CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

ESCOLA TÉCNICA ESTADUAL DA ZONA LESTE

Técnico em Desenvolvimento de Sistemas

Alexandre Vieira dos Santos

Henrique Cesar Fonseca Alves

Kauã Alves Seppe

Luciano Rodrigues Campos Vitor

Rafael Moriya Oliveira

**DEVFIT:** Criação e Atribuição de Treinos

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Desenvolvimento de Sistemas da Etec Zona Leste, orientado pelo Prof. Rogerio Bezerra Costa, 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.

**EPIGRÁFE**

“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**

Neste trabalho foi utilizada uma academia da Zona Leste de São Paulo como modelo para desenvolver uma aplicação *web*, tendo como objetivo substituir o uso de papel para sistema de criação e atribuição de treinos, a fim de facilitar o contato entre instrutores 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 clientes, que poderão visualizá-los e verificar com detalhes os exercícios atribuídos.

**PALAVRAS-CHAVE:** Web. Treino. Laravel. Exercício. Academia.

**ABSTRACT**

In this work, a gym from São Paulo’s east zone was utilized as a model to develop a web application, with the objective being to replace the use of paper for the gym’s management and creation and attribution of workout programs, to facilitate the connection between gym instructors and their students and appease their doubts in regard to their exercises with explanatory videos and dynamic and intuitive demonstrations. The instructor will be able to create workout programs and attribute them to different clients, who will be able to visualize them and see in detail their attributed exercises.

*KEYWORDS: Web. Training. Laravel. Exercise.*

**LISTA DE ILUSTRAÇÕES**

[Figura 1 - *Doctype* no HTML 4 15](#_Toc121670821)

[Figura 2 -Estrutura Básica HTML, Código HTML. 15](#_Toc121670822)

[Figura 3 – Formulário de Cadastro de Endereço, Código HTML 16](#_Toc121670823)

[Figura 4 – Formulário de Cadastro de Endereço, Página *Web* 16](#_Toc121670824)

[Figura 5 - Programação Básica CSS 18](#_Toc121670825)

[Figura 6 - Formulário Estilizado, Código HTML 18](#_Toc121670826)

[Figura 7 - Código CSS 19](#_Toc121670827)

[Figura 8 – Código da Tabela com Bootstrap, Código HTML 21](#_Toc121670828)

[Figura 9 - Código do Formulário com Bootstrap, Código HTML (2) 21](#_Toc121670829)

[Figura 10 - Formulário Estilizado com Bootstrap, Página Web 22](#_Toc121670830)

[Figura 11 - Validação do Formulário, Código JavaScript 23](#_Toc121670831)

[Figura 12 - Caixa de Alerta 23](#_Toc121670832)

[Figura 13 - Conectando ao Banco de Dados 26](#_Toc121670833)

[Figura 14 - Exemplo Conexão Banco de Dados, Código PHP 26](#_Toc121670834)

[Figura 15 - *Action* e *Method* do Formulário 27](#_Toc121670835)

[Figura 16 - Salvando Dados no Banco de Dados, Código PHP 27](#_Toc121670836)

[Figura 17 – Exibir Endereços Cadastrados, Código HTML 28](#_Toc121670837)

[Figura 18 - Exibir Endereços Cadastrados, Código HTML (2) 28](#_Toc121670838)

[Figura 19 - Exibir Endereços Cadastrados, Página Web 28](#_Toc121670839)

[Figura 20 - Exemplo de View com PHP 30](#_Toc121670840)

[Figura 21 - Exemplo de *View* com *Blade* 30](#_Toc121670841)

[Figura 22 - Pastas Padrões de um Projeto Laravel 9 30](#_Toc121670842)

[Figura 23 – Fluxo de dados padrão do Modelo MVC 32](#_Toc121670843)

[Figura 24 - Listagem de Endereços Cadastrados, Página Web. 33](#_Toc121670844)

[Figura 25 - Listagem de Endereços Cadastrados, Código BLADE 34](#_Toc121670845)

[Figura 26 - Listagem de Endereços Cadastrados, Código BLADE (2) 34](#_Toc121670846)

[Figura 27 - Controller do Formulário, Código PHP 35](#_Toc121670847)

[Figura 28 - Controller Endereços Cadastrados, Código PHP 35](#_Toc121670848)

[Figura 29 - Model dos Dados de Endereço, Código PHP 36](#_Toc121670849)

[Figura 30 - Migration dos Dados de Endereço, Código PHP 36](#_Toc121670850)

[Figura 31 - Exemplo DER 38](#_Toc121670851)

[Figura 32 - Exemplo Completo DER 39](#_Toc121670852)

[Figura 33 - Modelo Lógico Exemplo 40](#_Toc121670853)

[Figura 34 - Tabela Cliente Não Normalizada 41](#_Toc121670854)

[Figura 35 - Tabela Cliente e Telefone 1FN 41](#_Toc121670855)

[Figura 36 - Tabela Peças não Normalizada 42](#_Toc121670856)

[Figura 37 - Tabela Peça e Tabela Fornecedor 2FN 42](#_Toc121670857)

[Figura 38 - Tabela Venda não Normalizada 43](#_Toc121670858)

[Figura 39 - Tabela Venda e Vendedor 3FN 43](#_Toc121670859)

[Figura 40 - Exemplo Dicionário de Dados 44](#_Toc121670860)

[Figura 41 - Exemplo Diagrama de Caso de Uso 48](#_Toc121670861)

[Figura 42 - Exemplo Diagrama de Classes 49](#_Toc121670862)

[Figura 43 - Relacionamento Entre Classes 50](#_Toc121670863)

[Figura 44 - Exemplo Diagrama de Sequência 52](#_Toc121670864)

[Figura 45 - Exemplo Diagrama de Atividade 53](#_Toc121670865)

[Figura 46 - Requisitos Funcionais e Não Funcionais 54](#_Toc121670866)

[Figura 47 - Requisitos Não Funcionais 55](#_Toc121670867)

[Figura 48 - Diagrama de Casos de Uso 55](#_Toc121670868)

[Figura 49 - Diagrama de Classes 56](#_Toc121670869)

[Figura 50 - Diagrama de atividade: Cadastro 56](#_Toc121670870)

[Figura 51 - Diagrama de atividade: *Login* 57](#_Toc121670871)

[Figura 52 - Diagrama de atividade: Treino 57](#_Toc121670872)

[Figura 53 - Diagrama de Sequência: Cadastro de Usuário 58](#_Toc121670873)

[Figura 54 - Diagrama de Sequência: *Login* 58](#_Toc121670874)

[Figura 55 - Diagrama de Sequência: Treino Aluno 59](#_Toc121670875)

[Figura 56 - Diagrama de Sequência: Treino Instrutor 59](#_Toc121670876)

[Figura 57 - DER 60](#_Toc121670877)

[Figura 58 - Diagrama Entidade Relacionamento 60](#_Toc121670878)

**LISTA DE ABREVIATURAS E SIGLAS**

Organização Mundial da Saúde (OMS)

*Unifed Modeling Language* (UML)

SUMÁRIO

[1. INTRODUÇÃO 13](#_Toc121671455)

[2. REFERENCIAL TEÓRICO 14](#_Toc121671456)

[2.1 HTML 14](#_Toc121671457)

[2.2 CSS 17](#_Toc121671458)

[2.3 Bootstrap 20](#_Toc121671459)

[2.4 Javascript 22](#_Toc121671460)

[2.5 PHP 24](#_Toc121671461)

[2.6 Laravel 29](#_Toc121671462)

[2.7 Banco de Dados 37](#_Toc121671463)

[2.7.1 Abordagem Relacional 39](#_Toc121671464)

[2.7.2 Normalização 41](#_Toc121671465)

[2.7.3 Dicionário de Dados 44](#_Toc121671466)

[2.8 UML 44](#_Toc121671467)

[2.8.1 Levantamento de Requisitos 45](#_Toc121671468)

[2.8.2 Diagrama de Casos de Uso 47](#_Toc121671469)

[2.8.3 Diagrama de Classes 49](#_Toc121671470)

[2.8.4 Diagrama de Sequência 51](#_Toc121671471)

[2.8.5 Diagrama de Atividade 52](#_Toc121671472)

[3. DESENVOLVIMENTO 54](#_Toc121671473)

[3.1 Levantamento de Requisitos 54](#_Toc121671474)

[3.2 Diagrama de Casos de Uso 55](#_Toc121671475)

[3.3 Diagramas de Classes 56](#_Toc121671476)

[3.4 Diagrama de Atividade 56](#_Toc121671477)

[3.5 Diagrama de Sequência 58](#_Toc121671478)

[3.6 DER 60](#_Toc121671479)

[3.7 Modelo Lógico 60](#_Toc121671480)

[3.8 Aplicação 61](#_Toc121671481)

[4. CONCLUSÃO 62](#_Toc121671482)

[REFERÊNCIAS 63](#_Toc121671483)

# 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. Vale 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.

Visando substituir o meio subdesenvolvido de criação e atribuição de treinos nas 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.

