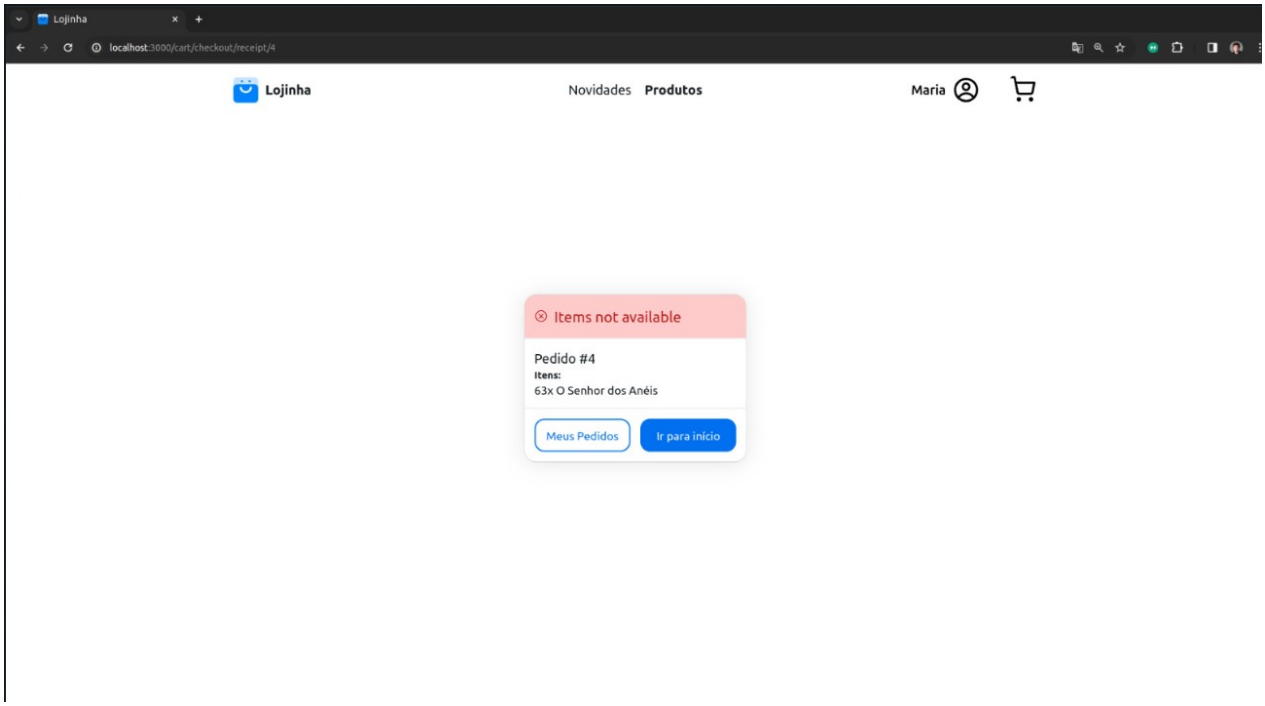

Figura 11: Exemplo de erro ao efetivar compra

7 Conclusão

A conclusão do trabalho final do semestre de 2023/2 é positiva, evidenciando o sucesso na exploração e aplicação dos conceitos abordados. A equipe demonstrou proficiência na gerência de configuração e deploy, desenvolvimento em equipe, teste unitário, padrões de projeto e arquitetura de software, incluindo a implementação da arquitetura CLEAN e dos princípios SOLID.
Para acessar o repositório com a aplicação: Repositório Sistema Vendas!