



Universidade Estadual da Paraíba (UEPB)  
Centro de Ciências e Tecnologia (CCT)  
Curso Bacharelado em Ciências da Computação



Pedro Lucas Vaz Andrade

Otávio Ferreira Clementino

Wagner Tibúrcio da Silva Junior

Relatório de Release 02 - Arquitetura

Campina Grande - 2025

<b>1. Visão Geral</b>	<b>3</b>
<b>2. Objetivos das Releases</b>	<b>3</b>
 Versão 1.0.0	3
 Versão 2.0.0	4
<b>3. Arquitetura da Versão 1.0.0</b>	<b>4</b>
3.1 Estrutura de Componentes	4
3.2 Decisões Arquiteturais	5
Padrão Facade	5
Persistência com GSON	6
3.3 Limitações Identificadas	6
<b>4. Evolução na Versão 2.0.0</b>	<b>6</b>
4.1 Novos Componentes	7
4.2 Implementação do Carrinho de Compras	7
4.3 Validações de Duplicidade	7
<b>5. Padrões e Boas Práticas Adotadas</b>	<b>8</b>
5.1 Aplicação dos Princípios SOLID	8
5.2 Tratamento de Erros com Exceções Personalizadas	8
<b>6. Conclusões e Próximos Passos</b>	<b>9</b>
6.1 Lições Aprendidas	9
6.2 Roadmap Técnico	9

# 1. Visão Geral

O sistema **Eng2Marketplace** foi concebido como uma plataforma B2B (Business-to-Business) voltada para o comércio eletrônico entre lojas e compradores institucionais. A versão 1.0.0 estabeleceu a base funcional com operações de cadastro e persistência. A versão 2.0.0 expandiu a arquitetura para contemplar relacionamentos entre entidades, validações e a introdução do conceito de carrinho de compras.

## 2. Objetivos das Releases



### Versão 1.0.0

- Implementar os módulos básicos do sistema:
  - Cadastro de Produtos (nome, valor, estoque).
  - Cadastro de Lojas (CNPJ/CPF, endereço).
  - Persistência de dados em arquivos JSON utilizando a biblioteca GSON.
  - Aplicação do padrão de projeto **Facade** para simplificação das interações com a View.

