

# Relatório Técnico - Sistema Kanban Lite v3.0

---

## 1. Visão Geral do Projeto

### 1.1. Introdução

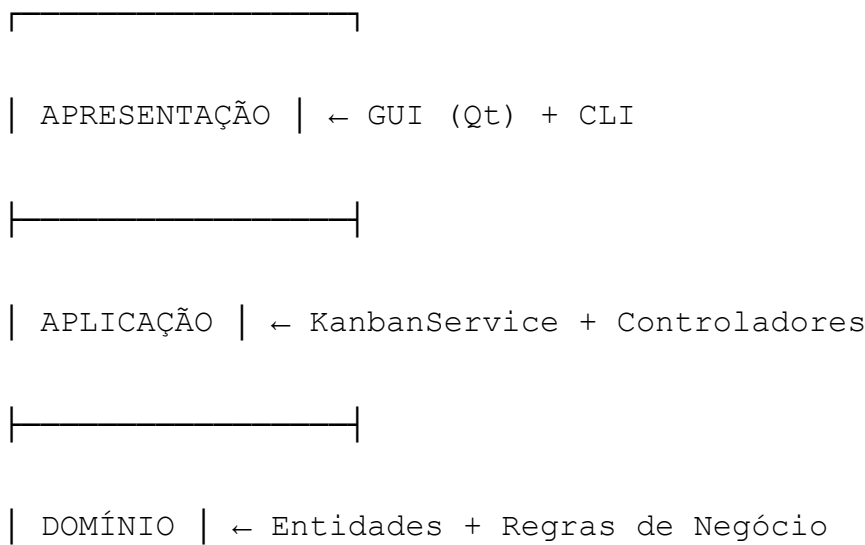
O **Sistema Kanban Lite** é uma aplicação completa desenvolvida em C++ moderno que implementa uma ferramenta de gerenciamento de tarefas baseada na metodologia Kanban. O projeto demonstra a aplicação prática dos principais conceitos de Programação Orientada a Objetos (POO) em C++, utilizando boas práticas modernas da linguagem e integrando uma interface gráfica robusta desenvolvida com Qt6.

### 1.2. Objetivos Principais

- Demonstrar domínio dos conceitos de POO em C++ moderno
- Implementar uma aplicação completa com arquitetura em camadas
- Fornecer interfaces múltiplas (CLI e GUI) para o mesmo domínio
- Aplicar padrões de design e boas práticas de engenharia de software

### 1.3. Arquitetura do Sistema

O sistema segue uma arquitetura em camadas bem definida:

