

# Projeto Brasileiro 2025

Engenharia de Software e Arquitetura de Sistemas

GRUPO DE PROJETO:

- Lucas Oliveira
- Matheus Rossaneze
- Nicolas Morales
- Pedro Schaurich
- Phelipe Antonio

PROFESSORA: Lucy Mari

INSTITUIÇÃO: FECAP - 2025/2

# O Projeto: Funcionalidades e Requisitos Validados

## Funcionalidades-Chave

Visualizar os **20 times da Série A 2025**

- Consultar informações (elenco, estatísticas)
- Analisar dados históricos (gols, cartões, partidas)
- Filtros dinâmicos e gráficos interativos para análise

## Tecnologias Utilizadas

Construído com uma stack moderna para garantir performance e escalabilidade:

**Backend:** Python, Pandas

**Frontend:** Streamlit

**Visualização:** Plotly

**Dados:** Kaggle e Scoreaxis

## Engenharia de Requisitos: Foco no Usuário

### Pesquisa Quantitativa

32 usuários entrevistados. Público majoritário de 15-25 anos (81,25%), garantindo aderência ao target.

### Feedback Qualitativo

Principal necessidade (João): **"O sistema deve atualizar constantemente"**. Isso priorizou a integração com APIs em tempo real.

✓ **Requisitos Validados e Implementados** com base nas necessidades do usuário final, garantindo relevância.

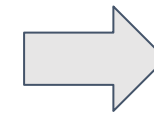
# Design de Software – Princípios SOLID

A aplicação dos princípios SOLID garante um código robusto, flexível e de fácil manutenção. O **SRP (Single Responsibility Principle)** é o pilar da nossa organização.

```
# utils/db_utils.py - Single Responsibility Principle
def get_lista_times():
    return TIMES_SERIE_A_2025
# Apenas retorna lista

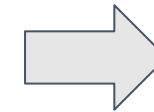
def normalizar_nome_time(time: str):
# Apenas normaliza nomes (e.g., para busca de escudo)

def get_escudo_path(time: str):
# Apenas busca caminho do escudo (.png)
def get_team_id(time: str):
# Apenas retorna ID único do time (API Scoreaxis)
```



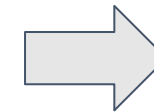
## Responsabilidade Única

Cada função tem **UMA** e **SOMENTE UMA** responsabilidade clara.



## Manutenção Facilitada

Se precisar alterar o caminho do escudo, não afeta a lógica de normalização do nome. Testes isolados.



## SRP em Ação!

O módulo de utilitários de dados é coeso e minimamente acoplado.

**Outros Princípios Aplicados:** Também implementamos o **Open/Closed Principle** (extensível, mas fechado para modificação) e o **Dependency Inversion Principle** (uso de abstrações).

# Design Patterns – Otimização com Padrões

Utilizamos padrões de projeto para resolver problemas comuns de forma eficiente e estruturada.

## Singleton

`@st.cache_data` do Streamlit atua como Singleton para DataFrames.

**Benefício:** Performance +80% ao evitar recarregamento desnecessário do CSV.

## Facade

`def`  
`mostrar_elenco(team_id):`  
encapsula a complexidade do IFrame e da API de terceiros.

**Benefício:** Oferece uma interface simples e limpa para a camada de apresentação.

## Adapter

`def`  
`normalizar_nome_time(time)`  
: transforma nomes de times para o formato de arquivo (e.g., "São Paulo" → "sao\_paulo.png").  
**Benefício:** Integração perfeita entre diferentes formatos de dados e fontes.

## Observer

Componentes Streamlit (`st.multiselect(..., key="rodada")`) monitoram o estado.

**Benefício:** Interface reativa, atualizando gráficos e tabelas automaticamente a cada filtro.


# Arquitetura em Camadas: Separação em 4 camadas

Adotamos uma arquitetura hierárquica para estruturar o sistema, isolando a interface do usuário da lógica de dados.

## CAMADA 1:

### APRESENTAÇÃO


Pagina\_Inicial.py, pages/\*

 **Foco:** Interface do usuário e interação (Streamlit).

## CAMADA 2:

### SERVIÇOS


service\_kaggle, scoreaxis

 **Foco:** Lógica de negócio, como cálculo de estatísticas e manipulação de APIs.

## CAMADA 3:


### UTILITÁRIOS

db\_utils.py, Funções de limpeza

 **Foco:** Funções auxiliares, como normalização de nomes e busca de escudos.

## CAMADA 4: DADOS

Kaggle + Scoreaxis APIs

 **Foco:** Fontes primárias e secundárias de dados brutos.

## Benefícios Estruturais

**Separação Clara:** Mudanças na UI não afetam o acesso a dados.

**Fácil Manutenção:** Debugging isolado em cada camada.

**Evolução Independente:** Permite a troca de uma fonte de dados (Camada 4) sem refatorar a lógica de negócio (Camada 2).