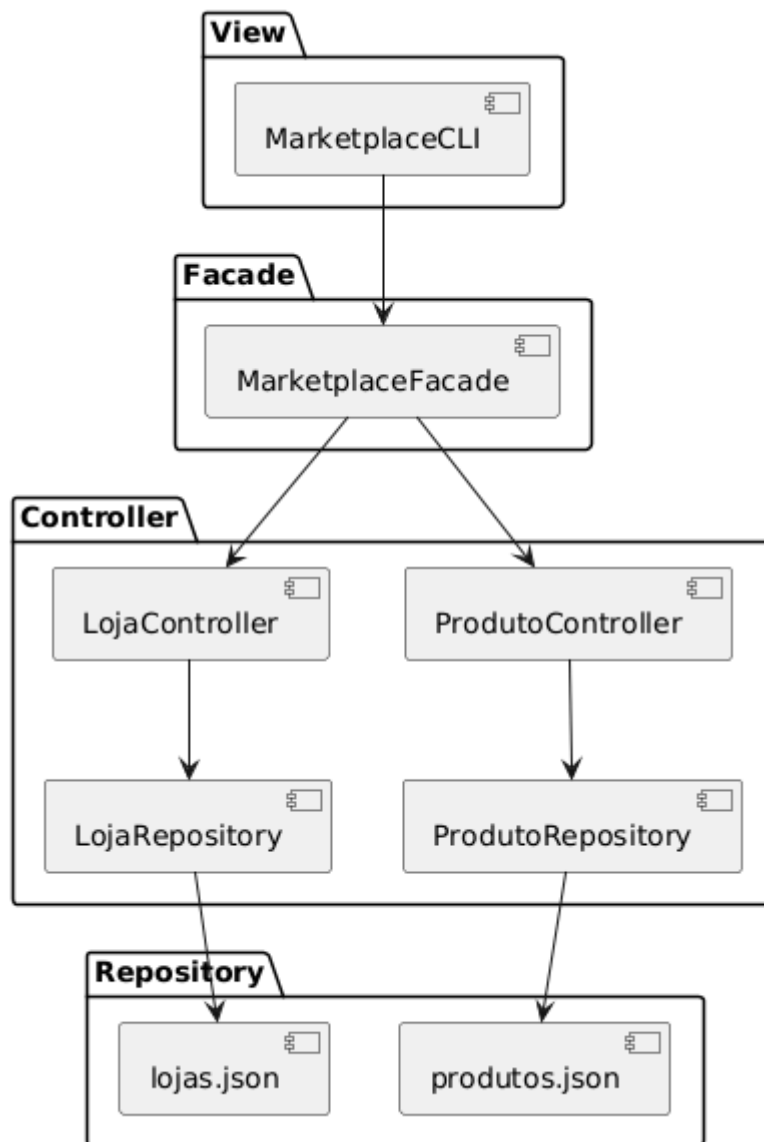


### Versão 2.0.0

- Evoluir a arquitetura com os seguintes recursos:
  - Cadastro e gerenciamento de Compradores (CPF, endereço).
  - Estruturação do Carrinho de Compras com controle de quantidades.
  - Implementação de validações de duplicidade e integridade.
  - Introdução da lógica de **venda** e consumo de estoque.
  - Melhorias na modularização e coesão entre camadas.

## 3. Arquitetura da Versão 1.0.0

### 3.1 Estrutura de Componentes



### 3.2 Decisões Arquiteturais

#### Padrão Facade

Aplicado para centralizar e abstrair as operações dos controllers. Simplifica a comunicação entre a camada de apresentação (View) e a lógica de negócio.

```
public class MarketplaceFacade {  
    private ProdutoController produtoController;  
    private LojaController lojaController;  
  
    public void adicionarProduto(String nome, double valor, Loja loja) {
```

```
    produtoController.adicionarProduto(nome, valor, loja);  
  }  
}
```

## Persistência com GSON

Dados são serializados em arquivos JSON. Cada entidade possui seu repositório específico.

### Exemplo de Estrutura:

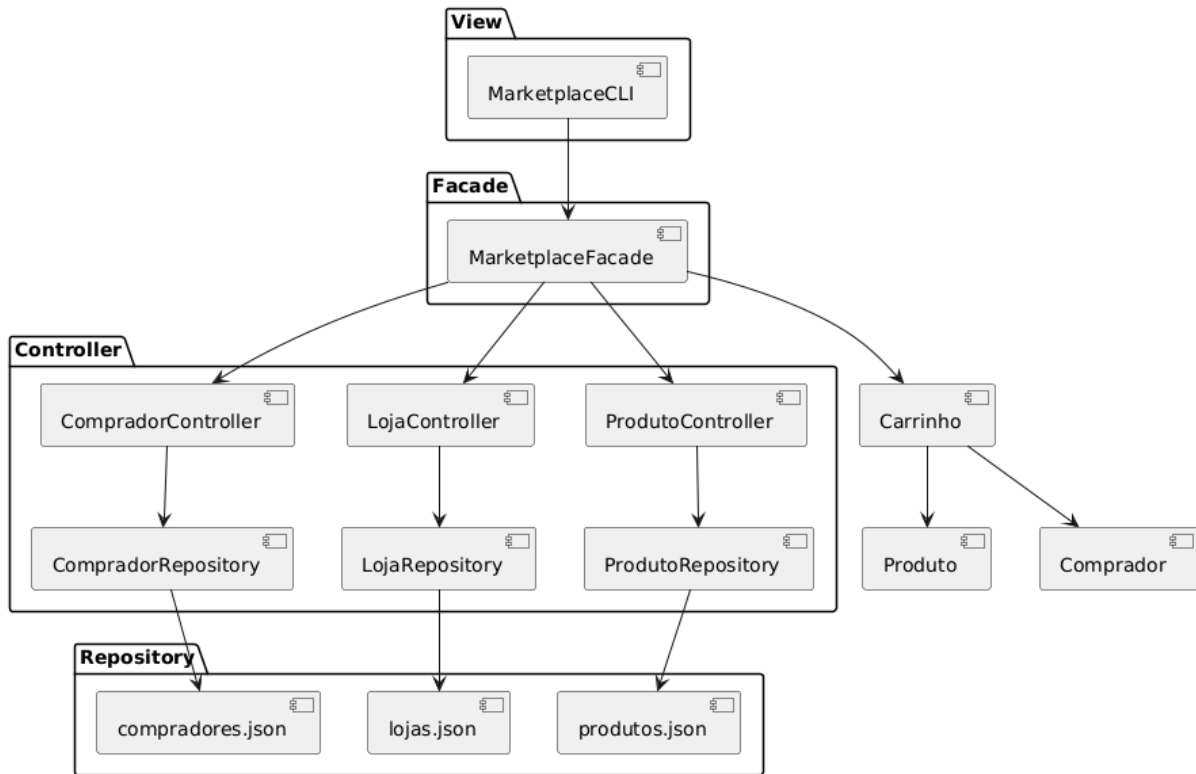
```
{  
  "produtos": [  
    {  
      "id": "abc123",  
      "nome": "Notebook",  
      "valor": 3500.00,  
      "loja": "loja123"  
    }  
  ]  
}
```

