Relatório Acadêmico

Equipe:
Raví Fernandes - 3001
Ewerton Ferreira - 3001
Ana Gabryella - 3001

Curso - Análise e desenvolvimento de sistemas
Orientador:Lucas Gomes

Matéria:Desenvolvimento Web em Html5,CSS,JavaScript e Php

Tema : Desenvolvimento de um sistema Web completo com Zend Framework e Doctrine ORM(**CRUD Completo**)

CRUD:

- Cadastro de alunos ou usuários,tendo que conter:Tela de login com autenticação
- Tela de listagem,inclusão e exclusão de registros
- Estrutura MVC com Zend Framework
- Persistência de dados via Doctrine e ORM

SUMÁRIO

1 - INFORMAÇÕES GERAIS

Site fictício unicamente usado para apresentação e dissertação deste trabalho, nesse projeto foi utilizado a ideia de criar uma instituição para que seja utilizado nossos conhecimentos aprendidos em sala de aula, referente a criação e execução de linguagem de marcação de hipertexto(HTML), o modelo CRUD é um conceito fundamental e amplamente utilizado no desenvolvimento de sistemas e bancos de dados, sendo um acrônimo para as quatro operações básicas de manipulação de dados em sistemas com armazenamento persistente (**Create, Read, Update, Delete**).

1.2 - OBJETIVO

Primordialmente , tivemos a ideia de criar um clube de futebol imaginário baseado no Sport(Sport club do recife),porém achamos melhor uma escolinha que se enquadra mais no que todos estavam discutindo,pensamos num selo,numa logo,forma de entrar e até uniformes, mas preferimos focar no principal que é a respeito do cadastro, e de como iria se parecer esse site.

O nome” Leões Imperiais ‘’, foi basicamente um brainstorm de todos, e escolhemos o mais imponente dentre as opções.

Paleta de cores (Vinho Escuro, Vinho Vibrante, Azul Acinzentado e Branco)

Logos feitas, e posteriormente discutidas para ser a que representaria o projeto, tendo base que seria um leão num escudo, foi facilmente montada utilizando canvas e tendo auxílio de IA

1.3 - STAKEHOLDERS

Desenvolvido por **Equipe Leões**

Membros : Ana Gabryella, Ewerton Ferreira e Raví Fernandes
Avaliador:Lucas Gomes

---

2 - ARQUITETURA E TECNOLOGIA

2.2 - LINGUAGENS USADAS

Foram utilizados:
Frontend (Cliente): HTML, CSS, JavaScript (e qualquer framework/biblioteca JS).
Backend (Servidor): PHP.
Banco de Dados**:** SQL.
ZendFramework ou Laminas
Doctrine

HTML (Hypertext Markup Language)

Função: É a espinha dorsal de qualquer página web. O HTML é usado para estruturar o conteúdo (títulos, parágrafos, imagens, links, formulários, etc.) através de marcações (tags), definindo a estrutura semântica e a hierarquia da informação.

CSS (Cascading Style Sheets)

Função: Responsável por toda a apresentação visual e design do projeto. O CSS é utilizado para estilizar os elementos estruturados pelo HTML, controlando aspectos como cores, fontes, layouts, espaçamentos, animações e a responsividade (adaptação a diferentes tamanhos de tela).

JavaScript

Função: É a linguagem de programação que adiciona interatividade e comportamento dinâmico às páginas web. No lado do frontend, permite a manipulação do DOM (Document Object Model), validação de formulários, comunicação assíncrona com o servidor (AJAX/Fetch) e a criação de uma experiência de usuário rica e fluida.

PHP (PHP: Hypertext Preprocessor)

Função: Uma linguagem de script de lado do servidor (backend). O PHP é usado para processar dados, gerenciar a lógica de negócios, interagir com o banco de dados (como o SQL), manipular sessões de usuário, realizar autenticação e gerar o conteúdo dinâmico do HTML que será enviado ao navegador.

SQL (Structured Query Language)

Função: Não é uma linguagem de programação, mas sim uma linguagem de consulta padrão para gerenciar e manipular bancos de dados relacionais. O SQL é utilizado para realizar operações de CRUD (Create, Read, Update, Delete) nos dados do projeto, garantindo a persistência e a recuperação eficiente das informações.

Zendframework(laminas)

O Zend Framework (ZF) foi um framework de código aberto, robusto e de nível empresarial, desenvolvido para o PHP, que promovia as melhores práticas de desenvolvimento com base em padrões de design.

