**CENTRO PAULA SOUZA**

**ETEC ZONA LESTE**

**Desenvolvimento de Sistemas**

**Alexandre Vieira dos Santos**

**Henrique Cesar Fonseca Alves**

**Kaua Alves Seppe**

**Luciano Rodrigues Campos Vitor**

**Rafael Moriya Oliveira**

**AUTOMATIZAÇÃO NA CRIAÇÃO E ATRIBUIÇÃO DE TREINOS PARA CLIENTES DE ACADEMIA**

**São Paulo**

**2022**

**Alexandre Vieira dos Santos**

**Henrique Cesar Fonseca Alves**

**Kaua Alves Seppe**

**Luciano Rodrigues Campos Vitor**

**Rafael Moriya Oliveira**

**AUTOMATIZAÇÃO NA CRIAÇÃO E ATRIBUIÇÃO DE TREINOS PARA CLIENTES DE ACADEMIA**

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Desenvolvimento de Sistemas da Etec Zona Leste, orientado pelo Prof. Rogerio, como requisito parcial para obtenção do título de técnico em Desenvolvimento de Sistemas.

**São Paulo**

**2022**

**DEDICATÓRIA**

Não há exemplo maior de dedicação do que o da nossa família. À nossa querida família, que tanto admiramos e amamos, dedicamos o resultado deste trabalho tão árduo realizado ao longo deste percurso. Dedicamos, este trabalho aos nossos docentes que sempre estiveram ao nosso lado disponibilizando-nos tempo e auxílio, dedicamos, também aos amigos e familiares.

**AGRADECIMENTO**

Agradecemos a todos que nos auxiliaram no desenvolvimento deste trabalho de conclusão de curso, onde fez-se imperiosa a ajuda, a compreensão e a dedicação de todos os participantes. Durante o desenvolvimento tivemos momentos de crises e dificuldades. Todavia, conseguimos superar os entraves com êxito e afinco. Ao que concerne aos mestres, transpareço nosso respeito, afeto e admiração. Por todo apoio e ensinamento, somos muito gratos. Aos nossos amigos somos gratos e exultamos pela paciência e confiança. Por derradeiro e não menos importante agradecemos as nossas famílias, por todo amor, apoio, carinho, benevolências e bem-aventurança.

“Nenhum cidadão tem o direito de ser um amador em matéria de treinamento físico. Que desgraça é para o homem envelhecer sem nunca ver a beleza e a força do que o seu corpo é capaz”

SÓCRATES

**RESUMO**

Em primeira análise, neste trabalho buscamos criar uma aplicação *web* para substituir o sistema de criação e atribuição de treinos em uma academia da Zona Leste de São Paulo, a fim de facilitar o contato entre professores e alunos, amenizar as dúvidas recorrentes aos exercícios com vídeos de explicativos e de demonstração de maneira intuitiva e dinâmica. O instrutor poderá criar treinos e atribui-los aos alunos, que poderão visualizá-los e verificar com detalhes os exercícios atribuídos.

**PALAVRAS-CHAVE:** *Web*. Treino. Comunicação. Exercício.

**ABSTRACT**

**SUMÁRIO**

[**1 INTRODUÇÃO 10**](#_Toc116528169)

[**2 REFERENCIAL TEÓRICO 11**](#_Toc116528170)

[**2.1 HTML 11**](#_Toc116528171)

[**2.2 CSS 14**](#_Toc116528172)

[**2.3 BOOTSTRAP 15**](#_Toc116528173)

[**2.4 JavaScript 16**](#_Toc116528174)

[**2.5 PHP 17**](#_Toc116528175)

[**2.6 LARAVEL 20**](#_Toc116528176)

[**2 .6 BANCO DE DADOS 32**](#_Toc116528177)

[**2.7.1 Abordagem Relacional 33**](#_Toc116528178)

[**2.7.2 Normalização 33**](#_Toc116528179)

[**2.7.3 Dicionário de Dados 33**](#_Toc116528180)

[**2.8 UML 33**](#_Toc116528181)

[**2.8.1 Levantamento de Requisitos 33**](#_Toc116528182)

[**2.8.2 Diagrama de Casos de Uso 33**](#_Toc116528183)

[**2.8.3 Diagrama de Atividades 33**](#_Toc116528184)

[**2.8.4 Diagrama de Classes 33**](#_Toc116528185)

[**2.8.5 Diagrama de Sequência 33**](#_Toc116528186)

[**3 DESENVOLVIMENTO 33**](#_Toc116528187)

[**3.1 Diagrama de Casos de Uso 33**](#_Toc116528188)

[**3.2 Diagrama de Atividades 33**](#_Toc116528189)

[**3,3 Diagramas de Classes 33**](#_Toc116528190)

[**3.4 Diagrama de Sequência 33**](#_Toc116528191)

[**3.5 DER 33**](#_Toc116528192)

[**3.6 Aplicação 33**](#_Toc116528193)

[**4 CONCLUSÃO 33**](#_Toc116528194)

# 1 INTRODUÇÃO

Em 11 de março de 2020, a Organização Mundial da Saúde (OMS) declarou que enfrentava uma nova pandemia OMS (2020), prejudicando assim, a forma já precária e desatualizada na atribuição de treinos e cadastros de alunos e funcionários das academias, tendo em vista que o papel ainda é utilizado como principal meio de gerenciamento de informações.

Visando substituir o meio subdesenvolvido de criação e atribuição de treinos em academias, por intermédio de uma aplicação web que tem a finalidade de facilitar a criação de treino por parte de um instrutor, e atribui-lo à um aluno, substituindo o uso do papel para o mesmo, melhorando a comunicação entre aluno e instrutor.

