CENTRO DE EDUCAÇÃO TECNOLÓGICA DO AMAZONAS - CETAM
ESCOLA ESTADUAL DE TEMPO INTEGRAL GOVERNADOR MELO E PÓVOAS
CURSO TÉCNICO DE NÍVEL MÉDIO EM INFORMÁTICA

DESENVOLVIMENTO DE UM SISTEMA DE LOJA ONLINE UTILIZANDO PROGRAMAÇÃO ORIENTADA A OBJETOS

Modelagem, implementação e integração via API REST em C++

MANAUS - AM
2025

ANA JULIA DAMASCENO
BRUNO SIMÕES AMORIM
LEO FELIPE DOS SANTOS PINTO
HELIO EMANUEL PEREIRA
HENRIQUE GABRIEL FONSECA
MIGUEL ANGELO ROA RONDON

Relatório técnico de atividades práticas, apresentado como requisito parcial para aprovação na unidade curricular/curso: Linguagem de Programação - Técnico de Nível Médio em Informática.

MANAUS - AM
2025

SUMÁRIO

INTRODUÇÃO

Este relatório técnico apresenta o desenvolvimento de um Sistema de Loja Online completo, utilizando a linguagem C++ e seguindo rigorosamente os princípios da Programação Orientada a Objetos (POO). O projeto tem como objetivo simular funcionalidades reais de e-commerce, estruturando um sistema modular e escalável.

Além de demonstrar a implementação do sistema, o relatório descreve seus requisitos, modelagem de dados robusta baseada em UML e a integração externa via API REST (utilizando a biblioteca `cpp-httplib`), finalizando com uma análise sobre a aplicação dos pilares da POO no contexto do projeto.

OBJETIVO GERAL

Construir uma implementação sólida de um sistema funcional de vendas, respeitando os pilares da POO (Encapsulamento, Herança, Polimorfismo e Abstração).

OBJETIVOS ESPECÍFICOS

- Estruturar o sistema utilizando classes reutilizáveis e bem estruturadas (baixo acoplamento);

- Aplicar encapsulamento para proteção de dados sensíveis como estoque e preço;
- Modelar o domínio do problema através de diagramas UML claros;
- Implementar classes derivadas para diferentes tipos de produtos (Herança);
- Realizar integração via API REST para operações de cadastro e consulta.

DESENVOLVIMENTO

5.1 Levantamento de Requisitos

5.1.1 Requisitos Funcionais (RF):

- **RF01** - Cadastrar produtos (ID, Nome, Preço, Estoque).
- **RF02** - Gerenciar carrinho de compras (agregação de produtos).
- **RF03** - Gerenciar Loja (Controle de estoque e catálogo).
- **RF04** - Processar requisições de compra via API.
- **RF05** - Verificar disponibilidade de estoque em tempo real.

5.1.2 Requisitos Não Funcionais (RNF):

- **RNF01** - Código desenvolvido em C++17 para alta performance.
- **RNF02** - Uso da biblioteca `cpp-httplib` para servidor REST.
- **RNF03** - Modelagem visual utilizando PlantUML e JSON para dados.

5.1.3 Modelagem UML Estruturada

Diagrama de classes contendo:

- **Classe Produto** (Entidade base: ID, Nome, Preço).
- **Classe ProdutoDigital** (Derivada: Expansão via herança).
- **Classe Carrinho** (Agregação de itens selecionados).
- **Classe Loja** (Manager/Controlador central).

(Caso deseje, posso gerar o diagrama em imagem baseada na descrição).

5.1.3 Implementação em C++

Classe Base - Produto

A classe utiliza encapsulamento para proteger os dados, permitindo acesso apenas via métodos públicos.