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, sendo as academias um dos setores mais atingidos pela pandemia. Para Leitão (1993), não tem um padrão específico inerente ao modo de como é gerenciada uma academia, o que pode levar a prejudicar e dificultar a comunicação entre aluno e professor, ocasionando a insatisfação de ambas as partes.

Dessa maneira, fez-se necessário 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 a administração 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 objetivo de amenizar e solucionar as dificuldades apresentadas.

# REFERENCIAL TEÓRICO

No referencial teórico estão presentes as tecnologias e a descrição de suas abordagens e funcionalidades que dão embasamento a todo o trabalho.

## 2.1 HTML

O HTML originou-se em 1991 pelo então considerado pai da internet Tim Berners-Lee, na Suíça. De acordo com o autor Costa (2007), a linguagem HTML é construída por textos e códigos especiais chamadas marcas ou comandos de linguagem conhecidos como *tags*. O autor ressalta que 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 e o JavaScript.

Segundo Torres (2018), 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, vídeos e áudios conectados por links. São exemplos de dados 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.

O autor ressalta 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 1 - *Doctype* no HTML 4



Fonte: (FLATSCHART, 2015, p. 25)

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 2 -Estrutura Básica HTML, Código HTML.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: Do próprio autor, 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 formulários. Pedroso (2007) aborda os formulários como um dos recursos mais fascinantes do HTML, pois permitem o usuário interagir com o servidor enviando-lhe dados.

O trecho de código HTML apresentado na figura 3 demonstra a criação de um formulário. O resultado é apresentado na figura 4.

Figura 3 – Formulário de Cadastro de Endereço, Código HTML

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Figura 4 – Formulário de Cadastro de Endereço, Página *Web*

Uma imagem contendo Diagrama

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Pedroso (2007) descreve as *tags* utilizadas para a criação do formulário:

* *Form*: define um formulário, especifica o local do programa que controlará o formulário e define um método de como os dados serão enviados.
* H1: utilizada para alterar o tamanho da letra e aplicar o estilo negrito.
* *Label*: facilita a navegação rápida pelos campos de um formulário através de teclas de atalho. Ao ser pressionada juntamente com a tecla ALT, move o cursor do usuário diretamente para o campo especificado.
* *Input*: responsável por definir um campo de entrada de dados, onde o usuário digita as informações requeridas pelo formulário.
* *Small*: reduz a fonte do caractere.
* Br: realiza uma quebra de linha sem acrescentar espaços.
* *Button*: envia o conteúdo do formulário para o banco de dados ao ser pressionado.

## 2.2 CSS

*Cascading Style Sheets*, 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 o trabalho de marcar e modelar era do HTML, toda vez que era preciso utilizar um determinado tipo de estilo era necessário utilizar *tags* e mais *tags*, 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 Lewis e Moscovitz (2010), a inovação feita com o surgimento do CSS melhorou e facilitou a forma de trabalhar na web.

Os autores ressaltam que o CSS apresenta uma grande compatibilidade com navegadores *web*, também possuí uma facilidade para a remodelagem do projeto, apenas uma mudança no projeto pode transformar o escopo inteiro dando outra cara ao projeto. Para se ter uma modelagem faz-se necessário 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.

De acordo com Souza (2020), o CSS está presente em diversos *frameworks* como *bootstrap*, bulma[,](https://www.brasilcode.com.br/frameworks-css/" \l "3-_Materialize" \o "3- Materialize) *[materialize](https://www.brasilcode.com.br/frameworks-css/" \l "3-_Materialize" \o "3- Materialize)*, *foundation*, *Semantic* UI, esses *frameworks* são utilizados na criação de sites contribuindo na velocidade e leveza no desenvolvimento do projeto.

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

Figura 5 - Programação Básica CSS

Texto

Descrição gerada automaticamente com confiança média

Fonte: (Jobstraibizer, 2009, p. 7)

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 determinado conteúdo, preferencialmente utilizados apenas em locais específicos como, por exemplo, em formatação de blocos de conteúdo HTML (Jobstraibizer, 2009).

Figura 6 - Formulário Estilizado, Código HTML

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

Fonte: Do próprio autor, 2022.

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

Figura 7 - Código CSS

Texto

Descrição gerada automaticamente com confiança baixa

Fonte: Autoria própria, 2022.

Jobstraibizer (2009) e Pedroso (2007) definem as propriedades utilizadas no código:

* *Background*: indica as 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*.
* *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 os atributos *margin-top* e *left* utilizados no código para definir respectivamente a margem esquerda e superior da classe.
* *Width*: informa o tamanho geral de um elemento.
* *Padding*: é utilizada para definir um espaçamento interno entre sua borda, e seu respectivo conteúdo.
* *Border*: Define a borda 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 (BOOTSTRAP, [201-]). 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 entre outros, 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 predefinidos pelo *framework*, componentes de Javascript e outras opções de personalização.

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 assim, o tempo gasto 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 exemplificar a funcionalidade do Bootstrap, a figura 8 e 9 representam o código do mesmo formulário apresentado na figura 4, porém, com as classes de estilização predefinidas pelo *framework.*

Figura - Código da Tabela com Bootstrap, Código HTML

Linha do tempo

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Segundo Silva (2014), O bootstrap possui um módulo de responsividade por padrão para dispositivos *mobile*, que é ativada através da *meta tag* *viewport*.

Figura 9 - Código do Formulário com Bootstrap, Código HTML (2)

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Para utilizar as classes predefinidas do *bootstrap*, é necessário utilizar a *tag* *link*, para se comunicar com o CSS que o *framework* disponibiliza, assim, tornando as classes disponíveis para uso (Silva, 2014).

Figura 10 - Formulário Estilizado com Bootstrap, Página Web

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Os modelos prontos abordados por Techio e Chicon (2016) foram de grande utilidade para a criação do formulário, já que foram utilizadas as *tags* padrão para formulários em bootstrap.

## 2.4 Javascript

De acordo com Prescott (2016), o JavaScript 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 JS juntamente do HTML e CSS, estão presentes na tríade de tecnologias *web* que todo desenvolvedor *front-end* deve conhecer.

Stefanov (2011) ressalta que o Javascript é uma linguagem considerada 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. 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 e 3D animados, alertas e funções deixando o sistema fluido e leve.

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; Pontin, 2008, pg.4)

Silva (2010) afirma que o JS é uma linguagem que permite acessar os campos e valores digitados em um formulário, sendo capaz de realizar validações e apresentar mensagens para auxiliar o usuário. A figura 11 apresenta uma função JavaScript que realiza a validação dos campos.

Figura - Validação do Formulário, Código JavaScript

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Através da estrutura de condição *if*, a função captura o valor digitado pelo usuário através do elemento *value*, verificando se ele está vazio, caso esteja, ele emite uma mensagem ao usuário.

Figura - Caixa de Alerta

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

A caixa de alerta tem por finalidade apresentar uma caixa de diálogo com uma mensagem ao usuário (Silva, 2010), possibilitando uma maior comunicação entre a aplicação e o usuário.

## 2.5 PHP

O PHP, 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).

O autor afirma que 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. 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 *Application Programming Interface* (API), 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 tem como objetivo tonar a organização do código mais fácil, utilizando a orientação a objetos (CONVERSE; PARK, 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.