Doctrine

Doctrine é um conjunto de bibliotecas PHP que fornecem uma camada de Mapeamento Objeto-Relacional (ORM) e ferramentas de banco de dados. Ele permite que desenvolvedores trabalhem com objetos PHP em vez de comandos SQL complexos para interagir com o banco de dados.O Doctrine é frequentemente usado em grandes *frameworks* PHP, como o antigo Zend Framework/atual Laminas, Symfony e Laravel.

2.3 - BANCO DE DADOS

Foi utilizado MYSQL para o banco de dados do projeto.
Database = projeto-web
Tabelas principais (onde são guardados os dados)

- 1”**jogadores**” : id,nome,data de nascimento,cpf,email,telefone,posição,categoria,cidade,estado,data de inscrição e id da peneira.
- 2”**usuarios**”: id,nome,email,senha,tipo de usuários e data de criação

- Sem relacionamentos complexos apenas tabelas diretas
- Arquivos de estrutura = A estrutura do banco pode ser importada pelo arquivo: **Projeto-Web\Projeto-Web\app\config**

2.4 - ORGANIZAÇÃO DE PASTAS

Projeto-Web

/app - (contém os controladores modelos)
-controller - (lógica principal das rotas e ações do usuário)
-models - (interação com o banco de dados)
-uploads - (pasta para onde vai os downloads das operações)

/cronograma - (perto de logística de construção do projeto)

/documentação - (obrigatório para instalar e tirar dúvidas a respeito do funcionamento do site)

/public - (arquivos públicos)
-admin - (area do moderador, onde tem o painel de controle)
-css - (parte de estilização de todo o projeto)
-js - (parte operacional, onde os comandos são requisitados para outras áreas)

/zend - (é um framework, uma galeria de api onde proporciona robustez e flexibilidade na preparação de site e aplicações)
-auth - (Este arquivo define a classe LoginService, que encapsula toda a lógica de negócios para a autenticação de usuários na aplicação)
-entity - (Define a entidade “usuário”, que representa a tabela de usuário do banco de dados)
---

3 - COMO EXECUTAR O PROJETO

3.1 - GUIA DE INSTALAÇÃO

- 1 Requisitos obrigatórios para rodar o projeto

- Xampp(para rodar o sql e php localmente)
- Navegador web(chrome,firefox ou edge)
- (Opcional)Editor de registro(visual code)
- Clonar o repositório (git clone)
- (https://github.com/ewertonsf/Projeto-Web)

- 2 Configurações do projeto

dentro desta pasta C:\xampp\htdocs\ , colocar o arquivo projeto-web dentro

- 3 Passos para execução

Após a instalação, abra o **XAMPP Control Panel** e inicie os módulos:Na opção “start”
- Apache
- MySQL

- 4 Configurar o banco de dados em config/conexão.php
- 5 Executar localmente com o servidor embutido do php:php -S localhost:8000 -t public

---

4 - FUNCIONALIDADES PRINCIPAIS

4.1 - RESUMO DA PÁGINA PRINCIPAL

- 1 Página principal

-Banner 1 ,Banner 2 e textos
-Navbar
-Recursos
-Formulário de inscrição
-Footer ou Rodapé
-Botões

Permite que o cliente cheque todas as informações a respeito da peneira, e faz com que ele seja fisgado caso tenha interesse na proposta, e direcionado para o formulário, e por fim sendo consolidado como um consumidor de nosso sistema

- 2 Contato

-Contatos

Aba feita unicamente para orientar os canais por onde o cliente terá acesso para entrar em contato com a equipe Leões, via email,telefone e endereço,

- 3 O Portal

-Portal do jogador

Aba onde mostra uma apresentação do que se trata o projeto,sendo basicamente uma introdução para quem não sabe de muito, a respeito do projeto em si.

- 4 Área de Cadastro

-Banner
-Botões
-Informações de cadastro

Página onde o candidato irá colocar suas informações pessoais em nosso banco de dados, para que seja avaliado pelos administradores e posteriormente introduzidos em nosso projeto, caso seu perfil se encaixe.