C++

```
class Produto {
private:
    int id;
    std::string nome;
    double preco;
    int estoque;
public:
    Produto(int _id, string _nome, double _preco, int _estoque)
        : id(_id), nome(_nome), preco(_preco), estoque(_estoque) {}

    // Método virtual para Polimorfismo
    virtual void exibirInfo() {
         cout << "Produto: " << nome << " | Preço: " << preco << endl;
    }

    double getPreco() const { return preco; }
    int getEstoque() const { return estoque; }

    void atualizarEstoque(int qtd) {
        estoque -= qtd;
    }
};
```

[Código adaptado das fontes: 55, 56, 57, 58, 59, 60, 62, 63]

Classe Derivada - ProdutoDigital (Exemplo de Herança)

C++

```
class ProdutoDigital : public Produto {
private:
    string linkDownload;
public:
    ProdutoDigital(int id, string nome, double preco, int est, string link)
```

```
        : Produto(id, nome, preco, est), linkDownload(link) {}

    void exibirInfo() override {
        cout << "[Digital] ";
        Produto::exibirInfo();
        cout << "Link: " << linkDownload << endl;
    }
};
```

[Baseado no conceito de herança descrito na fonte: 119]

Classe Carrinho (Agregação)

C++

```
class Carrinho {
private:
    struct Item { Produto* p; int qtd; };
    vector<Item> itens;
    double total = 0.0;
public:
    void adicionarItem(Produto& p, int qtd) {
        if (p.getEstoque() >= qtd) {
            itens.push_back({&p, qtd});
            total += p.getPreco() * qtd;
            p.atualizarEstoque(qtd);
            cout << "Item adicionado!" << endl;
        } else {
            cout << "Estoque insuficiente." << endl;
        }
    }

    void listarItens() {
        for (auto& i : itens)
            i.p->exibirInfo();
    }
};
```

[Código adaptado das fontes: 66, 68, 69, 70, 71, 74]

5.1.4 Polimorfismo

Demonstração de flexibilidade onde métodos se comportam diferentemente em subclasses.

```
C++
void mostrarDetalhes(Produto* p) {
    p->exibirInfo(); // Chama o método específico (Produto ou ProdutoDigital)
}
```

5.1.5 Teste de Integração (Simulação com API)

```
C++
int main() {
    Loja loja;
    Produto p1(1, "Teclado", 150.00, 10);
    ProdutoDigital p2(2, "Ebook C++", 50.00, 100, "down.load/pdf");

    Carrinho carrinho;
    carrinho.adicionarItem(p1, 1); // Valida estoque e adiciona

    // Simulação API REST (httplib)
    httplib::Server svr;
    svr.Get("/produtos", [&](const Request& req, Response& res) {
        res.set_content(loja.toJSON().dump(), "application/json");
    });

    return 0;
}
```

[Baseado nas fontes: 84, 85, 86, 88, 122, 123]

---

6. REFATORAÇÃO

- Melhoria da estrutura para suportar integração externa via API REST.

- Separação modular para garantir baixo acoplamento entre Loja e Carrinho.
- Uso de `std::vector` para gerenciamento dinâmico de memória.
- Validação de dados (preço não negativo) antes da atribuição.

7. CONCLUSÃO

O desenvolvimento do Sistema de Loja Online em C++ permitiu a aplicação prática dos conceitos de POO, como encapsulamento e polimorfismo, essenciais para a segurança e extensibilidade do código.

A integração com a biblioteca `cpp-httplib` demonstrou a capacidade do C++ moderno de atuar em arquiteturas web, enquanto a modelagem UML garantiu uma visão clara da arquitetura do software antes da implementação. O projeto resultou em um sistema funcional que simula corretamente um fluxo de e-commerce real.

8. REFERÊNCIAS BIBLIOGRÁFICAS

- STROUSTRUP, Bjarne. **The C++ Programming Language**. 4th Edition. Addison-Wesley, 2013.
- CPP-HTTPLIB. A C++11 single-file header-only cross platform HTTP/HTTPS library. Disponível em GitHub.
- JSON FOR MODERN C++. Niels Lohmann. Documentação oficial da biblioteca.
- GAMMA, Erich et al. **Design Patterns**: Elements of Reusable Object-Oriented Software.