A interação entre aluno e professor foi afetada pelo período pandêmico, visto que nesse período todas as academias foram fechadas, o que prejudicou principalmente academias de pequeno porte, que por sua vez afetou diretamente a economia do país, afetando tanto grandes empresas quanto pequenas, sendo as academias um dos setores mais atingidos pela pandemia. Ademais, vale-se ressaltar que o atrasado tecnológico nas academias é o principal agente da perda de clientes. Isso porque, há uma ausência de sites e aplicativos que auxiliam no desenvolvimento dos treinos. Para JP Leitão (1993), não tem um “contrato” especifico inerente ao modo de como é gerenciada uma academia, o que pode levar a prejudicar e dificultar, no próprio gerenciamento do perfil dos clientes e dos docentes, na comunicação entre aluno e professor, ocasionando a insatisfação de ambas as partes.

Dessa maneira, fez-se imperiosa a realização do estudo de tecnologias como *Unifed Modeling Language* (UML) para a projeção e visualização prévia das funcionalidades da aplicação. *HyperText Markup**Language* (HTML), *Cascading Style**Sheet* (CSS) e JavaScript com o intuito de estruturar a parte visual da aplicação. Para o manejo de dados, o *Hypertext Preprocessor* (PHP) e seu *framework* Laravel foram os meios escolhidos para o desenvolvimento das funcionalidades do sistema juntamente com a conexão ao banco de dados, as tecnologias estudadas foram utilizadas com o fito de amenizar e solucionar as dificuldades apresentadas, também foram feitas pesquisas de campo afim de coletar o repertório de mudanças a serem feitas na situação atual das academias.

# 2 REFERENCIAL TEÓRICO

Neste trabalho de conclusão de curso foram utilizadas as seguintes tecnologias para mitigar os problemas de atrasos tecnológicos nas academias visando principalmente as academias locais.

# 2.1 HTML

O HTML (Linguagem de Marcação de Hipertexto), originou-se em 1991, pelo então considerado pai da internet Tim Berners-Lee, na Suíça. De acordo com o autor CJ Costa (2007). A linguagem HTML é construída por textos e códigos especiais chamadas marcas ou *tags* (comandos de linguagem), o HTML foi projetado inicialmente para interligar situações de pesquisas próximas e compartilhar arquivos com facilidade. Atualmente é a linguagem de marcação mais utilizada para formatar páginas web, sites e aplicativos mobiles em conjunto com linguagens de programação como o PHP (*Hypertext Preprocessor*) e o JavaScript. As páginas web possuem um sistema eficaz de edição e modelagem, necessitando de uma IDE de modelagem como Visual Studio Code, Not Pad++, etc.

Para criar qualquer projeto em HTML é necessário seguir algumas regras, tais regras são as *tags*. As *tags* são responsáveis pela criação de objetos identificáveis como: listas, links, parágrafos, textos e formulários.

Para Flatschart (2011), um hipertexto do HTML é um documento ou sistema formado por grupos de dados como (textos, imagens, vídeo, áudio) conectados por links. São exemplos de elos de associação a *tag* <href> utilizada para conectar uma página a outra, a *tag* imagem que salva uma imagem à página web. Estes recursos estão presentes em todos os sites e em toda parte visual de sistemas e até mesmo televisores e celulares.

Flatschart (2011) afirma que a estrutura básica de um documento HTML sofreu poucas alterações ao passar de suas atualizações, sendo o *Doctype* um dos elementos a serem destacados. O *Doctype* é responsável por indicar para o navegador quais os critérios que ele utilizará para processar o documento. O HTML 5 foi responsável por simplificar o uso deste elemento, a figura 1 apresenta como era declarado *Doctype* no HTML 4.

Figura - *Doctype* no HTML 4



Fonte: Caldeira, (2015).

Com a chegada do HTML 5, esse processo agora se resume a <DOCTYPE html>.

Para se criar uma página *Web* com o HTML, requer elementos básicos e estritamente necessários para programar nesta linguagem. A figura 2 retrata o escopo simples de um documento HTML.

Figura -Estrutura Básica HTML, Código HTML.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

De acordo com Caldeira (2015), esses elementos têm as seguintes funcionalidades dentro do código:

* *Html*: responsável por definir o início e o fim do programa.
* *Head*: é o cabeçalho do programa, normalmente não aparece na janela *web*.
* *Title*: elemento aninhado ao *Head*, escreve o título da página na barra de título acima na janela do navegador.
* *Body*: contém o conteúdo principal da página *web*, possuindo todo o texto e todas as imagens que constituem a página.
* *<!-- --!>*: tem como função inserir comentários sobre o programa e não são aparentes na janela do navegador.

Com o HTML 5 é possível realizar diversas atividades, dentre elas está a criação de tabelas. As tabelas em HTML são estruturas de divisão dos elementos que fazem parte de uma página *web*, operando de forma similar a estruturas semelhantes que existem nos processadores de texto. As tabelas não só servem para escrever texto em colunas, mas ainda, e essencialmente, para modificar o *layout* padrão de uma página *web* (Caldeira, 2015).

O trecho de código HTML apresentado na figura 3 demonstra a criação de uma tabela básica com código HTML. O resultado produzido é apresentado na figura 4.

Figura - Código Básico de uma Tabela, Código HTML

Texto

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Figura - Tabela HTML, Página *Web*

Tabela

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Segundo Caldeiras (2015), o atribudo *border=1* utilizado na *tag* <*table*> é responsável pela bordadura em torno da tabela com uma espessura definida em 1 pixel. O mesmo descreve os outros elementos utilizados:

* *Th*: define o cabeçalho de uma coluna da tabela.
* *Tr*: encarregado de definir cada linha da tabela.
* *Td*: define as células que compõem as linhas. Podem conter qualquer outra *tag* HTML.

# 2.2 CSS

*Cascading Style Sheets* (CSS) em português, folha de estilo em cascata. Surgiu no final de 1996 para revolucionar a web com seu o designer, formatação e leveza, visto que esse era o trabalho do HTML de marcar e modelar, toda vez que era preciso utilizar um determinado tipo de estilo era necessário utilizar *tags* e mais *tags* em HTML, tornando impossível a criação de *layouts* eficientes.