Candidato irá colocar, uma foto ou vídeos de seus talentos ou passagens por outros clubes(mostrar experiência e que realmente quer fazer parte do projeto), cpf,email,celular,sua cidade,estado,categoria,posição e data de nascimento

- 5 Área de Login

-Banner
-Email
-Senha
-Botões

Área de login será unicamente para o administrador do projeto conseguir logar e avaliar cada participante

4.2 - BACK-END

- JS - **principal.js**

O arquivo principal.js gerencia a interface de exibição dos jogadores na página principal. Ele é responsável por iniciar a comunicação assíncrona com o backend (via Fetch API) para buscar a lista de atletas, processar a resposta e renderizar os dados dinamicamente no container cardsJogadores do HTML.

Inicialização (`DOMContentLoaded`): Dispara a função `carregarJogadores()` imediatamente após o DOM estar pronto, garantindo que o conteúdo seja carregado ao abrir a página.
Formatação de Data (formatarDataBrasileira): Converte o formato de data ISO (YYYY-MM-DD) recebido do banco de dados para o padrão brasileiro (DD/MM/YYYY) para exibição amigável ao usuário.
Carregamento de Dados (carregarJogadores):

Faz uma requisição GET ao Controller (../app/controller/controllerCardsJogadores.php).

Limpa o contêiner de exibição (cardsJogadores) para evitar duplicação.

Em caso de sucesso, itera sobre o array de jogadores recebido, constrói a estrutura HTML (div.card-jogador) e insere as informações (Nome, Email, Posição, Telefone, etc.) no DOM.

Exibe mensagens específicas caso a lista de jogadores esteja vazia ou se ocorrer um erro durante a requisição (fetch).

- JS - **login.js**

O arquivo login.js contém a lógica client-side para gerenciar a tentativa de login do administrador. Ele captura as credenciais inseridas pelo usuário (e-mail e senha) e as envia de forma assíncrona para o backend para validação.

Função Principal (loginAdmin): É a única função responsável pelo processo de autenticação.

Captura de Credenciais: Obtém os valores dos campos de input com os IDs email e senha.

Preparação dos Dados: Cria um objeto FormData e anexa as variáveis email e senha para prepará-las para o envio via POST.

Comunicação com o Backend:

- Executa uma requisição fetch com o método POST para o endpoint: ../app/controller/controllerLogin.php.
- Envia as credenciais no corpo (body) da requisição usando o objeto formData.

Processamento da Resposta:

- Recebe a resposta do servidor e a converte para o formato JSON.
- Sucesso: Se a propriedade success da resposta for true, redireciona o usuário para a página administrativa (admin/admin.html) após um pequeno *delay* de 1 segundo.

Tratamento de Erro (catch): Em caso de falha na conexão ou na requisição, registra o erro no console e exibe uma mensagem de erro na interface.

- JS - **cadastro.js**

O arquivo cadastro.js é responsável por capturar todos os dados de um novo jogador a partir do formulário de cadastro, incluindo o arquivo de foto, e enviá-los de forma assíncrona para o backend para processamento e persistência.

Função Principal (cadastroJogadores): Gerencia o processo completo de envio do formulário.

Captura de Dados: Coleta os valores de todos os campos do formulário (nome, data de nascimento, CPF, email, telefone, posição, categoria, cidade, estado) usando seus respectivos IDs.

Captura de Arquivo: Obtém o arquivo de imagem selecionado pelo usuário no campo com ID foto usando .files[0].

Preparação do Envio:

Cria um objeto FormData, que é essencial para enviar dados de formulário que incluem arquivos (upload de foto).

Anexa todos os dados de texto e, se uma foto foi selecionada, anexa o objeto File da foto.

Comunicação com o Backend:

Executa uma requisição fetch com o método POST para o endpoint do Controller: ../app/controller/controllerCadastro.php.

O FormData é enviado como corpo (body) da requisição.

Processamento da Resposta:

Converte a resposta do servidor para JSON.

Sucesso: Se data.success for true, exibe um alerta de sucesso e reseta o formulário (formCadastro) para que o usuário possa cadastrar outro jogador.

Erro: Se data.success for false, exibe um alerta com a mensagem de erro fornecida pelo servidor.

Tratamento de Erro: Trata erros de requisição (como problemas de conexão) registrando-os no console e não afetando a interface do usuário.

- CONFIG - **banco_projeto_web.sql**

O arquivo banco-projeto_web(atualizado).sql é o script SQL completo que define a estrutura de tabelas e popula o banco de dados do projeto. Ele é essencial para configurar o ambiente de dados.

