1. Introdução

Este documento apresenta o projeto "Studio de Yoga", desenvolvido para a disciplina de Estruturas de Dados Orientadas a Objetos (CIN0135) do Centro de Informática (CIn) da UFPE.

O objetivo foi projetar e implementar um Sistema de Informação (Opção 1) funcional que aplicasse os conceitos fundamentais da Programação Orientada a Objetos (POO). O sistema escolhido gerencia as operações de um estúdio de yoga, permitindo o cadastro de praticantes e instrutores, o agendamento de aulas e a inscrição de alunos nessas aulas.

O sistema foi desenvolvido em C++, utiliza CMake para gerenciamento de build e SQLite para persistência de dados em um banco de dados relacional.

Utilizamos uma abordagem bottom-up para implementação do sistema, fazendo primeiro as classes e configuração do banco de dados para depois criarmos a UI.

2. Arquitetura do Sistema

Para garantir um design robusto, manutenível e que seguisse os princípios SOLID (especialmente o da Responsabilidade Única), optamos por uma arquitetura em 3 camadas (3-Layer Architecture).

Isso separa claramente as responsabilidades do nosso código em:

1. **Camada de UI (Interface do Usuário):** Responsável por interagir com o usuário. No nosso caso, é o arquivo main.cpp, que exibe menus no console e coleta dados (via std::cin).
2. **Camada de Serviços (Lógica de Negócios):** Contida na pasta services/, esta camada representa o "cérebro" do sistema. As classes aqui (Aula, Praticante) são responsáveis por aplicar as regras de negócio (ex: "uma aula está lotada?"). Elas operam em memória e **não sabem** que um banco de dados existe.
3. **Camada de Repositório (Acesso a Dados):** Contida na pasta repository/, esta camada é a única que "fala" com o banco de dados. Ela implementa o **Padrão de Design Repositório (DAO)**, encapsulando toda a lógica SQL (INSERT, SELECT, UPDATE, DELETE).

A classe DatabaseManager é responsável por abrir a conexão com o **SQLite** e inicializar as tabelas a partir do database/schema.sql.

3. Conceitos de POO Aplicados

A arquitetura foi projetada para aplicar diretamente os conceitos de POO exigidos:

- **Classes e Objetos:** O sistema é totalmente modelado em classes, como Pessoa,

