Relatório RAD (Rapid Application Development) - Sistema de Administração de uma academia de Cross Training (CrossX)

1. Visão Geral do Projeto

O Sistema de Gerenciamento CrossX é uma aplicação web desenvolvida para auxiliar no gerenciamento de informações de alunos e registros de pagamentos para uma instalação de treinamento CrossX. O objetivo principal é otimizar tarefas administrativas, como matrícula de alunos, acompanhamento de mensalidades e controle de status de alunos.

2. Metodologia RAD Aplicada: Fases Detalhadas

A metodologia RAD foi a escolha ideal para este projeto, dada a necessidade de uma entrega eficiente e a capacidade de se adaptar rapidamente aos requisitos em evolução. As fases do RAD, aplicadas ao projeto CrossX, são descritas a seguir:

2.1. Fase de Planejamento de Requisitos

Esta fase inicial foca na compreensão das necessidades essenciais do negócio e na definição do escopo do projeto. No contexto do CrossX, isso teria envolvido:

- **Coleta de Requisitos Iniciais:** Através de workshops rápidos ou sessões de brainstorming, os requisitos essenciais foram levantados. Exemplos incluem:
    - Gerenciar informações básicas dos alunos (nome, contato, endereço).
    - Registrar e visualizar pagamentos.
    - Determinar o status de atividade do aluno (ativo/inativo).
- **Definição do Escopo do Projeto:** Foco em um sistema de gerenciamento básico para alunos e pagamentos, com funcionalidades CRUD (Criar, Ler, Atualizar, Deletar).
- **Seleção de Tecnologias:** Decisão por Flask para o backend e um frontend baseado em HTML/CSS/JavaScript (Tailwind CSS, Font Awesome) para agilidade no desenvolvimento. SQLite foi escolhido como banco de dados pela facilidade de uso em prototipagem.

2.2. Fase de Projeto do Usuário

Esta fase é altamente interativa e envolve a construção de protótipos funcionais que são apresentados aos usuários para feedback.

- **Protótipos Iterativos:** O **index.html** atua como o protótipo central, permitindo que os usuários visualizem e interajam com os formulários de cadastro de alunos e pagamentos, listas e botões de ação.
- **Feedback Contínuo:** Através da interação com o protótipo, o feedback sobre a usabilidade, a clareza dos campos e o fluxo de trabalho seria coletado rapidamente. Por exemplo, a forma como os dados do aluno são exibidos ou os campos para registro de pagamento.
- **Ajustes Rápidos:** Alterações no design da interface (cores, layout, campos) seriam implementadas rapidamente com base no feedback, usando as capacidades do CSS e JavaScript para modificações dinâmicas.

2.3. Fase de Construção

Nesta fase, o desenvolvimento real do código é acelerado, utilizando ferramentas e componentes pré-existentes sempre que possível.

- **Desenvolvimento de Módulos (Backend):** As rotas Flask para **alunos** e **pagamentos** foram construídas de forma modular, permitindo o desenvolvimento independente das funcionalidades de CRUD para cada entidade.
- **Componentes Reutilizáveis (Frontend):** Funções JavaScript como **mostrarMensagem**, **alternarSecaoAtiva** e **alternarVisibilidadeLista** demonstram a criação de componentes para reusabilidade e consistência na interface.
- **Testes Iterativos:** Testes unitários das APIs e testes de integração do frontend com o backend seriam realizados continuamente à medida que cada funcionalidade é construída, garantindo a qualidade desde o início. Por exemplo, a validação de campos obrigatórios no backend.

2.4. Fase de Transição

A fase final do RAD envolve a implantação do sistema e a preparação para o uso em produção.

- **Implantação Simples:** Para um ambiente de desenvolvimento ou testes iniciais, a execução do app.py com debug=True permite uma implantação rápida.
- **Criação de Banco de Dados:** O db.create_all() garante que o esquema do banco de dados seja criado automaticamente na inicialização da aplicação, facilitando a configuração.

- **Treinamento de Usuários:** Com uma interface intuitiva e funcionalidades claras, o treinamento necessário para os usuários finais é minimizado.
- **Validação Final:** Testes de aceitação do usuário seriam realizados para garantir que o sistema atenda aos requisitos finais antes da implantação completa.

3. Arquitetura da Aplicação

A aplicação segue um padrão arquitetural simplificado de Model-View-Controller (MVC) utilizando o framework Flask:

- **Modelo (app/model/models.py):** Define a estrutura do banco de dados e as classes Aluno e Pagamento utilizando Flask-SQLAlchemy para ORM. O banco de dados é SQLite (crossx.db).
- **Visão (app/templates/index.html):** Uma única página HTML com Tailwind CSS para estilização e JavaScript para manipulação dinâmica do DOM e comunicação com o backend via APIs.
- **Controlador (Rotas - app/routes.py):** Define os endpoints da API para operações CRUD (Criar, Ler, Atualizar, Deletar) em alunos e pagamentos.
- **Aplicação Core (app/__init__.py, app.py):** app/__init__.py cria e configura a aplicação Flask, inicializa o banco de dados e registra as rotas. app.py é o ponto de entrada principal, que cria a aplicação e habilita o CORS.

4. Funcionalidades Chave

- **Gestão de Alunos:** Cadastro, listagem, busca, visualização, atualização e exclusão de alunos.
- **Gestão de Pagamentos:** Listagem, registro e exclusão de pagamentos.
- **Interface Web Dinâmica:** Frontend de página única com JavaScript para interações sem recarregamento da página.
- **Integração com Banco de Dados:** SQLite e Flask-SQLAlchemy para persistência de dados.
- **CORS Habilitado:** Permite a comunicação entre diferentes origens do frontend e backend.

5. Conclusão

O projeto CrossX demonstra um excelente uso da metodologia RAD, focando na entrega rápida e funcional de um sistema de gerenciamento essencial. A estrutura modular, a escolha de tecnologias adequadas e o ciclo de desenvolvimento iterativo e interativo com o usuário (mesmo que implícito na estrutura do código) são indicativos de uma abordagem RAD bem-sucedida. O sistema está pronto para ser expandido com funcionalidades adicionais em futuras iterações, mantendo a agilidade característica do RAD.