Definição de Estrutura (DDL): O script cria três tabelas principais: usuarios, jogadores, e peneiras.

Tabela usuarios: Armazena os dados de login, incluindo nome, email (chave única), senha (armazenada com hash), e tipo_usuario (admin ou avaliador).

Tabela jogadores: Contém o cadastro detalhado dos atletas, como nome, data_nascimento, cpf (chave única), email, telefone, posicao, categoria, cidade, estado, e o caminho da foto. Possui uma chave estrangeira (peneira_id) para relacionar o jogador a uma peneira.

Tabela peneiras: Registra as sessões de avaliação, incluindo nome, descricao, data_inicio, data_fim, local, vagas, e status (aberta ou encerrada).

Dados Iniciais (DML): O script já inclui dados de exemplo na tabela usuarios (para o administrador) e na tabela jogadores (para atletas como Messi, Neymar e Cristiano Ronaldo), permitindo testes imediatos no sistema.

Chaves e Restrições: O arquivo define chaves primárias (id), chaves únicas (cpf em jogadores e email em usuarios), e o relacionamento entre jogadores e peneiras através da chave estrangeira.

- MODEL - **modeledit.php**

Este arquivo contém a classe JogadorModel (compartilhada com outras funcionalidades de jogador), focada em buscar dados para validação e executar a atualização dos registros.

Método buscarPorCpfOuEmail($cpf, $email, $id = null):

  - Utilizado para verificar a existência de um jogador com o CPF ou Email fornecidos, o que é crucial antes de uma atualização para evitar duplicidade.
  - O parâmetro $id permite que a verificação ignore o próprio registro que está sendo editado, evitando falsos positivos de duplicidade.
  - Nota sobre o código: O código possui repetição na lógica de consulta e o LIMIT 1 final está fora da lógica de execução, mas sua intenção é validar a unicidade.

Método atualizarJogador($dados):

  - Executa um *prepared statement* de UPDATE na tabela jogadores.
  - Define todos os campos do jogador (nome, data_nascimento, cpf, email, etc., incluindo foto) com base no array $dados.
  - A atualização é restrita pelo WHERE id = :id, garantindo que apenas o registro correto seja modificado.
  - Retorna o resultado da execução da query (booleano).

- MODEL - **modelCardsJogadores.php**

Este arquivo define a classe CardsJogadoresModel, dedicada exclusivamente à funcionalidade de buscar todos os jogadores para exibição na página principal (em formato de cards).

Método buscarTodosJogadores():

- Executa uma simples consulta SQL (SELECT * FROM jogadores) para recuperar todos os registros da tabela jogadores.
- Ordena os resultados pelo id de forma decrescente (ORDER BY id DESC), mostrando os jogadores mais recentes primeiro.
- Retorna um array associativo de todos os jogadores encontrados, pronto para ser consumido e formatado pelo Controller e JavaScript.

- MODEL - **modelCadastro.php**

Este arquivo contém a classe JogadorModel, responsável por duas operações essenciais relacionadas à criação de novos registros de jogadores no banco de dados.

Método buscarPorCpfOuEmail($cpf, $email):

- Utilizado para validação pré-cadastro.
- Verifica se já existe um jogador com o mesmo CPF ou Email cadastrado no sistema antes de permitir a inserção de um novo registro.
- Retorna o registro encontrado (se houver) ou false.

Método cadastrarJogador($dados):

- Executa um *prepared statement* de INSERT na tabela jogadores.
- Recebe um array $dados contendo todos os campos necessários (nome, cpf, email, foto, etc.).
- Insere o novo registro no banco de dados.
- Retorna o resultado da execução da query (booleano, indicando sucesso ou falha na inserção).

- CONTROLLER - **controllerLogin.php**

O controllerLogin.php lida exclusivamente com requisições de login via método POST.

1. Configuração Inicial: Inicia a sessão (session_start()) e define o cabeçalho de resposta como application/json para comunicação via AJAX/API.
2. Inicialização de Componentes:
   - Carrega as dependências via Composer (require '../../vendor/autoload.php').
   - Carrega o *Service Manager* da aplicação através do *bootstrap* do Zend/Laminas.
   - Obtém a instância do EntityManager da Doctrine ORM e injeta-a no LoginService.
3. Lógica de Login:
   - Verifica se a requisição é do tipo POST.
   - Captura o email e a senha enviados via POST.
   - Chama o método autenticar() do LoginService com as credenciais fornecidas.
4. Processamento da Resposta:
   - Sucesso: Se a autenticação for bem-sucedida ($usuario é retornado), as informações essenciais do usuário (ID e email) são salvas na sessão PHP ($_SESSION), e um JSON de sucesso é retornado: {"success": true}.