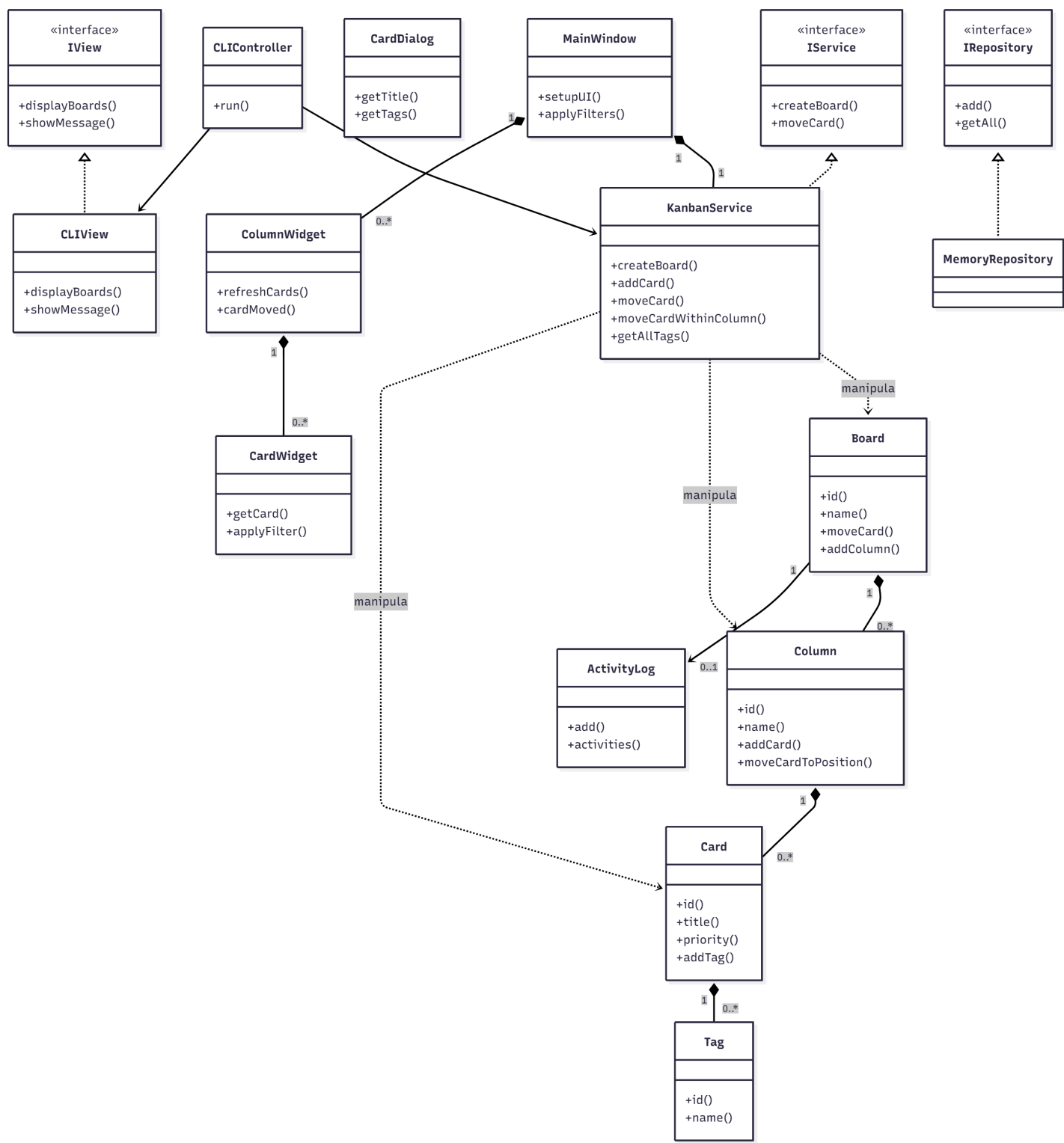


|

| PERSISTÊNCIA | ← Repositórios (Memória/Arquivo)

|

## 2. Diagrama de Classes Atualizado



## 3. Mapeamento dos Requisitos P00

### 3.1. Abstração & Encapsulamento

Onde foi atendido:

- **Interfaces claras:** `IService.h`, `IView.h`, `IRepository.h`, `IFilter.h`
- **Campos privados:** Todas as classes de domínio possuem campos privados
- **Getters/Setters:** Métodos como `Card::title()`, `Card::setTitle()`
- **Classes:** `domain/Board.h`, `domain/Card.h`, `domain/Column.h`

### 3.2. Classes e Objetos

Onde foi atendido:

- **Modelo de domínio:** `Board`, `Column`, `Card`, `User`, `ActivityLog`
- **Controllers:** `CLIController`, `KanbanService`
- **Views:** `CLIView`, GUI widgets (`MainWindow`, `ColumnWidget`, `CardWidget`)

### 3.3. Herança & Polimorfismo

Onde foi atendido:

- **Hierarquia com métodos virtuais:**
- `IService` → `KanbanService`
- `IView` → `CLIView`
- `IRepository<T>` → `MemoryRepository<T>`
- **Métodos virtuais puros:** Todos os métodos das interfaces são puros
- **Classes:** `interfaces/IService.h`, `interfaces/IView.h`

### 3.4. Composição vs Herança

Onde foi atendido:

- **Composição:**
- `Board` contém `vector<Column>`
- `Column` contém `vector<Card>`
- `Card` contém `vector<Tag>`
- `MainWindow` compõe `ColumnWidget` e `CardWidget`
- **Justificativa:** Relações "tem-um" são mais flexíveis que "é-um"

## 3.5. Polimorfismo dinâmico

Onde foi atendido:

- **Ponteiros polimórficos:** `std::shared_ptr<IService>`, `std::unique_ptr<IView>`
- **Uso de interfaces:** `KanbanService` implementa `IService`
- **Evita `dynamic_cast`:** Uso de interfaces em vez de RTTI

## 3.6. Gerenciamento de recursos

Onde foi atendido:

- **RAII:** Construtores/destrutores gerenciam recursos automaticamente
- **Smart pointers:**
- `std::unique_ptr<KanbanService>` em `MainWindow`
- `std::shared_ptr<Card>` em `Column::cards_`
- `std::shared_ptr<Board>` nos repositórios
- **Classes:** Todo o código base usa smart pointers

## 3.7. Templates e STL

Onde foi atendido:

- **Templates:** `MemoryRepository<T>`, `FileRepository<T>`
- **STL Containers:**

- `std::vector<std::shared_ptr<Card>>` em `Column`
- `std::map<Id, std::shared_ptr<T>>` em `MemoryRepository`
- `std::optional<std::shared_ptr<Board>>` em `KanbanService::findBoard()`
- **STL Algorithms:** `std::find_if`, `std::any_of` em operações de busca

## 3.8. Sobrecarga de operadores

Onde foi atendido:

- **Operador de saída:**
- `operator<<` para `Card`, `Tag`, `User`, `Activity`
- Implementado em `Card.cpp`, `Tag.cpp`, etc.
- **Operadores de comparação:**
- `Card::operator==` e `Card::operator<`
- `User::operator==` e `User::operator!=`

## 3.9. Tratamento de exceções

Onde foi atendido:

- **Exceções customizadas:** `MemoryRepositoryException`, `FileRepositoryException`
- **Captura adequada:** Try-catch em `main.cpp` e `MainWindow.cpp`
- **Mensagens ao usuário:** Via `CLIView::showError()` e `QMessageBox` na GUI
- **Classes:** `persistence/MemoryRepository.h`, `application/CLIView.cpp`

## 3.10. Documentação técnica

Onde foi atendido:

- **UML:** Diagrama de classes completo em Mermaid
- **Documentação Doxygen:** Todos os headers documentados
- **README:** Instruções de build e uso

- **Relatório técnico:** Este documento

## 3.11. Build automatizado

Onde foi atendido:

- **CMake:** `CMakeLists.txt` configurado para Linux e Windows
- **Dependências:** Qt6 declarada como dependência
- **Multi-plataforma:** Suporte a GCC, Clang e MSVC

## 4. Instruções de Build

### 4.1. Pré-requisitos

- **CMake 3.16+**
- **Compilador C++17** (GCC 9+, Clang 10+, MSVC 2019+)
- **Qt6** (opcional para GUI, obrigatório para build completo)

### 4.2. Build no Linux

```
# Clone o repositório
```

```
git clone https://github.com/seu-usuario/KanbanSystem-lite.git
```

```
cd KanbanSystem-lite
```

```
# Checkout da versão final
```

```
git checkout v3.0-final
```

```
# Configure o build
```

```
mkdir build && cd build
```

```
cmake -DCMAKE_BUILD_TYPE=Release ..
```

```
# Build (substitua 4 pelo número de cores da sua CPU)
```

```
make -j4
```

```
# Execute
```

```
./bin/kanban_gui # Para GUI
```

```
./bin/kanban_cli # Para CLI
```

### 4.3. Build no Windows

```
# Clone o repositório
```

```
git clone https://github.com/seu-usuario/KanbanSystem-lite.git
```

```
cd KanbanSystem-lite
```

```
# Checkout da versão final
```

```
git checkout v3.0-final
```

```
# Configure o build
```

```
mkdir build && cd build
```

```
cmake -G "Visual Studio 16 2019" -A x64 ..
```

```
# Build
```

```
cmake --build . --config Release
```

```
# Execute
```

```
.\bin\Release\kanban_gui.exe # Para GUI
```

```
.\bin\Release\kanban_cli.exe # Para CLI
```

## 4.4. Build sem Qt (apenas CLI)

```
cmake -DBUILD_GUI=OFF -DCMAKE_BUILD_TYPE=Release ..
```

```
make -j4
```








```
./bin/kanban_cli
```

## 5. Demonstração em Vídeo

**Link para o vídeo de demonstração:**

 [Assista à demonstração de 3 minutos](#)

**Conteúdo demonstrado no vídeo:**

1.  **Criação de boards e colunas**
2.  **Adição e edição de cards**
3.  **Movimentação de cards entre colunas**
4.  **Sistema de filtros por tags e prioridades**
5.  **Gerenciamento de tags**
6.  **Histórico de atividades**
7.  **Reordenação de cards dentro da coluna**



## 6. Tag no GitHub

**Versão Final:** `v3.0-final`

**Link do repositório:**

<https://github.com/Cigilo/KanbanSystem-lite>

**Para acessar a versão final:**

```
git clone https://github.com/Cigilo/KanbanSystem-lite

cd KanbanSystem-lite

git checkout v3.0-final
```

## 7. Conclusão

O Sistema Kanban Lite v3.0 representa uma implementação completa e robusta que demonstra domínio dos conceitos de POO em C++ moderno. A arquitetura bem definida, o uso apropriado de padrões de design e a integração harmoniosa entre domínio, aplicação e interfaces tornam este projeto um exemplo de boas práticas em engenharia de software.

**Principais conquistas:**

- ✅ Arquitetura em camadas bem definida
- ✅ Implementação completa de todos os requisitos POO
- ✅ Interfaces múltiplas (CLI e GUI) para o mesmo domínio
- ✅ Código moderno com C++17 e smart pointers
- ✅ Sistema de filtros e buscas avançado
- ✅ Documentação técnica completa
- ✅ Build automatizado multi-plataforma

O projeto está pronto para uso em ambiente acadêmico e serve como base sólida para futuras expansões e melhorias.

---

**Desenvolvido por:** João Pedro de Oliveira Ribas

**Data:** 06/10/24

**Disciplina:** Programação Orientada a Objetos (C++)

**Instituição:** UFPB