**Centro Universitário Estácio do Ceará**

**PARANGABA**

**Projeto Cross Training**

**Igor Stenio Pereira Santos**

**Luan Victor Dias Lopes**  
**Vitor Vinícius Lima Silva**

**Willian**

**Lian Wille Matos da Silva Lino**

**Orientador: Cynthia Maia**

**2025**

**Fortaleza/CE**

Sumário

[**1.** Planejamento de requisitos 3](#_Toc199157243)

[1.1. Identificação das partes interessadas e parceiros 3](#_Toc199157244)

[1.2. Problemática e/ou problemas identificados 3](#_Toc199157245)

[1.3. Justificativa 3](#_Toc199157246)

[1.4. Objetivos/resultados/efeitos a serem alcançados (em relação ao problema identificado e sob a perspectiva dos públicos envolvidos) 3](#_Toc199157247)

[2. Design do usuário 3](#_Toc199157248)

[2.1. Protótipo 3](#_Toc199157249)

[3. Construção 3](#_Toc199157250)

[3.1. Refinando o protótipo e criando um entregável 3](#_Toc199157251)

[4. Transição 3](#_Toc199157252)

[4.1. Testes de integração 3](#_Toc199157253)

[4.2. Implementação 3](#_Toc199157254)

# Planejamento de requisitos

## Definição do escopo

O projeto Cross Training se iniciou com o anseio da responsável pelo estabelecimento em criar um sistema que fosse capaz de gerenciar as principais informações do seu negócio em um só lugar, deixando de lado a caderneta de anotações e possibilitando consultas mais eficientes.

Para que este objetivo pudesse ser efetivamente aplicado, foi estabelecido o seguinte escopo do projeto em conjunto a todas os Stakeholder do projeto:

|  |  |
| --- | --- |
| Manter informação de contato de alunos | Um novo aluno é adicionado ao sistema. Manter a seguinte informação de contato dos alunos: Nome, Endereço, Cidade, Estado, Telefone. "Manter" significa que podem ser feitas 4 operações (ou CRUD): Create, Read, Update, Delete. |
| Alterar estado "matriculado" do cadastro | Uma pessoa cadastrada poderá ser aluna ou não. Deve-se manter a data de matrícula e a data em que o aluno deixa de ser aluno, data de vencimento da matrícula (mas sem eliminar o cadastro). Não se deve remover um registro se a pessoa ainda for aluno (tipicamente, você não joga fora seu cadastro, mesmo depois que alguém deixa de ser aluno!). |
| Registrar pagamento | Registrar todos os pagamentos feitos pelo aluno (valor, data, tipo). Cada pagamento tem um tipo. Não precisa fazer cadastro de novos tipos de pagamentos. Os tipos de pagamentos podem ser dinheiro ou cartão. |

# Design do usuário

## Criação do Protótipo

Para o desenvolvimento rápido de um protótipo para a validação do cliente, foi utilizado o modelo de projeto MTV (Model, Templates e View) em conjunto ao framework *Flask* e todas suas dependências, buscando atingir o desenvolvimento rápido do projeto.

Deste modo, foram desenvolvidos seguindo a estrutura de projeto menciona 4 arquivos bases para testar a viabilidade de execução do escopo mencionado no tópico 1.1, sendo 2 destes referentes a configuração do servidor *Flask* e a integração das funcionabilidades, estes arquivos foram denominados de *app.py* e *\_\_init\_\_.py,* para o desenvolvimento das funcionabilidades e criação de rotas foram criados mais 2 arquivos denominado *aluno\_view.py* e *aluno\_rotas.py.*

Do ponto de vista técnico, alguns detalhes são relevantes, para a realização de testes foi criado um ambiente de desenvolvimento no *Flask* no *IP local na porta 8080* e instanciado alguns *endpoints* no modelo API que recebe dados em *Json* e retorna respostam também neste modelo.

Após o desenvolvimento do protótipo a partir dos modelos mencionados nos parágrafos anteriores, foi entregue o protótipo do projeto para teste dos Stakeholders, e este já foi capaz de satisfazer duas das três principais exigências mencionadas no escopo do projeto.