Não demorou muito para as pessoas perceberem que essa abordagem não funcionaria no futuro porque ela era fundamentalmente limitante. Em vez de tentar disponibilizar um documento monolítico aos navegadores web, fazia muito mais sentido dar aos navegadores os blocos de construção do conteúdo em si e, então, deixar que o navegador cuidasse de juntar tudo. Esse princípio é conhecido como separação de interesses (*separation of concerns*). (JR Lewis, 2010, p.18)

Para JR Lewis (2010), a inovação feita com o surgimento do CSS melhorou e facilitou a forma de trabalhar na web.

O CSS tem uma grande compatibilidade com navegadores web, também possuí uma manutenção de projetos rápida e fácil, apenas uma mudança no projeto pode transformar o escopo inteiro dando outra cara ao projeto.

O CSS está presente em diversos *frameworks* como *bootstrap*, [bulma](https://www.brasilcode.com.br/frameworks-css/#2-_Bulma)[, *materialize*](https://www.brasilcode.com.br/frameworks-css/#3-_Materialize)[, *foundation*](https://www.brasilcode.com.br/frameworks-css/#4-_Foundation), [*Semantic* UI](https://www.brasilcode.com.br/frameworks-css/#6-_Semantic_UI), esses *frameworks* são utilizados na criação de sites, ajudam na velocidade e leveza no desenvolvimento do projeto.

Para se ter uma modelagem faz-se mister ter todo o escopo do projeto CSS dentro do elemento <*style*>, as funcionalidades do CSS são definidas em *tags*, classes, containers, identificadores e atributos. O CSS puxa a *tag* utilizada no HTML, como a *tag* <*div*>, e modela de acordo com as particularidades das divisões.

Segundo Jobstraibizer (2009), os documentos CSS são compostos por uma programação básica:

Figura - Programação Básica CSS

Texto

Descrição gerada automaticamente com confiança média

Fonte: Jobstraibizer, 2009.

Os blocos identificados pelo ponto (.) são atributos de classes. Podendo ser manipulados quantas vezes forem necessárias dentro de uma página, sendo utilizados com frequência. Já os blocos identificados pela cerquilha (#) são atributos de ID. São identificadores de um conteúdo específico, preferencialmente utilizados apenas em locais específicos, como, por exemplo, em formatação de blocos de conteúdo HTML (Jobstraibizer, 2009).

Seguindo a programação básica do CSS, a figura X apresenta o resultado da estilização da tabela HTML, figura 4. E a figura 7 mostra o código CSS utilizado, demonstrando como seus valores são aplicados.

Figura - Tabela Estilizada, Código HTML

Tabela

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Figura - Código CSS

Texto

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Jobstraibizer (2009) define as propriedades utilizadas no código:

* *Background*: inicia informações sobre o plano de fundo de determinado elemento. Neste caso, o elemento afetado foi o <*body*>, tendo sua a cor de plano de fundo alterada através do *background-color*.
* *Height*: informa a altura de determinado elemento.
* *Margin*: informação geral de margens de um elemento. Pode-se informar o valor da margem de uma direção específica, seja ela, inferior, esquerda, direita ou superior. Para definir um lado específico, basta especificar a direção, como o atributo *margin-top* utilizado no código para definir a margem superior da classe.
* *Width*: informa o tamanho geral de um elemento.
* *Font*: *tag* geral para informações acerca de fontes em um elemento. Possui diversos valores, como, por exemplo, *font-size* utilizado para definir o tamanho de uma fonte específica.
* *Color*: responsável por definir a cor do texto de um elemento.

# 2.3 BOOTSTRAP

O *bootstrap* é um *framework* front-end desenvolvido em 2011 para a produção de sites responsivos. Criado por Mark Otto e Jacob Thornton como um recurso interno do Twitter para resolver os problemas relacionados à códigos dentro de sua equipe de desenvolvimento. Por não haver um padrão de estrutura de código, cada engenheiro tinha sua maneira de programar, o que dificultava a junção das partes do código do projeto posteriormente. Segundo Silva (2014) A finalidade original do Bootstrap era motivar as equipes de desenvolvimento a utilizar uma única estrutura de código, nomenclatura de classes e etc, com o objetivo de reduzir inconsistências de código e otimizar o processo de desenvolvimento.

Levou poucos meses para milhares de desenvolvedores se interessarem com o projeto após o mesmo ser lançado publicamente no Github como um projeto de software-livre. Grande parte desses desenvolvedores contribuíram para o código, acelerando seu desenvolvimento e o tornando mais ativo.

Conforme Silva (2014), o Bootstrap proporciona a possibilidade de escolher quais elementos serão utilizados no projeto, dentre arquivos CSS, componentes do *framework* (botões, ícones, *sliders* e etc.), componentes de Javascript e entre outros.

Um *framework* *front-end* é um conjunto de elementos que ajudam a criar e desenvolver interfaces para a web. Como as estruturas de desenvolvimento de software mais confiáveis, as estruturas de desenvolvimento da *Web* possuem bibliotecas projetadas para simplificar a codificação CSS e aumentar sua compatibilidade com os padrões de estilo, bem como bibliotecas para desenvolvimento que tem modelos semiprontos e prontos. Diminuindo o tempo trabalhado nos projetos. Hoje o *bootstrap* não é apenas um framework com design responsivo eficaz, mas oferece todos os tipos de opções de funcionalidade e estilo (Techio; Chicon, 2016).

Para conseguir

Para exemplificar a funcionalidade do Bootstrap, a figura 8 representa o código da mesma tabela apresentada na figura 4, porém, com as classes de estilização predefinidas pelo *framework*.

Figura – Código da Tabela com Bootstrap, Código HTML

Texto

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Figura - Tabela Estilizada com Bootstrap, Página Web

Tabela

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Através das classes e elementos predefinidos pelo Bootstrap, desenvolvimento e estilização se torna mais prático e dinâmico, além de padronizar a estrutura do código.