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.

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.

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 (NIEDERAUER, 2017), assim como apresenta a figura 13.

Figura 13 - Conectando ao Banco de Dados

Tabela

Descrição gerada automaticamente

Fonte: (Niederauer, 2017 p. 37)

Essa função retorna o objeto da conexão, que será utilizado para as funções de manipulação do banco de dados (NIEDERAUER, 2017). Uma boa prática no meio de programação é utilizar apenas caracteres simples para definir nomes chaves, evitando o uso de acentuações e facilitando a escrita do código. Utilizando boas práticas, para exemplificar uma conexão com um banco de dados local chamado formulário, que possui o nome de usuário padrão root e não possui uma senha, podemos executar o seguinte comando apresentado na figura abaixo.

Figura 14 - Exemplo Conexão Banco de Dados, Código PHP

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Após abrir a conexão, já é possível realizar operações sobre o banco de dados, possibilitando o envio de dados do formulário apresentado nas figuras anteriores para o banco de dados e a apresentação de dados ao usuário através de códigos embutidos em HTML. Para realizar o processo de envio de dados de um formulário, devemos definir qual programa receberá esses dados através da opção *action*, utilizada na *tag* *form* do HTML. Agora para definir como os dados serão passados, utilizamos a opção *method* também na *tag form* (NIEDERAUER, 2017). Exemplo:

Figura 15 - *Action* e *Method* do Formulário



Fonte: Do próprio autor, 2022.

As seguintes figuras exemplificam o processo de envio, armazenamento e exibição de dados enviados pelo formulário apresentado na figura 10:

Figura 16 - Salvando Dados no Banco de Dados, Código PHP

Interface gráfica do usuário, Texto

Descrição gerada automaticamente com confiança média

Fonte: Do próprio autor, 2022.

Segundo Bento (2021), a função *isset()*, é responsável por verificar se um envio do formulário aconteceu, caso aconteça, a página se conectara com o banco de dados através do *include\_once*, que tem por função chamar o código apresentado na figura 14 para o código em questão.

O método *POST* é utilizado para passar valores inseridos pelo usuário, são conhecidos como *arrays* superglobais (CONVERSE; PARK, 2003). Após acessar os valores inseridos atribuí-los às variáveis, a função *mysqli\_query* permite a execução de comando Standard Query Language (SQL), utilizando o comando *insert into* que indica um comando de inclusão de dados e *values* que indica a parte do comando onde são declarados os valores que serão salvos na tabela presente no banco de dados (COSTA, 2007). Ao salvar os dados, o *header* junto do parâmetro *location,* que segundo Vivas (2000), é responsável por redirecionar o usuário para uma outra página, no caso, será encaminhado para *endereco.php,* responsável por exibir os dados cadastrados pelo usuário e apresentada na figura 18.

Niederauer (2017) diz que cada script de código PHP existente na página deve começar com a *tag* *<?php* e terminar com *?>.* Essa combinação se torna muito útil, pois o PHP é utilizado para gerar os dados de forma dinâmica, enquanto o HTML é usado para formatar e exibir esses dados nas páginas, assim como mostrado nas figuras 17 e 18.

Figura 17 – Exibir Endereços Cadastrados, Código HTML

Linha do tempo

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

O trecho PHP no início do código, permite a busca dos dados salvos no banco de dados através do *SELECT \* FROM enderecos*, comando responsável por listar todas as informações na tabela (COSTA, 2007).

Figura 18 - Exibir Endereços Cadastrados, Código HTML (2)

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022

Figura 19 - Exibir Endereços Cadastrados, Página Web

Forma

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022

## 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 (NETO, 2020).

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 (GABARDO, 2017).

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 20 e 21 fazem a comparação de um código padrão *blade* e PHP.

Figura 20 - Exemplo de View com PHP

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

Fonte: (Silva, 2017 p. 61)

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 21 - Exemplo de *View* com *Blade*

Texto, Carta

Descrição gerada automaticamente

Fonte: (Silva, 2017 p. 61)

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 fazem parte da pasta raiz do projeto.

Figura - Pastas Padrões de um Projeto Laravel 9

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: Do próprio autor, 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 feita pela 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 os 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.

O MVC é um padrão de arquitetura com foco em aplicações *web.* Sua dinâmica é simples, todas as requisições da aplicação são encaminhadas para a camada *Controller*, que se comunica com a camada *Model* para processar a requisição, finalizando o processo na camada *View,* onde o resultado é exibido (LUCIANO; ALVES, 2011). A figura 23 ilustra o funcionamento do modelo MVC.

Figura 23 – Fluxo de dados padrão do Modelo MVC

Diagrama

Descrição gerada automaticamente

Fonte: (Gabardo 2017, p. 33)

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 24 25, 26, 27, 28, 29 e 30 exemplificam o mesmo processo de cadastro e listagem de endereços, porém, utilizando de *routes, models*, *views*, *controllers* e *migrations,* fazendo uso de *bootstrap* para a estilização.

Figura 24 - Listagem de Endereços Cadastrados, Página Web.



Fonte: Do próprio autor, 2022.

Figura 25 - Listagem de Endereços Cadastrados, Código BLADE

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Fonte Do próprio autor, 2022.

A diretiva foreach que segundo Niederauer (2007), oferece uma maneira mais ´fácil de “navegar” entres os elementos de um array, possibilitando a repetição do mesmo trecho de código enquanto houver endereços cadastrados. Nesse trecho de código, os campos enviados para o banco de dados são adicionados ao código e apresentados ao usuário.

Figura 26 - Listagem de Endereços Cadastrados, Código BLADE (2)

Texto

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Figura 27 - Controller do Formulário, Código PHP

Texto

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Como Luciano e Alves (2011) afirmam, o *Controller* acessa a camada *Model*, no caso a de endereço*,* após isso atribui os campos do formulário da figura 19 à variáveis, onde o atributo “endereco::*insert”* substitui o comando *INSERT INTO* na figura 16, onde é responsável por salvar os 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 28 - Controller Endereços Cadastrados, Código PHP

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Fonte Do próprio autor, 2022.

Com o uso do termo endereço::*all*(), substitui o uso do *SELECT \* FROM* utilizado na figura 18. O *Controller* permite o retorno da *View* de uma página junto dos dados armazenados no banco de dados.

Figura 29 - Model dos Dados de Endereço, Código PHP

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Responsável por interagir com o *front* e *back-end* da aplicação, a *Model* interage com as tabelas que foram criadas nas *Migrations* e posteriormente adicionadas ao banco de dados (SILVA, 2018). Após a *model* possuir as mesmas colunas e linhas que a *Migration*, ela se torna capaz de realizar a interação entre o lado cliente e lado servidor.

Figura 30 - Migration dos Dados de Endereço, Código PHP

Texto

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

*Migration* responsável por criar a tabela que terá o objetivo armazenar os dados enviados pelo formulário (SILVA, 2018). 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.

## 2.7 Banco de Dados

O banco de dados nos padrões SQL foi criado em meados dos anos 80 pela empresa Oracle, com o objetivo de armazenar e salvar informações e registros, sejam eles funcionais ou não funcionais. Para Christopher (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, 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 Geral de Proteção de Dados (LGPD), 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 Modelo Entidade-Relacionamento (MER) o Diagrama Entidade-Relacionamento (DER) 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 têm 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, como exemplificado na figura 31.

Figura - Exemplo DER

Uma imagem contendo objeto, relógio

Descrição gerada automaticamente

Fonte: (Araújo, 2021, p. 29).

Segundo Araújo (2021), relacionamentos são descritos através da cardinalidade, que tem como função indicar como as entidades se relacionam, Korth, Silberchatz e Sudarshan (2006) descrevem os tipos utilizados na modelagem:

* Um-para-um (1:1): uma instância A é associada com até uma instância B, e uma instância B está associada com no máximo uma instância em A.
* Um-para-muitos (1:n): uma instância A está associada a qualquer número de instâncias B, e uma instância B é limitado a estar associado a no máximo uma instância A.
* Muitos-para-muitos (n:n): uma instância A está associada a qualquer número de instâncias B e vice-versa. Certos autores chamam esta cardinalidade de m:n, por considerar que podem representar diferentes valores.

