## Relatório Técnico — Etapa 1

Disciplina:	Programação Orientada a Objetos (C++)
Projeto:	Agenda (Planner/Organizer)
Aluno:	Vinícius da Silva Cunha
Matrícula:	20240011042
Data:	25/09/2025

## 1. Introdução

Este relatório apresenta o desenvolvimento da primeira etapa do projeto Agenda, proposto na disciplina de Programação Orientada a Objetos. O objetivo é construir uma aplicação modular em C++ que permita o gerenciamento de eventos, explorando os principais conceitos de POO e boas práticas modernas da linguagem.

## 2. Estrutura e Arquitetura

A etapa 1 contempla a definição da arquitetura do sistema, com a criação dos headers das principais classes e interfaces. Todos os arquivos seguem o padrão de modularização, encapsulamento e uso de recursos modernos do C++.

#### Arquivos criados:

- Event.h: Define a classe Event, que representa um evento na agenda, com título, descrição, horário de início e fim, tags e regra de recorrência.
- Calendar.h: Define a classe Calendar, responsável por armazenar eventos e associá-los a um usuário.
- RecurrenceRule.h: Interface abstrata para regras de recorrência, permitindo polimorfismo e extensibilidade futura.
- User.h: Define a classe User, representando o dono da agenda.
- IView.h: Interface para a camada de apresentação (CLI/GUI).
- IController.h: Interface para o controlador, conectando modelo e visualização.
- Persistence.h: Interface para persistência dos dados (JSON/SQLite).
- GuiView.h: Stub para futura implementação da interface gráfica.
- CMakeLists.txt: Arquivo de build automatizado usando CMake.

# 3. Conceitos de POO Aplicados

Encapsulamento: Todos os atributos das classes são privados, com acesso controlado por getters e setters.

 $Herança\ \&\ Polimorfismo:\ Uso\ de\ interfaces\ abstratas\ (Recurrence Rule,\ IView,\ IController,$ 

Persistence) e métodos virtuais, permitindo extensibilidade e polimorfismo dinâmico.

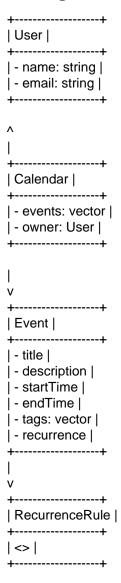
Composição: A classe Calendar contém uma coleção de eventos (std::vector>), e cada Event pode conter uma regra de recorrência.

Uso de STL: Utilização de std::vector para armazenar eventos e tags.

Smart Pointers: Uso de std::shared\_ptr para gerenciamento automático de memória e recursos. Sobrecarga de Operadores: A classe Event implementa operadores de comparação (<, ==) para facilitar ordenação e busca.

Modularização: Separação clara entre modelo, controlador, visualização e persistência. Build Automatizado: Utilização de CMake para facilitar compilação e integração futura.

## 4. Diagrama UML (Simplificado)



Outras interfaces: IView, IController, Persistence

# 5. Alinhamento com as Especificações

Todos os requisitos da Etapa 1 foram atendidos:

- Headers principais criados e documentados
- Encapsulamento, herança, polimorfismo, composição, STL e smart pointers presentes
- Modularização e interfaces claras
- Pronto para build automatizado com CMake
- Diagrama UML incluso
- Comentários explicativos em cada classe