# 2.4 JavaScript

O *JavaScript* (frequentemente abreviado como JS) é uma linguagem de programação voltada para o desenvolvimento *web*, criada por Brendan Eich tendo como base a linguagem Java em 1995 para a empresa Netscape. O JS foi criado com o fito de fornecer interatividade as páginas *web*. Com o decorrer dos anos após sua criação, de acordo com Flanagan (2004), a grande maioria dos navegadores e *sites* *web* comportam interpretadores *JavaScript* tornando-a a linguagem de programação mais utilizada da história. O *JavaScript* está presente na tríade de tecnologias *web* que todo desenvolvedor *front-end* deve conhecer sendo eles o HTML, CSS e o *JavaScript*.

O JS é uma linguagem onipresente dentro de navegadores e é considerada uma linguagem multiparadigma, com recursos para orientação a objetos e com tipagem fraca. Com o fato de a linguagem ser dinâmica, ou seja, não se faz necessária a especificação das variáveis antes da compilação do código, fazendo muitos programadores não se darem bem com a linguagem. Pensando em resolver essa problemática a empresa Microsoft criou em 2012 o sistema *typescript***,** dando autoria ao usuário de especificar a tipagem das variáveis, tendo mais controle individual dos objetos de sua biblioteca. Segundo Silva (2010)**,** a linguagem possui sua estrutura em linhas e blocos de códigos, sendo eles chamadas de *tags*. Nesse sentido, as *tags* são os comandos que compõem o corpo de um projeto JS.

O *JavaScript* é uma linguagem de alto nível. Segundo Flanagan (2004) uma linguagem de alto nível é interpretada, dinâmica e não tipada conveniente para programação orientada a objetos. Ou seja, está mais próxima à linguagem do programador a do computador o que a torna uma linguagem mais intuitiva e fácil de se trabalhar. O JS é muito utilizado para adicionar itens complexos a páginas web, adiciona também mapas interativos, gráficos 2D/3D animados, alertas e funções deixando o sistema fluido e leve.

Segundo Grillo (2008):

JavaScript permite criar pequenos programas embutidos no próprio código de uma página HTML e capazes de gerar números, processar alguns dados, verificar formulários, alterar valor de elementos HTML e criar elementos HTML. Tudo isso diretamente no computador). (Grillo, 2008, pg.4)

Oque torna o JS uma linguagem indispensável para a programação web.

# 2.5 PHP

O *Hypertext Preprocessor* (PHP), antes chamado de *Personal Home Page Tools,* é uma linguagem de programação originada no outono de 1994 por Rasmus Lerdorf. Segundo Dall’Oglio (2015), no início a linguagem era formada por conjuntos de *scripts* escritos em linguagem C com foco na criação de páginas dinâmicas, utilizada por Rasmus para acompanhar o acesso ao seu currículo na internet. Após um tempo, a linguagem passou a se tornar conhecida e atrair novos usuários. Motivado por isso, Rasmus adicionou diversos mecanismos, sendo a interação com bancos de dados um dos mais importantes. No ano de 1995, foi liberado o código-fonte do PHP, possibilitando que outros desenvolvedores se juntassem ao projeto para aprimorar ainda mais a linguagem. Nesse período, o PHP foi chamado de *Forms Interpreter* (FI)*,* sendo considerada a segunda versão do PHP, onde possibilitava analisar sintaticamente consultas de SQL (CONVERSE; PARK, 2003).

No mesmo ano, Andi Gutmans e Zeev Suraski, dois estudantes israelenses que faziam uso dessa linguagem em um projeto acadêmico de comércio eletrônico, cooperaram junto a Rasmus para aprimorar o PHP, reescrevendo todo o código-fonte com base no PHP/FI 2, resultando assim, na criação do PHP 3, disponibilizado oficialmente em julho de 1998 (DALL’OGLIO, 2015). O PHP 3 tinha como novidade extensibilidade, a possibilidade de conexão com diversos bancos de dados, novos protocolos, uma sintaxe mais consistente, suporte à orientação a objetos e uma nova API (*Application Programming Interface*), possibilitando a criação de novos módulos, o que ocasionou no interesse de diversos desenvolvedores em relação ao PHP. Lançado em 2000 e desenvolvido por Zeev e Andi, o PHP 4 tinha como objetivo melhorar seu desempenho e sua modularidade em aplicações complexas. Conforme Dall’Oglio (2015), esta versão foi responsável por trazer melhorias como seções, suporte a diversos servidores web, além da abstração de sua API, permitindo ser utilizado como linguagem para *shell* *script*. Hoje estando na versão 8, o PHP se consolidou como uma das mais utilizadas linguagens do lado servidor.

O PHP, assim como o JS, é uma linguagem comumente utilizada abordando o paradigma da Programação Orientada a Objetos (POO), que representa uma filosofia para criação de sistemas (DALL’OGLIO, 2015). De acordo com Mendes (2009), o paradigma da orientação a objetos teve seu início em meados da década de 70, mas somente após o sucesso da linguagem de programação Java é que o paradigma ganhou credibilidade. Este paradigma traz uma diferente perspectiva da programação estruturada, na questão de adotar meios mais próximos do mecanismo humano para gerenciar a complexidade de um sistema. Dall’Oglio (2015) afirma que as linguagens estruturadas como Cobol, Clipper e Pascal criavam sistemas formados por um conjunto de procedimentos e variáveis nem sempre agrupadas de acordo com o contexto, já na orientação a objetos, é utilizada uma ótica mais semelhante ao mundo real, lidando com objetos que retratam estruturas que carregam dados e comportamento próprio que conversam entre si com o objetivo de formar algo maior, um sistema.

Um programa PHP possui regras de escrita assim como outras linguagem, para se iniciar um código PHP, deve-se utilizar *<?php* no início do código e *?>* ao encerrá-lo. A Figura 1 demonstra a estrutura padrão de um código PHP.

**Figura 10 - Estrutura Código-Fonte PHP**

Interface gráfica do usuário

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

De acordo com Converse e Park (2003), PHP se tornou uma linguagem para a criação de scripts para a *Web* sendo utilizada juntamente ao HTML, possuindo compatibilidade com diversos servidores *Web* importantes (principalmente o Apache). Uma das funcionalidades do PHP é a possibilidade de incorporar fragmentos de códigos em páginas padrão HTML, sendo interpretado à medida que as páginas são apresentadas aos usuários.