## 3.3 Limitações Identificadas

- Ausência de validação de dados duplicados (ex: CPFs, IDs).
- Relacionamentos frágeis com ligações manuais via ID.
- Dificuldade em representar relacionamentos complexos utilizando arquivos JSON planos.

## 4. Evolução na Versão 2.0.0

### 4.1 Novos Componentes



### 4.2 Implementação do Carrinho de Compras

Cada comprador possui um `Map<String, Integer>` que representa os produtos adicionados ao carrinho e suas quantidades.

```
public class Comprador {  
    private String id;  
  
    private Map<String, Integer> carrinho = new HashMap<>();  
}
```

1. Fluxo de Adição ao Carrinho:
2. A View solicita os produtos disponíveis.
3. A Facade valida o estoque.
4. O Controller atualiza o carrinho.

```
carrinho.put(produtold, carrinho.getOrDefault(produtold, 0) + quantidade);
```

### 4.3 Validações de Duplicidade

As validações foram organizadas em três níveis:

- **Facade:** verificação superficial para evitar chamadas desnecessárias.
- **Controller:** aplicação de regras de negócio (ex: CPF válido).
- **Repository:** garantia de unicidade e atomicidade dos dados.

```
public void salvar(Comprador comprador) {  
  
    if (listar().stream().anyMatch(c -> c.getCpf().equals(comprador.getCpf()))) {  
  
        throw new DuplicateCpfException(comprador.getCpf());  
  
    }  
  
}
```

## 5. Padrões e Boas Práticas Adotadas

### 5.1 Aplicação dos Princípios SOLID

Princípio	Aplicação
SRP (Responsabilidade Única)	Controllers especializados por entidade
OCP (Aberto/Fechado)	Sistema facilmente extensível para novas regras
DIP (Inversão de Dependência)	Views interagem apenas com a Facade

### 5.2 Tratamento de Erros com Exceções Personalizadas

```
public class DuplicateCpfException extends RuntimeException {
```

```
public DuplicateCpfException(String cpf) {  
    super("CPF " + cpf + " já cadastrado");  
}  
}
```

## 6. Conclusões e Próximos Passos

### 6.1 Lições Aprendidas

#### Aspectos Positivos:

- Adoção do padrão Facade simplificou a manutenção.
- Arquitetura modular favoreceu a expansão da versão 1.0 para a 2.0.
- Processo de migração assegurou continuidade dos dados.

#### Desafios Encontrados:

- Limitações do GSON em representar relações complexas.
- Falta de suporte nativo a transações e validações compostas.

### 6.2 Roadmap Técnico

Versão	Funcionalidades Planejadas
2.1.0	Implementação do fluxo de pedidos e checkout
3.0.0	Implementação do projeto a ser sorteado.