# Qualidade de Software: Metodologia de Testes

Um plano de testes foi planejado para garantir a confiabilidade do sistema e a aderência aos requisitos.

Categoria	IDs	Descrição e Foco Principal
Unitários (UT)	1-6	Verificação de funções isoladas no módulo de utilitários (db_utils).
Integração (IT)	1-5	Conexão e retorno de dados corretos das APIs externas (Kaggle e Scoreaxis).
Funcionais (FT)	1-6	Testes de UI/UX, navegação entre páginas e precisão dos dados exibidos.
Robustez (RB)	1-5	Validação de <i>fallbacks</i> e tratamento de erros (dados ausentes ou conexões falhas).
Desempenho (DP)	1-3	Tempo de carregamento inicial e latência após aplicação de filtros.
Usabilidade (US)	1-3	Responsividade e facilidade de uso em diferentes dispositivos.

## Resultado (Planejado)

✔ Todos os casos conforme esperado..

# Manutenção e Evolução Contínua

## Estudo de Caso: Otimização Pós-Revisão

### ✗ ANTES

```
st.dataframe(tabela)
```

Dados ausentes (NaN) eram exibidos como "None" na interface, tornando-a **confusa** para o usuário final.

formacao_manda
None

- **Padronizar dados ausentes:** Implementado (substituição de NaN por " ").

### ✓ DEPOIS (FEEDBACK HUMANO)

```
tabela = tabela.fillna(" ")st.dataframe(tabela)
```

Implementação de tratamento de valores. O usuário vê um **espaço limpo** ou um traço, melhorando a experiência (UI/UX).

posicao

- **Adicionar gráficos:** Implementado com Plotly para visualizações interativas.

# DevOps e Feedback: Ciclo de Melhoria Contínua

## ⚠ Implementação Teórica de DevOps ⚠

Embora o deployment não seja totalmente automatizado neste ambiente, os conceitos de infraestrutura como código e pipeline foram o foco do estudo.

### Containerização (Docker)

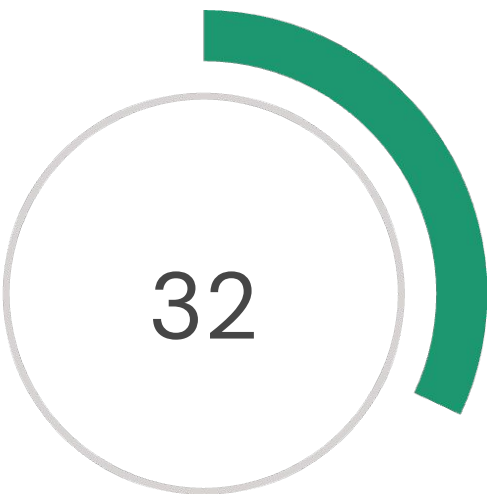
Definição de `Dockerfile` e `Docker Compose` para garantir a portabilidade do ambiente em qualquer sistema operacional.

### CI/CD Pipeline

Mapeamento de um pipeline ideal, incluindo etapas de *Code Quality*, **Testes Automatizados**, Build da imagem e Deploy para produção.

---

## Feedback e Reavaliação



### Pesquisa Inicial

Base para Engenharia de Requisitos.