A figura 2 apresenta um código PHP embutido em um código HTML

**Figura 11 - Código PHP embutido em HTML, Código HTML**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente Fonte: Autoria própria, 2022.

CONVERSE e PARK (2003) retratam o PHP como uma linguagem que facilita a conexão das páginas *Web* com o banco de dados.

Para realizar a conexão com o banco de dados, deve-se atribuir o nome do host, nome do banco de dados, usuário do banco de dados, e a senha de acesso ao banco de dados a variáveis. A figura 3 mostra o código utilizado para realizar a conexão.

**Figura 12 - Conectando ao Banco de Dados, Código PHP**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

O PHP é um módulo oficial de servidor HTTP Apache, um dos maiores servidores *Web* do mercado. A empresa W3Techs (2022), responsável por monitorar o uso de tecnologias na Web, realizou um levantamento de dados, que obteve como resultado a linguagem de programação PHP sendo utilizada por cerca de 77,4% dos sites disponíveis na internet. Devido o PHP ser uma linguagem de código livre e extremamente popular, diversos mecanismos de facilitação de manuseio do código foram criados, alguns deles chamados de *frameworks*, sendo Laravel o mais conhecido e utilizado.

# 2.6 LARAVEL

Lançado em 2011, o *framework* Laravel é apontado por Neto (2020) como versátil e robusto. A equipe desenvolvedora fez uso de múltiplas ferramentas e soluções já consolidadas e utilizadas na comunidade PHP, juntando-as dentro de uma estrutura que aproveita da melhor forma os artifícios fornecidos por essas ferramentas. Considerado um *framework* volátil, sendo útil tanto para projetos simples quanto de grande porte. Um de seus atributos mais característicos é a praticidade que ele oferece para o desenvolvimento de funcionalidades comuns, através da ferramenta *artisan*, executada por meio de linhas de comando.

A obra Laravel para ninjas (GABARDO, 2017) retrata o Laravel como um *framework* intuitivo, claro e com diversas funcionalidades. Sendo um *framework PHP Model View Control (*MVC*),* sob a norma de Programação Orientada a Objetos (PÓO).

Os *frameworks* possuem diversos propósitos, como diminuir ou deixar de escrever um código-fonte por meio da reutilização de métodos, classes e funções, forçar a utilização de um padrão de design de projetos, como por exemplo o MVC, que oferece diversos meios de uso comum no meio de desenvolvimento *Web* como roteamento, criação de formulários, sessões e outros artifícios já feitos ou semiprontos, evitando assim, a necessidade de reescrever tais funções em cada projeto.

Dentro desses artifícios disponíveis, o Laravel faz uso da *engine* do *Blade Template*, que traz diversas ferramentas que facilitam a escrita de um código HTML, junto com dados dinâmicos de forma limpa e sem duplicação.

Silva (2017) afirma que o *Blade Template* pode ser descrito como uma maneira diferente de escrever as *views*. O que normalmente era feito através de *tags* PHP diretamente no HTML, o *blade* faz uso de uma sintaxe com chaves e arrobas, que por sua vez, traz uma clareza para os códigos, deixando os arquivos das *views* mais limpos e de fácil leitura. Uma *view* em formato *blade* é nomeado como <nome>.*blade*.php. O Laravel, por padrão, reconhecerá que a *view* possui a estrutura *blade*, convertendo os códigos *blade* em código PHP.

A figura 4 e 5 fazem a comparação de um código padrão *blade* e PHP.

Figura - Exemplo de View com PHP

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

Fonte: Silva (2017).

Para simplificar ainda mais, o *Blade* não faz necessário o uso de códigos PHP na view, apenas utiliza uma *tag* específica do próprio para imprimir o valor de uma variável: {{ $exemplo->titulo }}.

Figura - Exempo de *View* com *Blade*

Texto, Carta

Descrição gerada automaticamente

Fonte: Silva (2017).

Nos tempos atuais frameworks não são novidade no meio de desenvolvimento de softwares. Grande parte das linguagens possuem frameworks dos mais variados tipos. De acordo com McCool (2012), um framework assimila um conjunto de classes ou funções implantadas em uma linguagem de programação específica, facilitando o processo de desenvolver um software. Um framework, então, é um conceito, uma estrutura que serve como início para o desenvolvimento de um projeto específico.

Ao criar um projeto com o Laravel, os seguintes arquivos e pastas mostrados na figura 6 fazem parte da pasta raiz do projeto.

**Figura 15 - Pastas Padrões de um Projeto Laravel 9**

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Cada pasta e arquivo ´possui sua funcionalidade para promover e facilitar a criação da aplicação em questão. Esses arquivos e pastas tem os seguintes papéis e funcionalidades no projeto apontadas por Gabardo (2017):

• *app*: é justamente a pasta da aplicação. Nesta pasta se encontra os models e a subpasta Http/*Controllers* onde serão os controladores serão construídos.

• *bootstrap*: pasta que possui *scripts* que carregam, inicializam a aplicação e retornam a solicitação da aplicação.

• *config*: pasta onde estão os arquivos de configuração, como as configurações de conexão com banco de dados e outros.

• *database*: esta pasta armazena classes específicas das interações com banco de dados sendo elas: *migrations*, *factories* e *seeds*.

• *public*: nesta pasta que a alocação de arquivos de acesso público como imagens, arquivos estáticos é realizada. Esta pasta possui o arquivo index.php, arquivo de configuração do servidor Apache.htaccess, e diversos outros arquivos de configuração

• *routes*: pasta onde as rotas serão definidas, seja para uma API, aplicação Web etc.

• *storage*: pasta destinada a manter arquivos gerados pelo framework, como logs, sessões, caches etc.

• *resources*: armazena as *views* do projeto, arquivos para compilação de CSS, arquivos de linguagem e outros recursos.