Utilizando o conceito de relacionamento, a seguinte figura exemplifica um Diagrama Entidade Relacionamento completo.

Figura - Exemplo Completo DER

Diagrama

Descrição gerada automaticamente

Fonte: (GARRIDO; BITU; SILVA, 2012, p. 6).

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:

* 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 e FK (Araújo, 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 deve-se ter alguns cuidados principalmente quando incluir uma linha na tabela que contêm 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 sendo eles: número inteiro, data, tamanho, *string* e *float*. 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 obrigatoriamente a chave estrangeira aparecer na chave primaria da tabela referenciada. O modelo lógico representa a visualização do banco de dados em tabelas, sendo exemplificado na figura abaixo.

Figura - Modelo Lógico Exemplo

Diagrama

Descrição gerada automaticamente

Fonte: (GARRIDO; BITU; SILVA, 2012, p. 7).

### 2.7.2 Normalização

A normalização é definida por Date (2004) como um conjunto de regras que ocasiona na construção de modelos mais robustos, com menos dependências entre seus elementos e menos redundância de informações, esse processo é denominado como forma normal, sendo utilizada para ter uma melhor projeção de um banco de dados. Há diversas formas normais, as mais utilizadas sendo primeira, segunda e terceira forma normal, sendo abreviadas como 1FN, 2FN e 3FN (HEUSER, 2009). Date (2004) explica cada forma normal e suas regras.

A 1ª Forma Normal (1FN) promove que em qualquer relação deve ter uma chave primária, assim como todo atributo deve ser atômico. Atributos compostos, como um Endereço, deve ser subdividido. Além disso, atributos multivalorados devem ser separados em outra relação como mostrado na tabela abaixo.

Figura - Tabela Cliente Não Normalizada

Tabela

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

A tabela de clientes apresenta atributos compostos como o endereço e multivalorado como o telefone, a tabela na primeira forma normal ficará desta forma:

Figura - Tabela Cliente e Telefone 1FN

Tabela

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

O atributo composto de endereço foi subdividido em colunas, já o atributo multivalorado telefone, foi separado em outra relação como pede as normas da primeira forma normal.

A segunda forma normal (2FN) ressalta que toda relação deve estar na 1FN e deve-se eliminar dependências funcionais parciais, ou seja, todo atributo que não seja dependente da chave primária. Para exemplificar, será usado exemplos de uma tabela de peças e fornecedor.

Figura - Tabela Peças não Normalizada

Tabela

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Essa tabela tem como chave primária composta as colunas id\_peça e id\_Fornecedor, as colunas Local\_Fornecedor e tel\_fornecedor dependem apenas da chave primária id\_fornecedor. A normalização desta tabela resultará na criação de uma tabela separada, pois assim é possível eliminar os atributos não dependentes da chave (HUDDLESTON, 2005).

Figura - Tabela Peça e Tabela Fornecedor 2FN

Tabela

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Huddleston (2005) afirma que a terceira forma normal estende o conceito de dependência funcional para a dependência funcional completa, que consiste na regra que todas as colunas não chave de uma tabela são identificadas com exclusividade pela chave primaria inteira, não somente parte dela. Heuser (2009), complementa essa definição afirmando que uma tabela se encontra na 3FN quando, além de estar na 2FN, não possui dependências funcionais transitivas ou indiretas. A figura 36 e 37 exemplificam o processo da terceira forma normal.

Figura - Tabela Venda não Normalizada

Tabela

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

A coluna Nome\_Vendedor não depende exclusivamente da chave primária, e sim da coluna id\_Vendedor, logo, como abordado por Heuser (2009), a dependência deve retirada para a tabela estar na 3FN. A tabela normalizada na terceira forma normal é apresentada na figura 37.

Figura - Tabela Venda e Vendedor 3FN

Tabela

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Após realizar as duas primeiras formas normais e retirar as dependências funcionais completas da tabela, agora ela se encontra na terceira forma normal (HUDDLESTON, 2005).

### 2.7.3 Dicionário de Dados

O dicionário de dados pode ser considerado um pequeno banco de dados que armazena toda a descrição do DER, onde deve ser possível visualizar e alterar as entradas do dicionário de dados correspondentes a elementos selecionados do DER (HEUSER, 2009). O autor afirma que para construir um dicionário de dados, pode-se optar por utilizar um processador de textos, uma planilha eletrônica ou um banco de dados, descrito como melhor opção pelo autor.

Elmasri e Navathe (BARBOSA; AVARENGA, 2012) afirmam que frequentemente durante a projeção de um banco de dados, os diagramas não abrangem todas as informações significativas sobre os dados que devem ser tratados, deixando a interpretação desses diagramas ineficientes, podendo gerar ambiguidades em sua implementação. A figura 40 exemplifica um dicionário de dados de uma tabela que representa um livro.

Figura - Exemplo Dicionário de Dados

Tabela

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Os metadados contém os nomes dos servidores de banco de dados, dos esquemas, das tabelas e das colunas, o tipo de dado da coluna, o tamanho em quantidade de bytes reservados para a coluna, o dicionário de dados da coluna e da tabela, se a coluna permite valores nulos, se a coluna é uma Foreign Key (FK), qual a quantidade de linhas da tabela e qual o tamanho da tabela em gigabytes (REIS, 2018, p.24).

## 2.8 UML

A *Unifed Modeling Language*, em português, linguagem de modelo unificada, 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 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, segundo Booch (1998) a UML 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.

Para Ribeiro (2013) uma das premissas da UML é o levantamento de requisitos, o levantamento de requisitos consiste em reunir todos os requisitos funcionais ou não funcionais, pegar todas as informações que o usuário do sistema quer e pretende 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, para Mello (2018) 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.

Segundo Carvalho (2009) e Mendonça (2014), descrevem que existem diversas técnicas de levantamento de requisitos, sendo um aspecto fundamental para o desenvolvimento de uma aplicação. O autor descreve algumas dessas técnicas:

* Amostragem: A amostragem é um processo exclusivo de uma sistemática de dados que representam uma parcela de um determinado conjunto de casos dentro do projeto. Quando os dados são escolhidos na amostragem, são observados, podendo-se assumir que a observação revelará informações necessárias e uteis para os dados. Para utilizar os seus benefícios como: amenizar os gatos, maximizar e acelerar completamente o processo de requisição de informações, melhoria na eficiência onde a informação passa a ser mais detalhada.

No processo de amostragem existem quatro passos que um desenvolvedor de sistemas deve seguir para planejar uma ótima amostragem:

1. Levantar os dados que devem ser descritos e coletados;
2. Definir os dados a serem amostrados na maioria dos casos são feitas perguntas para solucionar esse mistério e as principais são o que e quem?
3. Determinar a metodologia da amostra;
4. Decidir sobre a proporção que deve ter a amostra. Os dois primeiros tópicos fazem alusão ao sentido do desenvolvimento.

* Investigação: Diversas vezes em determinados casos alguns fatores informativos são terrivelmente difíceis de se obter por intermédio de entrevistas ou observação. Esses dados revelam um histórico da organização e sua direção. Em face destes casos deve-se utilizar da investigação, ou seja, a interpretação mais detalhada de documentos. Com a investigação, pode-se ter mais facilidade para detectar informações, tais como os documentos e entraves associadas ao investigado, dados financeiros e contextos organizacionais.
* Entrevistas:Uma entrevista dedicada ao levantamento de ideias é um diálogo com um fito específico, que por sua vez utiliza-se de um formato de perguntas e respostas. O intuito de uma boa entrevista inclui: coletar as necessidades de quem é entrevistado, o que auxilia na descoberta de problemas-chave que devem ser tratados, saber o que o entrevistado acha do sistema durante as fases de seu desenvolvimento, introduzir procedimentos formais e informais.
* Observação:analisar os comportamentos, decisões que podem caracterizar uma base bastante eficaz de levantamento de dados que ficam dispersas usando outras técnicas. Quando são tomadas decisões que afetam diversas fases da organização é de maneira imperiosa a observação de todos os níveis que possam ter interação com o sistema e através da observação faz-se possível capturar: o que é feito e não somente o que é explicado ou documentado.
* Prototipação*:* A prototipação é uma parte técnica muito importante para obter-se rapidamente as informações mais específicas de requisitos de dados do usuário. Comumente, a prototipação permite pegar os tipos de informação como: Reações do usuário de primeira instancia, opiniões do usuário com o intuito de auxiliar, modelar ou alterar o protótipo, Inovações que tragam novas funcionalidades capacidades, não pensadas antes da ligação com o protótipo, dados para revisão de tarefas e planos estabelecendo prioridades e refazer ou atualizar planos.