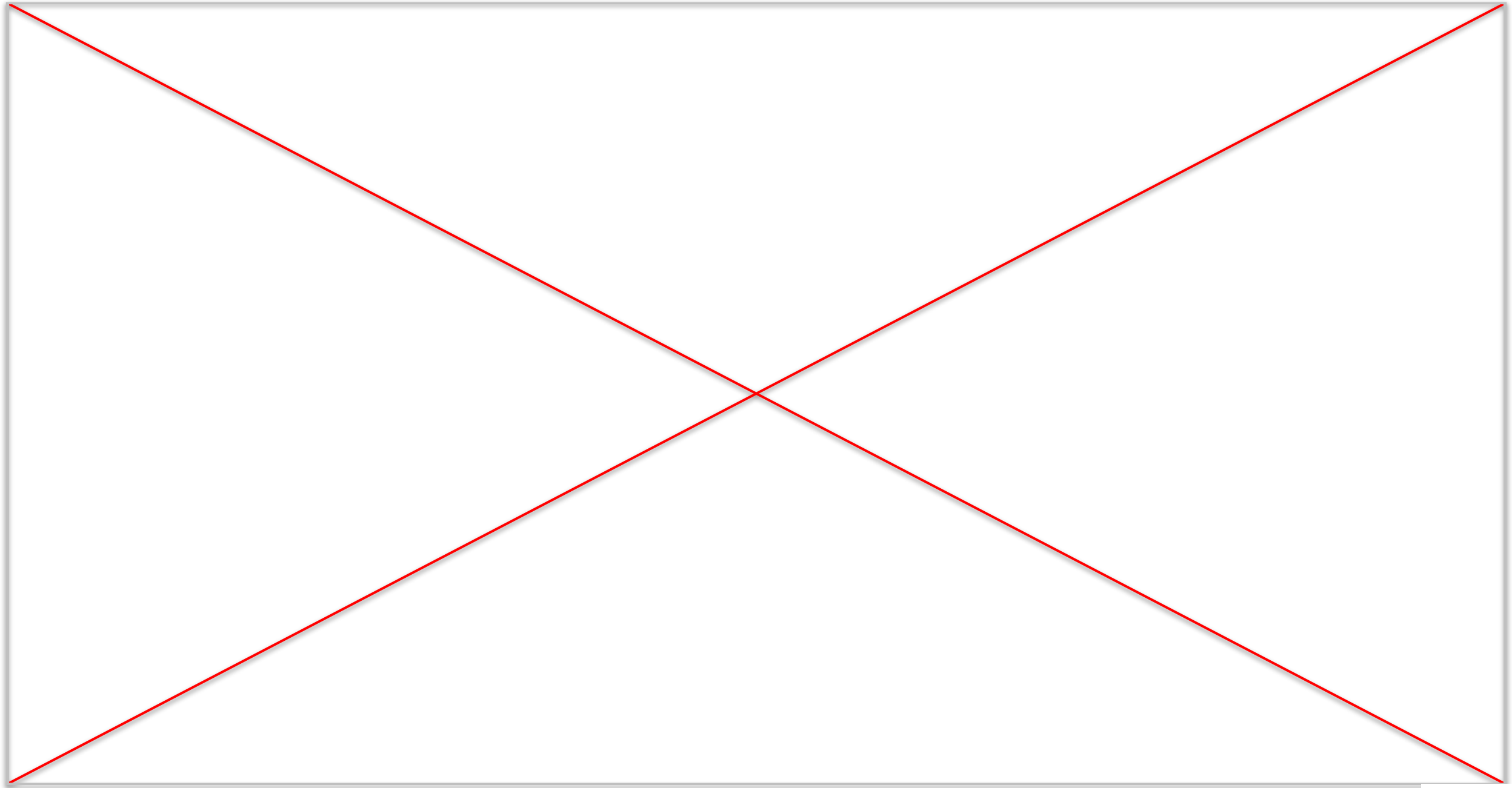


# Conceitos Aplicados – Resumo Final

O projeto demonstrou a aplicação prática e integrada dos principais conceitos da disciplina de Engenharia de Software.

01	02	03	04
ENGENHARIA DE REQUISITOS	DESIGN DE SOFTWARE	DESIGN PATTERNS	ARQUITETURA DE SISTEMAS
Validação com 32 usuários e entrevista qualitativa definiram o escopo.	Princípios SOLID aplicados para garantir modularidade e baixa complexidade.	Padrões identificados e utilizados para otimização.	Modelo de 4 camadas (Apresentação, Serviços, Utilitários, Dados).
05	06	07	08
QUALIDADE DE SOFTWARE	MANUTENÇÃO E EVOLUÇÃO	FEEDBACK E REAVALIAÇÃO	DEVOPS
30 casos de teste planejados	Processo de <i>Code Review</i> implementado para melhorias contínuas.	Coleta e incorporação de feedback de 32 usuários.	Conceitos de Containerização (Docker) e CI/CD estudados e mapeados.

# Vídeo do Projeto



# Obrigado.

Estamos à disposição para a discussão!

 GRUPO

Lucas, Matheus, Nicolas, Pedro, Phelipe

 GITHUB

[GitHub - schaurxch/projeto-brasileirao-streamlit](https://github.com/schaurxch/projeto-brasileirao-streamlit)