• *tests*: pasta com a finalidade de armazenar arquivos de teste unitários.

• *vendor*: possui arquivos do framework propriamente dito. Não havendo necessidade de alterar a pasta.

• .env: arquivo responsável por possuir as variáveis da aplicação incluindo a chave criada no momento da criação do projeto.

• .env-example: arquivo de exemplo para configuração das variáveis da aplicação.

• .gitattributes: arquivo que possui orientações para o servidor de versionamento.

• .gitignore: arquivo responsável por impedir outros arquivos de serem adicionados a repositórios GIT, como o arquivo detentor da chave da aplicação e a pasta do próprio framework.

• *artisan*: arquivo encarregado de carregar recursos do framework automaticamente.

• composer.json: responsável por determinar as dependências do projeto. Este arquivo possui as orientações que informam aos gerenciadores de pacotes quais são os pacotes e as bibliotecas das quais o projeto é dependente.

• composer.lock: arquivo criado automaticamente pelo gerenciador de dependências Composer. Após a instalação das dependências, o Composer guarda a lista de versões exatas dos pacotes instalados nesse arquivo, travando o projeto a essas versões específicas

• package.json: este arquivo possui as dependências para fazer uso do Gulp (ferramenta de automação de tarefas em JavaScript) com o Laravel.

• phpunit.xml: arquivo responsável pela configuração de testes unitários com a linguagem PHP.

• README.md: arquivos .md são arquivos de marcação que possibilita a utilização de *tags* e marcações comumente adotados para gerar documentação. Este arquivo possui informações sobre o *framework*, *links*, para a documentação, informações sobre a licença do *framework* etc.

Segundo Gabardo (2017), o MVC é um padrão de design de projetos que realiza a separação do HTML da lógica e das regras de negócio. Para o manuseio correto do *framework*, Gabardo (2017) afirma que é necessário compreender como o MVC funciona, pois é a base de um projeto Laravel.

A figura 7 representa uma ilustração do modelo MVC.

**Figura 16 – Fluxo de dados padrão do Modelo MVC**

Diagrama

Descrição gerada automaticamente

Fonte: Gabardo (2017).

Conforme apresentado na Figura 4, o fluxo de dados do programa tem início através de uma requisição HTTP. Uma rota mapeia e envia essa requisição para um *controller*, que é responsável pelo processamento da requisição, sendo capaz de aplicar e enviar dados a um *model*, para então apresentar uma *view,* e possuindo a capacidade de encaminhar dados para a *view*, ou apenas exibir um arquivo HTML estático. (GABARDO, 2017) aponta que o modelo MVC, possui as seguintes camadas:

* *Routes*: mapeiam as requisições HTTP e as enviam para o *controller* adequado. Apesar de não ser uma camada própria do modelo MVC, é o procedimento mais comum para abordar requisições HTTP. Certos *frameworks* realizam a liberação de acesso direto aos métodos dos *controllers*, já outros obrigam a formação de rotas para cada método.
* *Model*: são representações de objetos, tem como utilidade modelar os objetos que serão utilizados na aplicação. São responsáveis por definir quais são e de que tipo são os campos de um objeto.
* *Views*: arquivos da camada de exibição da aplicação, é o que apresenta os resultados das requisições para o usuário. Podendo ser HTML, PHP ou fazer uso de algum tipo de *template*.
* *Controllers*: responsáveis por coletar as requisições através das rotas, processam as requisições de acordo com a lógica de negócios empregada, consumindo e enviando dados para os *models*, também carregando arquivos de visualização.

As figuras 8, 9, 10, 11, 12, 13, e 14 exemplificam o processo de criação de um formulário de cadastro e listagem de endereços através de *routes, models*, *views*, *controllers* e *migrations,* fazendo uso de *bootstrap* para a estilização. As figuras 15 e 16 apresentam a página que o usuário visualiza

Figura - Cadastro de Endereço, Página Web.

**Uma imagem contendo Tabela

Descrição gerada automaticamente**

Fonte: Autoria própria, 2022.

A *view* de cadastro de endereço tem como função enviar os dados de cada campo do formulário inserido pelo usuário e armazená-los no banco de dados.

Figura - Cadastro de Endereço, Código HTML

*Linha do tempo

Descrição gerada automaticamente com confiança média*

Fonte: Autoria própria, 2022.

Figura - Listagem de Endereços Cadastrados, Página Web.

**Interface gráfica do usuário, Aplicativo, Word

Descrição gerada automaticamente**

Fonte: Autoria própria, 2022.

A figura 17 apresenta a página onde os dados enviados no formulário apresentado na figura 16 são apresentados.

Figura - Listagem de Endereços Cadastrados, Código BLADE

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

A diretiva @foreach própria do Laravel, possibilita a repetição do mesmo bloco de código enquanto houver endereços cadastrados. Nesse bloco de código, os campos enviados para o banco de dados na figura 20 são adicionados ao código e apresentados ao usuário.

Figura - Listagem de Endereços Cadastrados, Código BLADE (2)

Texto

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Figura - Controller do Formulário, Código PHP

**Texto

Descrição gerada automaticamente**

Fonte: Autoria própria, 2022.

Utilizando o *Controller*, os campos do formulário da figura 19 são atribuídos à variáveis, após isso, o trecho de código “endereço::*insert”* é responsável por enviar dados inseridos pelo usuário à tabela “endereco” dentro do banco de dados. A estrutura de condição verifica se os dados foram enviados, caso forem, exibe uma mensagem ao usuário dizendo que o local foi registrado, retornando a página principal com os dados recém registrados.

Figura - Controller Endereços Cadastrados, Código PHP

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Com o uso do termo “endereço::*all*()”, o *controller* permite o retorno da *view* de uma página junto dos dados armazenados no banco de dados.

Figura - Model dos Dados de Endereço, Código PHP

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

Fonte: Autoria própria, 2022.