### 2.8.2 Diagrama de Casos de Uso

O diagrama de caso de uso armazena os deveres que o sistema deve fazer no viés do usuário. Ou seja, ele mostra as principais funções do sistema e a interação das funcionalidades para com os usuários do sistema. Nesse diagrama não se tem um aprofundamento em pequenos detalhes que ditam como o sistema realiza as funcionalidades. Esta ferramenta é derivada da especificação dos requisitos levantados (Bezerra, 2007).

O diagrama representa também as funcionalidades internas e externas do sistema, documenta também os serviços a serem feitos pelo sistema (Júnior, 2020). O caso de uso mostra os relacionamentos entre quem realiza as ações que por sua vez a maioria das vezes é um agente externo como os atores do diagrama (Bezerra, 2007).

Os atores, segundo Júnior (2020), são responsáveis por idealizar as entidades de fora do sistema que o integram ao longo de sua execução, as entidades podem vir a ser humanos como: usuário, aluno, secretaria, gerente, pode vir a serem dispositivos como: impressora e diversos outros equipamentos, pode vir também a ser um software como banco de dados, sistemas e aplicativos.

A figura a seguir apresenta um exemplo de diagrama de caso de uso:

Figura - Exemplo Diagrama de Caso de Uso

Diagrama

Descrição gerada automaticamente

Fonte: (PEIXOTO, 2016, p. 48).

Os diagramas de caso de uso são formados por exatos seis tópicos sendo eles (PENDER, 2004):

* Caso de uso: constata um certo comportamento-chave do projeto. Desprovido desse comportamento, o sistema não colocara os requisitos previstos para o ator. Todo diagrama de caso de uso mostra um fito específico que o sistema deve proporcionar ou alcançar um resultado de finalidade que ele tem que exercer.
* Relacionamento de extensão: indica uma forma reciclável do caso de uso, que para a condição e a execução de demais diagramas de caso de uso para tornar sua funcionalidade mais rápida e viável.
* Relacionamento de inclusão: indica uma forma de uso reciclável, que é transmitido e na operação de outros casos de uso. A incumbência pela abordagem que deve se tomar sobre porque utilizar o caso de uso incluído demonstra-se no caso de uso que o instancia.
* Associação: mostra uma relação entre os atores e casos de uso. Toda associação tem-se uma conversa que obrigatoriamente deve ser uma narrativa de caso de uso. Cada característica da narrativa oferece um quadro que pode auxiliar no manuseio dos casos de teste e na avaliação de artifícios de análise e implantação do caso de uso e da associação.
* Generalização: mostra a familiaridade com a herança entre casos de uso e os atores.
* Ator: desempenha um papel por uma só pessoa, dispositivo, sistema ou empresa, que tem interesse em operar um sistema efetuado com sucesso

### 2.8.3 Diagrama de Classes

Um dos tipos mais populares de diagramas presentes na UML é o diagrama de classes. Muito usado pelos engenheiros de *software* e toda a engenharia usa modelos para que os produtos possam ser edificados com base nesses modelos. A criação de modelos requer um alto nível de criatividade. Na verdade, não há uma solução clara e correta. Moldar é uma arte e requer muita experiência do modelador, que não só precisa ter conhecimento sobre sua tarefa de modelagem, mas também precisa ter conhecimento sobre o domínio do problema. Um modelo não apenas representa o sistema, mas também fornece uma melhor compreensão do problema e da simulação. Na etapa de análise os pré-requisitos apresentados até agora estão sujeitos a detalhamento, estrutura e revisão. Isso permite a modelagem conceitual do sistema (Tavares, 2008). Tudo isso com o objetivo de armazenar as plantas ou arquiteturas que o sistema deve conter, os diagramas de classes são diagramas que sua estrutura idealiza o que deve conter no sistema a ser modelado pelo engenheiro de *software*. Um diagrama de classes também representa um viés do padrão do modelo estrutural estático, que é compreendido como a unificação de todos os diagramas de classe e de objetos, nesse sentido, pode-se estruturar figuras tridimensional em muitos planos bidimensionais (Gudwin, 2010). A figura 42 demonstra um exemplo de diagrama de classe.

Figura - Exemplo Diagrama de Classes

Diagrama

Descrição gerada automaticamente

Fonte: (MACORATTI, 2004).

Macoratti (2004) define os elementos utilizados:

* Atributo: representa uma propriedade que todos os objetos da classe possuem, porém, cada objeto terá valores particulares para seus atributos.
* Métodos: podem ser descritos como ações que implementam uma operação.

Além desses elementos, o exemplo fez uso do conceito de relacionamento entre classes, exemplificado na figura 43.

Figura - Relacionamento Entre Classes

Diagrama

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Macorrati (2004) explica o conceito de cada relacionamento:

* Associação: pode ser descrita como um relacionamento estrutural entre elementos e define que objetos de uma classe estão ligados a objetos de outras classes.
* Herança: relacionamento entre um elemento mais geral e um mais específico. Onde o elemento mais específico herda as os atributos, propriedades e métodos do elemento mais geral ao qual está relacionado.
* Dependência: são relacionamentos de utilização no qual uma mudança na especificação de um elemento pode alterar a especificação do elemento dependente. A dependência entre classes indica que os objetos de uma classe usam serviços dos objetos de outra classe.
* Agregação: é um tipo de associação onde o objeto parte se torna um atributo do todo, ou seja, os objetos partes somente são criados se o todo ao qual estão agregados estão agregados seja criado.
* Composição: relacionamento entre um elemento todo e outros elementos que são as partes. As partes apenas podem pertencer ao todo, sendo criadas e destruídas com ele.

Caso os diagramas de classe forem efetuados com erros ou inconsistências, isso pode afetar os demais diagramas e, por conseguinte, o código também. Para garantir que os diagramas de classe demonstrem uma ótima qualidade, faz-se necessária a utilização de técnicas que inspecionem os diagramas com o fito de localizar possíveis falhas e anomalias. Nesse sentido, tem técnicas de inspeção que visam detectar os erros e incoerências contidos nos diagramas de classe, também ajuda a reduzir custos, uma vez que os possíveis erros iriam para os demais passos e seriam identificados e retificados nos passos iniciais do desenvolvimento do projeto (Santos, 2019).

### 2.8.4 Diagrama de Sequência

É um diagrama que visa definir a ordem dos acontecimentos em um meio ou processo, mostrando os métodos a serem usados entre os objetos e atores na ordem especificada. O diagrama é embasado no caso de uso tendo um diagrama de sequência para cada diagrama de caso de uso. O diagrama de sequência depende inteiramente do diagrama de classe, haja vista que os objetos possuem classes declaradas, contudo, o diagrama de sequência não é a melhor ferramenta para validar um diagrama de classe, porque, percebe-se muitos erros, lacunas e ausências em alguns pontos tendo a necessidade de declarar novos métodos (Guedes, 2014).

Ao contrário de outros diagramas de interação. Os diagramas de sequência permitem modelar trocas de mensagens entre objetos em ordem cronológica, as mensagens devem ser trocadas uma de cada vez segundo a sequência que é demonstrada no diagrama. Consequentemente, entender os componentes constituído por um diagrama de sequência é, portanto, a base para a compreensão do trabalho aqui apresentado. A seguir estão alguns dos elementos a partir dos quais um diagrama de sequência pode ser feito. Para cada um deles, é dado um exemplo mostrando a sintaxe usada para modelar o diagrama. A figura 44 exibe um exemplo de diagrama de sequência (JÚNIOR, 2012).