    - Falha: Se a autenticação falhar, um JSON de falha com uma mensagem de erro é retornado: {"success": false, "message": "Email ou senha incorretos!"}.
    - Em ambos os casos, a execução do script é encerrada com exit.

- CONTROLLER **- controllerCardsJogadores.php**

Este controller é dedicado a fornecer a lista de todos os jogadores para a página principal (home), onde são exibidos em formato de cards. É o endpoint que o principal.js consulta ao carregar a página.

Interação com o Model: Instancia o CardsJogadoresModel e invoca o método buscarTodosJogadores().

Processamento de Dados: O model retorna a lista completa de jogadores. O controller não realiza processamento de dados adicional (como formatação de data), apenas envia o resultado.

Resposta: Retorna um objeto JSON com a chave "success": true e a lista de jogadores no array "data".

Tratamento de Erro: Implementa um bloco try-catch para capturar exceções do banco de dados e retornar uma mensagem de erro formatada em JSON.

- CONTROLLER - **controllerListarJogadores.php**

Este controller tem a função de fornecer dados resumidos dos jogadores, tipicamente para serem exibidos em uma tabela ou lista simplificada na área administrativa.

Consulta Direta: Ao invés de utilizar um Model, este controller executa a consulta SQL (SELECT id, nome, email, posicao, categoria...) diretamente utilizando o objeto PDO (conexão $pdo).

Dados Retornados: A consulta é otimizada para buscar apenas os campos essenciais para listagem.

Resposta: Retorna um objeto JSON com a chave "success": true e a lista de jogadores na chave "jogadores".

Segurança: Utiliza *prepared statement* ($pdo->prepare) para a consulta, garantindo a segurança mesmo em uma simples SELECT.

- CONTROLLER - **controllerCadastro.php**

Este é o controller responsável por receber, validar e persistir os dados do formulário de cadastro de novos jogadores, incluindo o upload de arquivos.

Coleta de Dados: Coleta todas as informações do jogador enviadas via POST ($_POST), como nome, CPF, email, etc.

Upload de Foto: Verifica se um arquivo de foto foi enviado ($_FILES['foto']), cria o diretório de uploads (../uploads/) se necessário, e move o arquivo temporário para o destino final. O caminho da foto é salvo para o banco de dados.

Validação de Unicidade: Utiliza o modelCadastro.php para checar se já existe um jogador com o mesmo CPF ou email antes de tentar inserir. Se a validação falhar, retorna um erro imediatamente.

Persistência de Dados: Se a validação for bem-sucedida, chama o método cadastrarJogador() do Model para realizar o INSERT no banco de dados.

Resposta: Retorna um objeto JSON indicando o sucesso ou a falha da operação de cadastro.

- CONTROLLER - **controllerEdit.php**

Este controller manipula a lógica para atualizar um registro de jogador existente, garantindo a continuidade da foto anterior ou o processamento de uma nova.

Coleta de Dados: Recebe todos os campos de um jogador via POST, incluindo o crucial id do registro a ser editado.

Gestão de Foto: Inicialmente, busca o jogador existente (embora o método buscarPorId não esteja visível no Model, é implicado), e mantém o caminho da foto anterior. Se um novo arquivo for enviado ($_FILES['foto']), processa o upload e atualiza o caminho da foto.

Validação: Utiliza o Model para verificar se o novo CPF ou email inseridos já pertencem a *outro* jogador (embora a lógica de exclusão do próprio ID no Model possa estar incompleta na versão fornecida, a intenção é evitar duplicidade).

Atualização: Chama o método atualizarJogador() do Model para executar a query UPDATE no banco de dados.

Resposta: Retorna um objeto JSON com o status da atualização.

- CONTROLLER - **controllerDelete.php**

Este controller é responsável por executar a remoção permanente de um registro de jogador do banco de dados, utilizando apenas o ID do registro.

Validação de Entrada: Verifica estritamente se o ID do jogador foi enviado via POST e se é um valor numérico válido.