Responsável por interagir com o front e back-end da aplicação, a *model* captura o esquema de tabelas criados na *migration*, o que permite a comunicação entre ambas as tabelas. Após a *model* possuir as mesmas tabelas que a *migration*, ela já se torna capaz de realizar a interação entre o lado cliente e lado servidor.

Figura - Migration dos Dados de Endereço, Código PHP

Texto

Descrição gerada automaticamente

Fonte: Autoria própria, 2022.

Criação da tabela que será responsável por armazenar os campos preenchidos pelo usuário ao informar um endereço no formulário. A variável $*table* informa a criação de uma coluna dentro da tabela endereço, após isso, é definido o tipo de dado que o campo utilizará, já entre parênteses, é definido nome da coluna e limite de caracteres.

# 2 .6 BANCO DE DADOS

O banco de dados foi criado em meados dos anos 80 baseado nos padrões SQL, pela empresa Oracle com o fito de armazenar e salvar informações e registros, sejam eles funcionais ou não funcionais. Para Christopher J, (2004) o banco de dados pode ser considerado um armário eletrônico de arquivamento, ou seja, é um sistema para armazenar informações também pode ser definido como

Uma coleção estruturada de dados. Os dados armazenados em um banco de dados são organizados de forma a permitir agilidade na busca e na recuperação por um computador, ou seja, não há nada além de uma simples coleção de itens. MANOVICH (2015, P. 8).

O banco de dados consegue fazer ligação com diversas linguagens de programação tornando-o mais prático. A estrutura do banco de dados consiste em tabelas relacionadas entre si, através de chaves primarias ou estrangeiras.

Existem dois tipos de bancos de dados, os relacionais e não relacionais. O relacional consiste em armazenar dados em tabelas. As tabelas são organizadas em colunas. E cada coluna armazena o tipo de dados inteiros, reais, varchar, palavras, caracteres, datas e assim por diante. Os dados de uma "instância" de uma tabela são armazenados em linhas, ao contrário do não relacional que não possui tabelas, linhas e colunas, ou seja, não possui artigos relacionáveis.

O banco de dados está presente em todas as aplicações e sistemas, pois é mais que necessário o armazenamento de dados, sejam eles pessoas ou dados do próprio sistema. O banco de dados necessita de uma boa criptografia para não ter vazamento de dados e estar dentro dos conformes da lei LGPD (lei geral de proteção de dados), que visa proteger a integridade do usuário ou cliente de uma empresa, assim não apresentando riscos a ambas as partes.

Para simplificar e facilitar na construção do banco de dados são utilizados como parâmetros e métodos o MER (modelo entidade-relacionamento) e o DER (diagrama entidade-relacionamento) criados pelo doutor Peter Shen em 1976. Para CINTRA (2012) o MER tem como base o conceito de que o tudo é criado por entidades e essas entidades se relacionam e essas entidades tem atributos. Com o MER é possível idealizar de maneira abstrata a estrutura do banco de dados que vai ser criado. O modelo entidade-relacionamento é composto por três tipos de objetos básicos sendo eles entidade, atributo e relacionamentos, ou seja, as entidades possuem atributos e se relacionam entre si por meio de relacionamentos.

(Exemplo mer)

Em contrapartida, para Bazzi (2018) – O DER que é um esboço menos trabalhado concernente ao banco de dados final, o DER também é a representação gráfica do modelo MER sendo representado por um diagrama, com o intuito de organizar as informações de relacionamento entre entidade, atributo e relacionamento de maneira simplificada e em ordem, são utilizados:

(Exemplo der)

* Retângulos - representam as entidades.
* Elipses - representam atributos.
* Losangos - representam relacionamentos.
* Linhas – ligam atributos a entidades e entidades a relacionamento.

# 2.7.1 Abordagem Relacional

Na abordagem relacional as informações presentes no MER e no diagrama DER, passam agora para a estrutura relacional do banco de dados, as entidades são representadas pelas tabelas e os atributos são representados nas linhas do escopo do banco de dados. De acordo com Alves (2014) faz-se mister que em todos os conjuntos de colunas interligadas os seus respectivos preceitos sejam únicos na tabela, assim não podendo substituir os valores das linhas das tabelas cridas.

Para Cardoso (2017). Os relacionamentos são classificados a partir do número de entidades presentes na conjuntura de relacionamentos, ou seja, também determina o tamanho da conjuntura desse grupo. As entidades podem possuir mais de um ligamento entre si, é importante ressaltar que os relacionamentos ligam as funcionalidades dos atributos as entidades e os atributos entre si.

Para ter-se uma boa abordagem relacional deve-se ter a separação e a colocação correta das chaves primarias e estrangeiras, comumente abreviadas por PK (*primary key* ou chave primaria) e FK (*fo rign key* ou chave estrangeira) e todas as propriedades que não necessitam (Araujo,2008). Ao definir uma chave primaria é definida também uma restrição de integridade, que é uma regra a ser obedecida em todos os bancos de dados, por outro lado para Alves (2014) as chaves estrangeiras caracterizam um relacionamento dentro do banco de dados relacional. Ao criar as chaves estrangeiras devem ter alguns cuidados principalmente quando incluir uma linha na tabela que contem chave estrangeira, ao alterar o valor da chave estrangeira e quando excluir uma linha que contém a chave primaria referenciada pela chave estrangeira.

Na abordagem relacional a integridade de domínio também é essencial para especificar as características dos atributos, especificando também o valor que um campo deve ter na coluna da tabela como domínios pré-definidos (número interiro (INT), data, tamanho, *string* ereal). Outra ferramenta muito importante é a interação de chave que define que o valor da chave primaria de ver único, algo paralelo acontece com a chave estrangeira na integridade referencial definindo a obrigatoriedade da chave estrangeira aparecer na chave primaria da tabela referenciada.