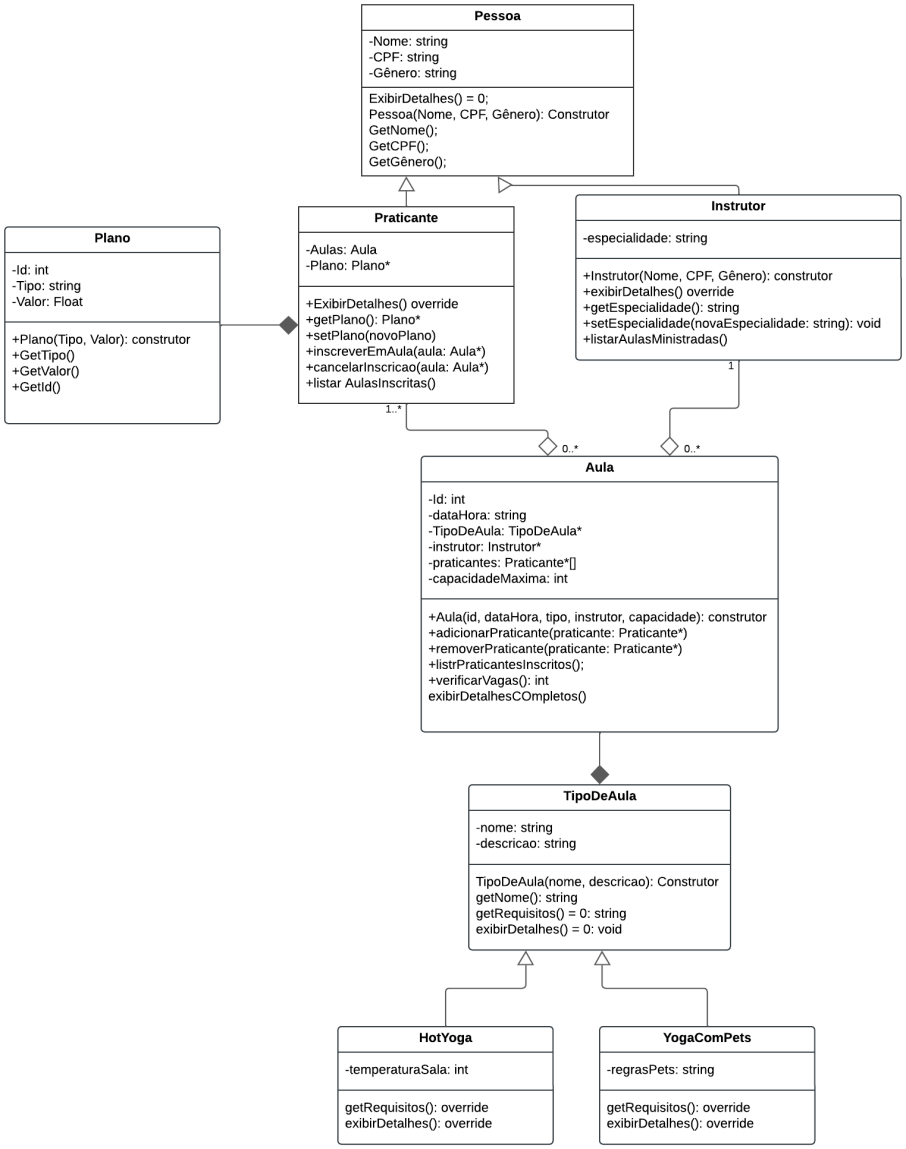
Aula, AulaRepository, DatabaseManager, etc.

- **Encapsulamento (Modificadores de Acesso):** Todos os atributos de nossas classes de serviço são private ou protected (ex: Aula::id). O acesso é controlado via métodos públicos (Getters e Setters), que validam os dados (ex: Aula::setDataHora).
- **Herança:** Utilizamos herança para reutilizar código e definir relações "É UM":
  - Praticante e Instrutor herdam da classe base Pessoa, compartilhando atributos como id, nome e email.
  - HotYoga e YogaComPets herdam da classe TipoDeAula.

- **Polimorfismo:** A herança de TipoDeAula é a base para o polimorfismo. Planejamos métodos virtuais (como virtual string getRequisitos()) que permitiriam a cada tipo de aula implementar regras específicas.
- **Padrão de Design (Bônus):** Implementamos o **Padrão Repositório** para isolar a lógica de persistência de dados (CRUD) da lógica de negócios, aderindo ao Princípio da Responsabilidade Única (SOLID).

4. Diagrama de Classes (UML)

Abaixo está o diagrama de classes que representa a arquitetura do nosso sistema.



5. Divisão de Tarefas

- **Nathan (Você):**
  - Responsável pela arquitetura do projeto (CMake, estrutura de pastas, princípio da responsabilidade única).
  - Implementação da integração com o banco de dados (SQLite, DatabaseManager).
  - Implementação da camada AulaRepository e da UI (main.cpp).

- **Membro 2:**
  - Responsável pela implementação das classes de serviço Pessoa, e Praticante.
  - Responsável pela implementação do CRUD (PraticanteRepository).

- **Membro 3:**

    - Responsável pela implementação das classes de serviço Instrutor e Plano.
    - Responsável pela implementação do CRUD (InstrutorRepository e PlanoRepository).
- **Membro 4:** Desenvolvedor Back-end e Documentação.
    - Responsável pela implementação da hierarquia TipoDeAula (Polimorfismo).
    - Responsável pelo CRUD (TipoDeAulaRepository).
    - Revisão final do código e preparação dos entregáveis (Relatório, Vídeo, GitHub Page).