Figura - Exemplo Diagrama de Sequência

Diagrama

Descrição gerada automaticamente

Fonte: (GUEDES, 2009, p. 34).

Segundo Júnior (2012), as linhas tracejadas são chamadas de linhas de vida, que representam os objetos do sistema que participam do fluxo de troca de mensagens. Ela é representada por um retângulo horizontal, onde deve estar escrito a identificação do objeto juntamente do nome que descreve esse objeto.

O autor afirma que as mensagens representam uma troca de informação entre entidades, com o objeto tendo a possibilidade de enviar mensagens para si mesmo. As mensagens são representadas por setas dispostas horizontalmente dentro do diagrama, sendo elementos cruciais para a projeção de um diagrama de sequência. A mensagem ao topo do diagrama é considerada a primeira mensagem, com sua ordem de leitura sendo da esquerda para a direita, e de cima para baixo.

### 2.8.5 Diagrama de Atividade

Silva, Valentim e Conte (2011) descrevem o diagrama de atividade como um dos tipos de diagramas que fazem parte das fases iniciais de desenvolvimento, sendo modelado previamente. Em comparativo com os outros diagramas comumente utilizado nessa fase de desenvolvimento, o diagrama de atividade é utilizado para ilustrar a execução lógica das ações que podem ser realizadas em uma aplicação, sendo uma alternativa para representar as características inerentes aos fluxos de execução que afetam o ciclo de vida dos projetos nos projetos das aplicações. Portanto, desenvolver este diagrama já pensando na utilização da aplicação final, ocasiona numa maior precisão e qualidade da aplicação projetada neste diagrama. Para exemplificar o uso desse diagrama, a figura 45 exemplifica o processo de compra de um produto.

Figura - Exemplo Diagrama de Atividade

Diagrama

Descrição gerada automaticamente

Fonte: (SILVA; VALENTIM; CONTE, 2014, p. 93).

Conforme Guedes (2011), O diagrama de atividades tem o dever de descrever o caminho a ser percorrido até a conclusão de uma ação específica, podendo ser representada por um método com certo grau de complexidade, um algoritmo, ou mesmo por um processo completo. O diagrama de atividade se concentra na ilustração do fluxo de controle de uma atividade, como representado na figura 45.

# DESENVOLVIMENTO

Este capitulo tem por finalidade abordar o desenvolvimento da aplicação *web* de criação e atribuição de treinos. Através de diagramas UML para demonstrar as funcionalidades da aplicação e a modelagem de dados para a esquematização do banco de dados.

## 3.1 Levantamento de Requisitos

Para a elaboração da aplicação, foi utilizada a entrevista como técnica de levantamento de requisitos, que por sua vez, levou a concepção dos seguintes requisitos funcionais e não funcionais.

Figura - Requisitos Funcionais e Não Funcionais

Tabela

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

Os requisitos funcionais baseiam-se nas funcionalidades indispensáveis da aplicação.

Figura - Requisitos Não Funcionais

Tabela

Descrição gerada automaticamente

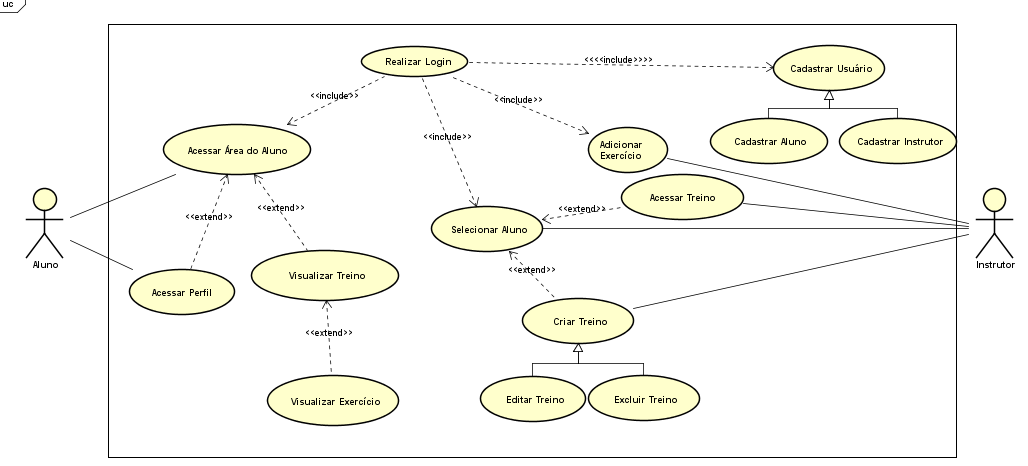
Fonte: Do próprio autor, 2022.

Os requisitos não funcionais demonstram formas de atender os funcionais, além de prover mais detalhes sobre as funcionalidades e elementos que serão utilizados no desenvolvimento.

## 3.2 Diagrama de Casos de Uso

O diagrama apresentado na figura abaixo representa as funcionalidades disponíveis para cada ator.

Figura - Diagrama de Casos de Uso



Fonte: Do próprio autor, 2022.

O após realizar o login, aluno acessa a página área do aluno, onde ele tem a opção de visualizar seu treino e acessar seu perfil. O instrutor, ao realizar login, seleciona um aluno para acessar, criar, editar e excluir o treino.

## 3.3 Diagramas de Classes

O diagrama de classe é responsável por apresentar as classes junto de seus atributos e métodos que serão utilizadas na aplicação.

Figura - Diagrama de Classes

Diagrama

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

O diagrama descreve as classes, seus atributos, métodos e seus relacionamentos, sendo cruciais para a projeção correta da aplicação.

## 3.4 Diagrama de Atividade

A aplicação é composta por três diagramas de atividade no total, responsáreis por demonstrar o fluxo que é realizado nos processos de cadastro, *login* e treino.

Figura - Diagrama de atividade: Cadastro

Diagrama

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

O fluxo apresentado na figura 50 ilustra o processo para realizar um cadastro de usuário. Caso o usuário tenha inserido algum dado que não condiz com a validação, ele deve inserir os dados novamente.

Figura - Diagrama de atividade: *Login*

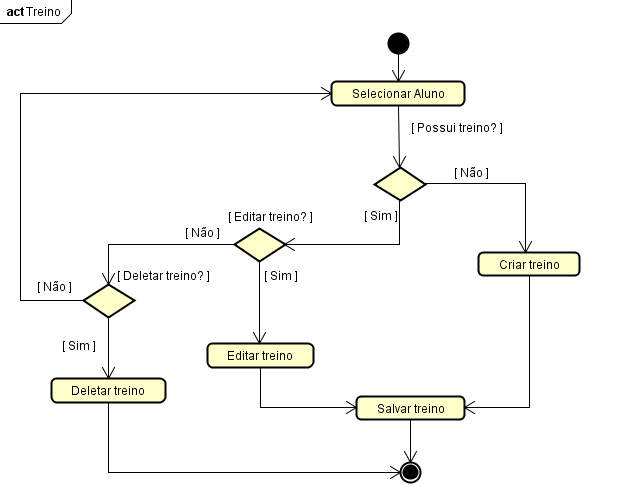
Diagrama

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022

Este fluxo apresentado na figura 51 descreve o processo que o usuário passa para realizar o *login*. Ao inserir o CPF e senha, os dados passam por uma validação onde o usuário é redirecionado para a página *home* caso os dados sejam válidos, caso não, ele deve inserir seus dados novamente.

Figura - Diagrama de atividade: Treino



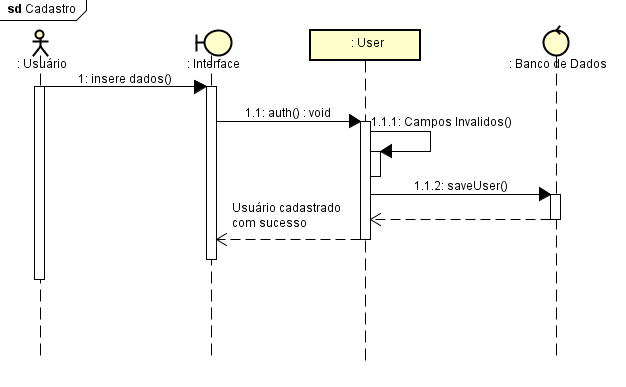
Fonte: Do próprio autor, 2022.