# Construção

## Refinando o protótipo e criando um entregável

Após a validação do protótipo inicial, teve início a fase de desenvolvimento efetivo do sistema, com o objetivo de transformá-lo em uma aplicação mais estável, organizada e apta a receber futuras melhorias. A estrutura do projeto foi concebida com base no padrão arquitetural MVC (Model-View-Controller), utilizando o framework Flask, o que contribuiu significativamente para a clareza e manutenção do código.

O uso da metodologia RAD começou com a elaboração de um protótipo funcional que abrangia as funcionalidades essenciais da gestão de alunos, como o registro, a listagem e a edição de dados. Essa primeira versão foi essencial para testar ideias e realizar experimentações com outras ferramentas e abordagens. Por meio desses testes, foi possível observar na prática os pontos fortes e as limitações do sistema, orientando ajustes necessários para o avanço do projeto.

Em vez de seguir uma abordagem tradicional e sequencial, o desenvolvimento foi conduzido por meio de ciclos curtos e iterativos. A cada nova iteração, recursos adicionais eram incorporados e imediatamente submetidos a testes, o que possibilitou rápidas correções de erros e ajustes com base nas necessidades identificadas. Um exemplo prático dessa evolução foi a melhoria da lógica de exclusão de alunos, que passou a impedir a remoção de registros ainda ativos, protegendo a integridade dos dados. Outra melhoria significativa foi a implementação automática do cálculo da próxima data de vencimento no módulo de pagamentos, sempre que um novo pagamento fosse registrado.

Durante todas as etapas, a estrutura do sistema permaneceu bem segmentada. O projeto foi dividido de maneira clara entre as camadas de modelos, rotas e controle de lógica, conforme os princípios do padrão MVC. O arquivo principal da aplicação, app.py, ficou responsável por iniciar o servidor Flask e definir as configurações do banco de dados, enquanto as funcionalidades específicas foram organizadas em pastas separadas, facilitando a manutenção, a leitura do código e a possibilidade de expansão futura.

Além disso, a comunicação de dados entre partes da aplicação foi padronizada no formato JSON, o que torna o sistema facilmente integrável com interfaces web ou aplicativos móveis. Foram também adicionadas rotinas de validação de dados e tratamento de exceções, com o objetivo de garantir a estabilidade e a confiabilidade da aplicação durante o uso.

Com a estrutura devidamente modularizada e uma clara separação entre os componentes do sistema, o projeto final apresenta uma base sólida, pronta para evoluir conforme novas necessidades surgirem. A versão entregue contempla todos os requisitos funcionais definidos no início do projeto e está preparada para ser utilizada em um ambiente real, exigindo apenas pequenos ajustes adicionais relacionados à segurança e autenticação.

# Transição

## Testes de integração

Para checar os resultados obtidos e a consistência dos dados enviados, foram realizados testes de integração. O objetivo dos testes foi validar o funcionamento das rotas de alunos e pagamentos, bem como garantir a integridade dos dados e o comportamento esperado nas operações de criação, listagem, atualização, desmatrícula e exclusão de registros juntamente com as operações de registros de pagamento. Para isso, utilizou-se a extensão Thunder Client, integrada ao Visual Studio Code, que permite simular requisições HTTP, possibilitando uma verificação detalhada do comportamento da API.

No módulo de alunos, foram testadas as rotas de criação, listagem, atualização, desmatrícula e exclusão. O fluxo de criação de aluno foi validado com sucesso, onde os dados fornecidos foram corretamente armazenados no banco e as respostas da API confirmaram a operação apropriadamente. Na sequência, realizou-se a listagem dos alunos cadastrados, e os dados retornaram corretamente, incluindo os formatos de data em “YYYY-MM-DD", demonstrando que o processo de serialização dos dados estava funcional.