# 2.7.2 Normalização

# 2.7.3 Dicionário de Dados

# 2.8 UML

A *Unifed Modeling Language* ou linguagem de modelo unificada (UML) é uma ferramenta de modelagem e planejamento visual de projetos de *software*. Uma UML mostra diversos tipos de pré-modelos ou esboços de diagramas, dentre todos os diagramas, os mais utilizados para este serviço são os diagramas de classe e de sequência (Larman, 2000). Utilizando-se dos diagramas questão presentes na UML com intuito de ter a pré-visualização dos requisitos e escopo do projeto e auxilia também na visualização plena da comunicação dos objetos no sistema. Os diagramas dependem completamente da UML para ter suas funcionalidades ela é também constituída por quatorze diagramas sendo eles os mais utilizados os diagramas de classe, diagrama de caso de uso, diagrama de sequência e diagrama de atividade. Tais diagramas facilitam a comunicação dentro do ambiente de trabalho, comunicação entre cliente e funcionário, pois dissipa as dúvidas dos clientes através da idealização dos projetos visuais, auxilia a manutenção do projeto de forma fácil e dinâmica.

A UML tem uma ferramenta muito interessante chamada levantamento de requisitos, o levantamento de requisitos consiste em conjecturar os requisitos funcionais ou não funcionais necessários para a conclusão do trabalho e UML está presente na construção de todas as aplicações e projetos de sistemas de *software* o que à faz ser confundida por muitos como uma linguagem de programação.

# 2.8.1 Levantamento de Requisitos

O levantamento de requisitos é uma ferramenta pertencente a UML, sua principal função é que o usuário e o desenvolvedor tenham a mesma visão do problema a ser resolvido e levantar o que o cliente quer e precisa, ou seja, entender o cliente e estar a par das regras e processos de negócio, é necessário verificar a viabilidade da aplicação se é possível ou não fazer a aplicação e de que forma fazer o projeto, observar também o que cada componente vai realizar no sistema.

# 2.8.2 Diagrama de Casos de Uso

# 2.8.3 Diagrama de Atividades

# 2.8.4 Diagrama de Classes

# 2.8.5 Diagrama de Sequência

# 3 DESENVOLVIMENTO

# 3.1 Diagrama de Casos de Uso

# 3.2 Diagrama de Atividades

# 3,3 Diagramas de Classes

# 3.4 Diagrama de Sequência

# 3.5 DER

# 3.6 Aplicação

# 4 CONCLUSÃO

**REFERENCIAS**

ALVES, WILLIAM PEREIRA. **Banco de dados**. Saraiva Educação SA, 2014.

ARAÚJO, M. A. P. Modelagem de Dados–Teoria e Prática. **Revista Saber Digital**, v. 1, n. 01, p. 27-64, 2008.

MCCOOL, Shawn. **Laravel starter**. Packt Publishing, 2012.

BAZZI, Cláudio Leones. Introdução a Banco de Dados. 2018.

CALDEIRA, Carlos. **Introdução ao HTML**. 2015.

CARDOSO, GISELLE CRISTINA; CARDOSO, VIRGÍNIA M. **Sistema de banco de dados**. Saraiva Educação SA, 2017.

CINTRA, Glauber Ferreira. Banco de Dados. 2012.

CONVERSE, Tim; PARK, Joyce. **PHP: a bíblia**. Gulf Professional Publishing, 2003.

COSTA, Carlos J. Desenvolvimento para web . ITML press/Lusocredito, 2007.

DALL’OGLIO, Pablo. **PHP Programando com Orientação a Objetos** 3ª Edição. Novatec Editora, 2015.

DOUGLAS, Michael; MARABESI, Matheus. **Aprendendo Laravel: O framework PHP dos artesãos da web**. Novatec Editora, 2017.

FLANAGAN, David. **JavaScript: o guia definitivo**. Bookman Editora, 2004.

FLATSCHART, Fábio. **HTML 5-Embarque Imediato**. Brasport, 2011.

GABARDO, Ademir C. **Laravel para ninjas**. Novatec Editora, 2017.

GRILLO, Filipe Del Nero; FORTES, Renata Pontin de Mattos. **Aprendendo JavaScript**. São Carlos: USP, 2008.

JOBSTRAIBIZER, Flávia. **Criação de sites com o CSS**. Universo dos Livros Editora, 2009.

JR LEWIS, Joseph R.; MOSCOVITZ, Meitar. **Css avançado**. Tradução de Edgard B, p. 16, 2010

LARMAN, Craig. Utilizando UML e padrões. Bookman Editora, 2000.

TECHIO, Gabriel Bressan; CHICON, Patricia Mariotto Mozzaquatro. **Implementação dos frameworks bootstrap e Foundation aplicados na construção de um objeto de aprendizagem para o ensino da Engenharia de Software**. 2016. 44. Monografia (Curso Super de Tecnologia da Informação) - Universidade de Cruz Alta (unicruz), Rio Grande do Sul, 2016.

MENDES, Douglas Rocha. **Programação Java com ênfase em Orientação a Objetos.** Novatec Editora, 2009.

SILVA, Arthur de Almeida Pereira da. **DESIGN RESPONSIVO: TÉCNICAS, FRAMEWORKS E FERRAMENTAS**. 2014. 86 f. TCC (Graduação) -Curso de Sistemas de Informação, Centro de Ciências Exatas e Tecnologia, Universidade Federal do Estado do Rio de Janeiro (unirio), Rio de Janeiro, 2014.

SILVA, Daniel R. da. **Sistema para elaboração e controle da planilha de pontuação docente**: desenvolvido utilizando o Framework Laravel e PHP. Jataí-GO, 2017. 198f. Monografia (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas). IFG: Jataí-GO, 2017.

SILVA, Maurício Samy. **JavaScript-Guia do Programador: Guia completo das funcionalidades de linguagem JavaScript**. Novatec Editora, 2010.

NETO, Jaime. **Laravel - Escolhendo Um Framework Php**. Rio de Janeiro: Clube de Autores, 2020, 138 p.

QUIERELLI, Davi Antonio. **Criando Sites Com Html-css-php**. Clube de Autores, 2012.

**Usage statistics of PHP for websites**. W3Techs. [entre 2009 e 2022] Disponível em: <https://w3techs.com/technologies/details/pl-php>. Acesso em: 12 out. 2022.