A figura 51 apresenta o fluxo que descreve as ações que o instrutor passa ao interagir com os treinos. O instrutor seleciona um aluno, caso ele não tenha um treino definido, o instrutor deve criar um, caso possua, ele tem as opções de editar o treino ou deletá-lo.

## 3.5 Diagrama de Sequência

A aplicação possui quatro diagramas de sequência no total, sendo responsáveis por exibir o processo que a ação do ator acarreta, mostrando o fluxo através de métodos e classes até chegar ao banco de dados.

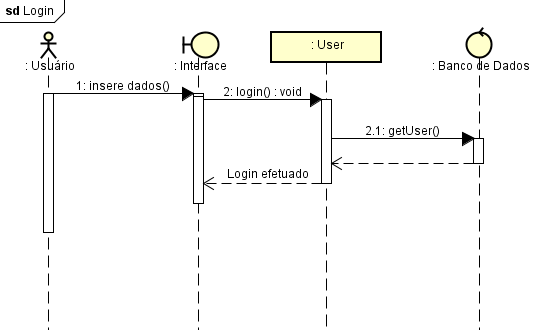
Figura - Diagrama de Sequência: Cadastro de Usuário



Fonte: Do próprio autor, 2022.

A figura 53 apresenta o processo de cadastro de um usuário, onde ele insere seus dados, e após isso, ocorre a validação deles, caso alguma informação esteja inválida, o cadastro não é realizado, caso esteja, o sistema armazena o usuário no banco de dados.

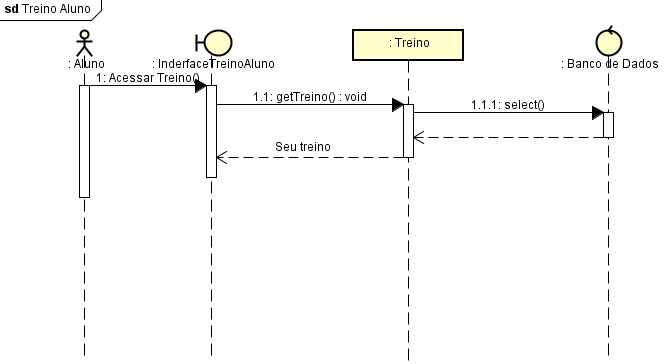
Figura - Diagrama de Sequência: *Login*



Fonte: Do próprio autor, 2022

O processo de *login* é demonstrada na figura 54, onde o usuário insere seus dados, e o sistema acessa no banco de dados os dados idênticos aos inseridos pelos usuários, realizando o *login*.

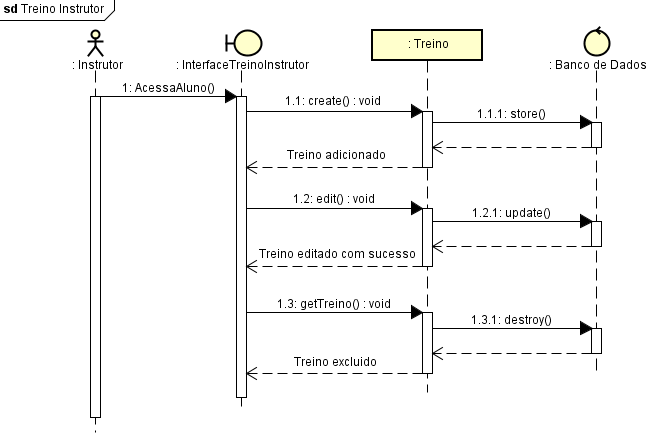
Figura - Diagrama de Sequência: Treino Aluno



Fonte: Do próprio autor, 2022.

Na figura 55, o processo apresentado descreve o acesso do aluno ao treino criado pelo instrutor e atribuído à ele.

Figura - Diagrama de Sequência: Treino Instrutor



Fonte: Do próprio autor, 2022.

A figura 56 é responsável por demonstrar as ações que o instrutor pode realizar em relação aos treinos, onde ele seleciona um dos alunos exibidos e realiza a criação, edição, exclusão e atualização de um treino.

## 3.6 DER

O diagrama entidade relacionamento responsável por demonstrar os elementos do banco de dados da aplicação possui três entidades e diversos atributos que serão convertidos em tabelas no modelo lógico.

Figura - DER

Diagrama

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

O Diagrama Entidade Relacionamento define as entidades e requisitos do banco de dados, e de que maneira elas se relacionam.

## 3.7 Modelo Lógico

O modelo lógico representa o modelo conceitual com mais detalhes, transformando entidades em tabelas e definindo tipos de dados aos seus atributos.

Figura - Diagrama Entidade Relacionamento

Diagrama

Descrição gerada automaticamente

Fonte: Do próprio autor, 2022.

A figura 58 apresenta as entidades presentes no diagrama, elas se relacionam entre si tornando o fluxo de dados possível.

## 3.8 Aplicação

# CONCLUSÃO

A partir do estudo aprofundado da UML e seus diagramas principalmente os diagramas de classe, diagrama de caso de uso, diagrama de atividade, diagrama de sequência e dos requisitos funcionais e não funcionais que estão presentes na aplicação, possibilitarem e facilitaram o planejamento no desenvolvimento daaplicação.

A aplicação foi projetada para atender as necessidades de uma academia, para isso foram utilizadas ferramentas de extrema importância para o desenvolvimento do projeto, como o *framework* Laravel que a partir do uso de suas bibliotecas auxiliou tanto no desenvolvimento *front-end*, quanto no *back-end,* facilitando o desenvolvimento geral da aplicação, a plataforma Azure da Microsoft para hospedagem.

Considerando o atual projeto, planeja-se atualizar a aplicação adicionando novas funcionalidades, dentre elas estão: opções de envio de duvidas ao instrutor, dietas específicas para o biotipo de cada aluno, cadastro de alunos em atividades como dança e luta esportiva, novos exercícios, pagamento de planos e mensalidades, sistema de venda de suplementação e outros produtos.

De modo geral, o sistema se demonstrou eficaz e competente, atendendo as funcionalidades necessárias, o que facilitou de maneira significativa a gestão de treinos, não sendo necessário o uso de papel.

# REFERÊNCIAS

ALVES, William Pereira. **Banco de dados**. Saraiva Educação

, SA, 2014.

ARAÚJO, M. **MODELAGEM DE DADOS**. TEORIA E PRÁTICA. Revista Saber Digital, *[S. l.]*, v. 1, n. 01, p. 27–64, 2021. Disponível em: https://revistas.faa.edu.br/SaberDigital/article/view/1029. Acesso em: 11 nov. 2022.

Barbosa, Eduardo Pinto; Alvarenga, Geoflavia Guilarducci de. **Uma Proposta para Ensino de Dicionário de Dados em Projetos de Bancos de Dados**. 2012. Trabalho de Conclusão de Curso (Graduação) - Universidade de Brasília, Brasília, 2012.

BAZZI, Cláudio Leones. **Introdução à Banco de Dados**. Mato Grosso: 2018. 90 p.

BENTO, Evaldo Junior. **Desenvolvimento web com PHP e MySQL**. São Paulo: Editora Casa do Código, 2021. 419 p.

BEZERRA, Eduardo**. Princípios de análise e Projeto de sistemas com UML**. 2. ed. Rio de Janeiro: Elsevier, 2007

BOOCH, Grady Grady. **UML**: Guia do Usuário. 2. ed. rev. e atual. Rio De Janeiro: Elsevier Brasil, 2006. 474 p.

BOOTSTRAP. **Sobre**: História. Estados Unidos: Bootstrap, [201-]. Disponível em: https://getbootstrap.com/docs/5.2/about/overview/. Acesso em: 27 out. 2022.

CALDEIRA, Carlos. **Introdução ao HTML**. 2015. 48 p.

CARDOSO, Giselle Cristina; CARDOSO, Virgínia M. **Sistema de banco de dados**. Saraiva Educação SA, 2017.