O processo de atualização de informações de um aluno também foi validado, permitindo alterações sem a necessidade de reenviar todos os dados. A rota de desmatrícula foi testada focando na lógica de negócio, que impede a desmatrícula de um aluno que não possui matrícula ativa ou que já esteja desligado, retornando mensagens de erro adequadas quando estas situações ocorreram. Da mesma forma, a tentativa de excluir um aluno que ainda não estivesse previamente desligado foi bloqueada, exigindo que o aluno passe pelo processo de desmatrícula antes da exclusão definitiva no banco de dados.

Em relação ao módulo de pagamentos, os testes também demonstraram resultados. A criação de um pagamento foi corretamente vinculada ao respectivo aluno, com a atualização automática do status do aluno, além do ajuste das datas de matrícula e vencimento. A cada novo pagamento, o sistema incrementa a data de vencimento em 30 dias, respeitando casos de alunos recém-matriculados e aqueles que já possuem pagamentos anteriores registrados.

As rotas de listagem de pagamentos funcionaram como esperado, trazendo as transações associadas ao aluno consultado. Além disso, as funcionalidades de atualização e exclusão de pagamentos foram devidamente validadas, permitindo a correção de valores ou tipos de pagamento, bem como a remoção de registros, mantendo a consistência dos dados.

Durante todo o processo de teste restante, não foram identificados erros críticos, exceções não tratadas ou falhas no fluxo das operações. As mensagens de erro implementadas auxiliaram na compreensão dos motivos das operações inválidas, reforçando a robustez das validações. As relações entre alunos e pagamentos, também se comportaram de forma consistente, garantindo a integridade referencial.

Portanto, conclui-se que os testes realizados demonstraram que o sistema está funcionando de acordo com os requisitos esperados. As rotas operam corretamente, os dados estão sendo persistidos e atualizados de maneira adequada no banco de dados, e as regras de negócio implementadas estão sendo rigorosamente respeitadas.

## Implementação

O processo de implementação do sistema começou com a etapa de identificação das necessidades para gerenciar alunos e controlar finanças. Decidiu-se que seria necessário criar uma API que possibilitasse o registro de alunos, controle de pagamentos e acompanhamento de status, como matrícula ativa, pendente ou cancelada.  
 A primeira fase prática envolveu a definição da estrutura do banco de dados, utilizando SQLAlchemy como ORM para facilitar o relacionamento entre tabelas e registros de forma orientada a objetos. Nessa fase, foram projetadas classes representando as tabelas principais: Aluno e Pagamento, onde cada atributo da classe equivalia a uma coluna no banco. O modelo considerou campos essenciais como nome, status, datas de matrícula, vencimento e desligamento para alunos, além de data, valor e tipo de pagamento na entidade Pagamento.  
 Em seguida, foi iniciada a construção da API com o framework Flask. Criaram-se endpoints REST que viabilizam operações CRUD para alunos, além de rotas específicas para gerenciar pagamentos. Durante essa fase, uma prioridade foi implementar regras de negócio que assegurassem a consistência dos dados. Por exemplo, ao processar um pagamento, o sistema precisava atualizar automaticamente o status do aluno para “matriculado”, além de registrar as datas de matrícula e calcular corretamente os vencimentos, conforme a situação de cadastro do aluno.  
 Além disso, foi imposto um fluxo rigoroso de validação dos dados recebidos nas requisições. Foram feitas checagens para garantir que os campos essenciais estivessem adequadamente preenchidos, como o nome do aluno, o valor do pagamento e o tipo de transação financeira. As respostas da API foram organizadas para oferecer mensagens claras e diretas, facilitando a compreensão dos usuários e promovendo uma experiência mais eficiente durante o uso da aplicação. Os testes foram realizados de maneira incremental, validando cada endpoint à medida que era implementado e corrigindo qualquer inconsistência que aparecesse.  
 Para concluir, a implementação seguiu um método iterativo, onde cada funcionalidade foi desenvolvida, testada e ajustada gradualmente. Esse procedimento não só garantiu o funcionamento correto do sistema, mas também a criação de uma base sólida, pronta para receber futuras expansões com novas funcionalidades futuramente.