Execução de Exclusão: Prepara e executa a query DELETE diretamente na conexão PDO ($pdo), sem a necessidade de um Model, usando o ID fornecido no WHERE com *preparação* para segurança.

Confirmação: Verifica o número de linhas afetadas ($stmt->rowCount()) para confirmar se a exclusão foi bem-sucedida.

Resposta: Retorna um objeto JSON com success: true se o jogador foi deletado ou success: false caso o ID seja inválido ou nenhum jogador correspondente tenha sido encontrado.

4.3 - FLUXO DO SISTEMAS

1. O jogador acessa o site e preenche o formulário de inscrição.
2. Os dados são enviados para o servidor e salvos no banco MySQL.
3. O administrador faz login no painel e visualiza todos os cadastros.

4.4 - MANUTENÇÃO E CONTRIBUIÇÃO

Este projeto é aberto à comunidade para manutenção, melhorias e novas funcionalidades. Encorajamos ativamente a colaboração externa por meio da plataforma GitHub.

Diretrizes de Contribuição

Para garantir a qualidade, a estabilidade e a integração eficiente de novas funcionalidades, todos os colaboradores devem seguir as diretrizes abaixo:

Fork e Clone: Crie um fork (cópia) do repositório principal para o seu perfil no GitHub e clone o *fork* para o seu ambiente local de desenvolvimento.

Criação de Branches: Trabalhe sempre em uma *branch* dedicada, nomeada de forma descritiva (ex: `feature/nova-autenticacao`, `fix/erro-login`).

Boas Práticas de Código:

- Padrões de Código: Recomenda-se estritamente seguir o padrão PSR-12 (Extended Coding Style Guide) para organização e padronização do código PHP. Isso inclui regras sobre indentação, declaração de classes, métodos e visibilidade.
- Comentários: Adicione comentários claros e concisos onde a lógica do código não for autoexplicativa.
- Testes: Se aplicável, inclua testes unitários ou de integração para as novas funcionalidades ou correções implementadas.

- Pull Requests (PRs):
  - Ao finalizar o trabalho, envie um Pull Request (PR) para a *branch* principal do repositório original.
  - O título do PR deve ser claro e o corpo da descrição deve detalhar o que foi modificado e por que a mudança é necessária (referenciando *issues* existentes, se houver).

- O PR passará por uma revisão de código (*Code Review*) antes de ser mesclado (*merge*).

Observação: Contribuições que não sigam os padrões de código especificados (PSR-12) ou que careçam de documentação e clareza podem ser solicitadas para revisão antes da aceitação.

4.5 - TESTES

Foram realizados testes manuais nas principais rotas do sistema, validando o fluxo de cadastro, edição e exclusão de jogadores.

---

5 - VERSÕES E DATAS

DIAGRAMA GANTT

Tarefa	Data de Conclusão	Prevista Versão
MVC (Model-View-Controller)	27/10	0.2
Protótipo da Página Inicial	30/10	0.3
Página Inicial + Página do Administrador	03/11	0.4
Cronograma	04/11	0.5
Login e Cadastro	10/11	0.6
Abas e Toques Finais na Página Inicial	11/11	0.7
Mais Abas e Funcionalidades	13/11	0.8
Funcionalidades para Lista	16/11	0.9
Sessão de Cards + Botão de Deletar	17/11	1.0
Zendframework + Doctrine	24/11	1.1

---

6 - REFERÊNCIAS

O projeto utilizou de referência o site : ProcurandoCraques Peneira de Futebol
utilizamos unicamente como inspiração e base para criarmos o nosso, onde iremos modificá-los no futuro para que seja ainda mais autêntico do que já é

Link : https://www.procurandocraques.com/

Utilizamos também o site : PSG Academy Brasil
onde é localizada essencialmente em Recife, tendo conhecidos que fizeram parte e trouxeram boas informações a respeito da qualidade, além de ser nosso concorrente pessoal, utilizamos ele como objetivo a ser alcançado.

Link : https://psgacademy.com.br/escolinha/escolinha-de-futebol-pe-recife/

7 - ANEXOS

Ferramentas, Materiais e canais utilizados na gestão do projeto

Canvas
Lightshot
Youtube
Serviços Google(doc,notas e etc)
Comet Browser,Brave Browser
Css Peeper
Awesome Screenshot
Visual Code
Xampp
Perplexity,Chat Gpt e Gemini
IloveMg
Discord