CARVALHO, Predro F. **Técnicas de Levantamento de Requisitos**. 2009. Disponível em: http://pedrofcarvalho.com.br/PDF/ENGENHARIA\_ANALISE\_LEVANTAMENTO\_REQUSITOS\_2.pdf. Acesso em: 27 nov. 2022.

CINTRA, Glauber Ferreira. **Banco de Dados**. Ceará: Instituto Federal de Educação. 2012.

CONVERSE, Tim; PARK, Joyce. **PHP: a bíblia**. Gulf Professional Publishing, 2003. 868 p.

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

COSTA, Rogério Luís de C. **SQL.** Guia Prático. 2. ed. Rio de Janeiro: Brasport, 2007.

DALL’OGLIO, Pablo. **PHP Programando com Orientação a Objetos** 3. ed. São Paulo: Novatec Editora, 2015.

DATE, C. J. **Introdução a sistemas de bancos de dados**. 8.ed. Rio de Janeiro: Campus, 2004.

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

FERREIRA, Brenda Sotero. **FRAMEWORK LARAVEL**: UM ESTUDO DE CASO FULL STACK DEVELOPMENT. 2021. 55 f. Trabalho de conclusão de curso (Ciência da Computação) - Universidade Federal de Ouro Preto, Minas Gerais, 2021.

FLATSCHART, Fábio. **HTML 5-Embarque Imediato**. 1. ed. Rio de Janeiro: Brasport, 2011. 256 p.

GABARDO, Ademir C. **Laravel para ninjas**. 1. ed São Paulo: Novatec Editora, 2017. 184 p.

GARRIDO, Gleiciana Martins; BITU, Luiza Karine; Silva Maria Elayne. **Projeto:** Gerenciamento de Locadora de Vídeo. Ceará: 2012

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

GUDWIN, Ricardo R. **Introdução à linguagem UML**. Disponível em: www. dca. fee. unicamp. br/~ gudwin/ftp/ea976/Estruturais2010. pdf>. Acesso em: 19 out. 2022.

GUEDES, Gilleanes T. A. **UML 2**: Guia Prático. 2. ed. São Paulo: Novatec Editora, 2014. 192 p.

GUEDES, Gilleanes T. A. **UML 2**: Uma Abordagem Prática. 2. ed. São Paulo: Novatec Editora, 2011.

GUEDES, Gilleanes TA. **UML 2-Uma abordagem prática**. 3. ed. São Paulo: Novatec Editora, 2018.

HEUSER, Carlos Alberto. **Projeto de banco de dados:** Volume 4 da Série Livros didáticos informática UFRGS. Porto Alegre: Bookman Editora, 2009.

HUDDLESTON, James. **Iniciando em Banco de Dados com VB 2005**: De Novatos a Profissionais. 1ª. ed. Rio De Janeiro: Ciência Moderna, 2008. 536 p

JOBSTRAIBIZER, Flávia. **Criação de sites com o CSS**. São Paulo: Universo dos Livros Editora, 2009. 146 p.

JÚNIOR, Edwar Saliba. **Diagrama de Caso de Uso**. Minas Gerais: Instituto Federal de Educação. 2020.

JÚNIOR, Edwar Saliba. **Diagrama de Classes**. Minas Gerais: Instituto Federal de Educação. 2020.

JÚNIOR, Sebastião Estefânio Pinto Rabelo. **Verificação de conformidade entre diagramas de sequência UML e código Java**. 2012. 76 f. (Dissertação de Mestrado em Ciência da Computação) Programa de Pós-graduação em Ciência da Computação, Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande - Paraiba - Brasil, 2012. Disponível em: http://dspace.sti.ufcg.edu.br:8080/jspui/handle/riufcg/1626

LEWIS, Joseph R; MOSCOVITZ, Meitar. **Css avançado**. São Paulo: Novatec Editora, 2010. 416 p.

KORTH, H. F; SILBERCHATZ, A.; SUDARSHAN, S. **Sistema de banco de dados**. 5.ed. Rio de Janeiro: Campus, 2006.

LARMAN, Craig. **Utilizando UML e padrões**. Porto Alegre: Bookman Editora, 2000.

LUCIANO, Josué; ALVES, Walisson Joel Barberá. **PADRÃO DE ARQUITETURA MVC**: MODEL-VIEWCONTROLLER. 3ª. ed. São Paulo: [*s. n.*], 2011. v. 01.

MACORATTI, José Carlos**. UML - Diagrama de Classes e objetos**. 2004. Disponível em: https://www.macoratti.net/net\_uml1.htm. Acesso em: 27 nov. 2022.

MCCOOL, Shawn. **Laravel starter**. Packt Publishing, 2012. 64 p.

MELLO, Alessandra Callado Bezerra de. **Levantamento de requisitos por meio da análise da atividade e da tarefa para sistemas digitais**. 2018.

Dissertação de Mestrado. Universidade Federal de Pernambuco. 2018.

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

MENDONÇA, Ricardo Augusto Ribeiro de. Levantamento de requisitos no desenvolvimento ágil de software. 2014. Pontifícia Universidade Católica de Goiás – Goiânia. 2014.

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

NIEDERAUER, Juliano. **PHP para quem conhece PHP**. 5. ed. São Paulo: Novatec Editora, 2017. 544 p.

NIEDERAUER, Juliano. **Web Interativa com Ajax e PHP**. 1. ed. São Paulo: Novatec Editora, 2007. 288 p.

PEDROSO, Robertha Pereira. **APOSTILA DE HTML**. Rio De Janeiro: [*s. n.*], 2007. 101 p.

PEIXOTO, Anderson Gomes. O uso de metodologias ativas como ferramenta de potencialização da aprendizagem de diagramas de caso de uso. **Outras Palavras**, Brasília, DF, v. 12, n. 2, p. 36-50, 2016.

PENDER, Tom. “UML, A Bíblia”. 2ª edição. Rio de Janeiro: Elsevier, 2004.

PRESCOTT, Preston. **Programação em JavaScript**. Babelcube Inc., 2016.

REIS, Débora Gomes dos. **Uso de técnicas de mineração de dados para identificar estruturas similares de bancos de dados**. 2018. 89 f. Dissertação (Mestrado Profissional em Computação Aplicada) - Universidade de Brasília, Brasília, 2018.

RIBEIRO, Karina Flaviana. **Estudo de caso sobre o uso da linguagem de modelagem UML no processo de levantamento de requisitos no desenvolvimento de aplicações do LCC-UFMG**. 2013. 37 f. Monografia de Especialização (Especialização em Educação) - Universidade Federal de Minas Gerais, Minas Gerais, 2013.

SANTOS, Déborados et al. **Técnicas de Inspeção para Diagramas de Classes UML**: Uma Revisão Sistemática. Anais da III Escola Regional de Engenharia de Software, p. 41-48, 2019.

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. 198 f. 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. São Paulo: Novatec Editora, 2010.

Silva, Leonardo Webster Ribeiro da. **Estudo Dos Benefícios Da Utilização Do Laravel Framework Na Manutenibilidade De Software**. 2018. 122 f. Trabalho de conclusão de curso (Curso em Ciência da Computação), Faculdades Integradas de Caratinga – Caratinga. 2018.

SILVA, Williamson; COSTA Valentim, Natasha M.; CONTE, Tayana. Projetando diagramas de atividade visando a usabilidade de aplicações interativas. In: **Proceedings of the 13th Brazilian Symposium on Human Factors in Computing Systems**. 2014. p. 349-352.

STEFANOV, Stoyan. Padrões JavaScript. São Paulo: Novatec, 2011.

TAVARES, Deisymar Botega. **Procedimento de análise para validação de diagrama de classes de domínio baseado em análise ontológica**. 2008. 87 f. Dissertação(Pós-Graduação em Ciência da Computação) – Universidade Federal de Viçosa, Minas Gerais, 2008.

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 f. Monografia (Curso Super de Tecnologia da Informação) - Universidade de Cruz Alta (UNICRUZ), Rio Grande do Sul, 2016.

QUIERELLI, Davi Antonio. **Criando Sites Com Html-css-php**. Santa Catarina: 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.

VIVAS, Mauricio. **Curso de Aplicações WEB em PHP**. CIPSGA: jun. 2000.