

**CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
ETEC DA ZONA LESTE**

Novotec Desenvolvimento de Sistemas AMS

João Pedro Silva de Oliveira

Luisa Santos Silva

Vinicius Valero Chabariberi

**NAVIGATE BUY: Sistema Web Agregador de Compras Online
Seguras e Simplificadas**

São Paulo

2024

João Pedro Silva de Oliveira

Luisa Santos Silva

Vinicius Valero Chabariberi

**NAVIGATE BUY: Sistema Web Agregador de Compras Online
Seguras e Simplificadas**

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

São Paulo

2024

RESUMO

Primeiramente, a compra online é uma tarefa muito comum nos dias de hoje, e os perigos e riscos que um consumidor pode encontrar são diversos, desde golpes a roubos de dados pessoais, causando uma preocupação ainda maior devido ao crescimento de consumidores online nos últimos anos. Com o objetivo de tornar a experiência em compras online uma tarefa rápida e mais segura, será necessário desenvolver um sistema web que irá utilizar as técnicas web crawler e web scraping. Para isso, agregar diversos produtos de lojas renomadas no ambiente digital através do sistema web e suas técnicas de extração de dados é essencial e permitirá que o consumidor tome sua decisão final com base em análises de dados como os comentários retirados de sites de reclamação e gráficos. Além disso, a compra online se torna algo demorado quando se faz pesquisas para comparar preços de um produto em específico, gastando várias horas para encontrar a melhor opção sem a certeza de ter de uma boa segurança. Esperamos que o sistema web possa direcionar os consumidores para sites seguros e recomendados por outros compradores, facilitando a busca pelos menores preços de produtos entre variados sites e proporcionando uma maior segurança. Portanto, pretende-se contribuir para avanço do comércio online oferecendo uma maior confiabilidade desta tarefa para a principal peça desta engrenagem, o consumidor.

Palavras-chave: Compra online. Consumidor. Sistema Web. Simplicidade. Segurança.

ABSTRACT

First of all, online purchasing is a very common task nowadays, and the dangers and risks that a consumer can find are diverse, from scams to stealing personal data, causing even greater concern due to the increase in online consumers in the last few years. The aim of this work is to make online purchasing a fast and secure task, it will be necessary to develop a web system that will use web crawler and web scraping techniques. For that, aggregating different products from renowned stores in the digital world through a web system and its data extraction techniques is essential and allows the consumer to make their final decision based on data analysis such as comments taken from complaint websites and graphics. Furthermore, online purchasing can become time-consuming when searching to compare prices of a specified product, spending several hours to find the best option without being sure there is good security. As a result, we expect that the web system will direct consumers to secure and recommend sites from other buyers, facilitating the search for lower product prices among these sites and providing greater security. Therefore, the intention is to contribute to the advancement of online purchasing by offering greater confidence in this task to the principal part of this gear, the consumer.

Keywords: Online purchasing. Consumer. Web System. Simplicity. Security.

LISTA DE ILUSTRAÇÕES

Figura 1 - Exemplo de código em HTML para a criação de uma avaliação simples .	12
Figura 2 - Resultado do código HTML.....	15
Figura 3 - Exemplo de código em CSS	16
Figura 4 - Conexão entre CSS e HTML	16
Figura 5 - Exemplo de código em CSS para a criação de uma estilização	17
Figura 6 - Resultado do código CSS	18
Figura 7 - Código construído com a linguagem JavaScript	19
Figura 8 - Resultado do código com HTML, CSS e JavaScript	20
Figura 9 – Resultado de caixa de alerta com JavaScript	20
Figura 10 - Exemplo de código para aplicação de um login simples em React	21
Figura 11 - Resultado da aplicação de um login simples feito em React	22
Figura 12 - Exemplo de código para aplicação de cálculo de média em Python	23
Figura 13 - Resultado do código do cálculo da média em Python pelo terminal	24
Figura 14 - Exemplo de inicialização de um projeto NPM	25
Figura 15 - Exemplo de inserção dos dados exigidos pelo NPM	26
Figura 16 - Exemplo de instalação do Tailwind	28
Figura 17 - Exemplo de criação da conexão entre HTML e o Tailwind	29
Figura 18 - Acrescentando diretivos do Tailwind	29
Figura 19 – Exemplo executando comando Tailwind	29
Figura 20 - Código HTML com a implementação do Tailwind	30
Figura 21 - Resultado do código HTML com a implementação do Tailwind	31
Figura 22 - Exemplo de código para aplicação de cálculo de média com o Flask ...	32
Figura 23 - Tela de inserimento dos dados em HTML	33
Figura 24 - Resultado da codificação com o Flask no terminal	34
Figura 25 - Exemplo de codificação em Python com o Scrapy	34
Figura 26 - Resultado da codificação com o Scrapy	36
Figura 27 - Exemplo de código com a AwesomeAPI	36
Figura 28 - Resultado da aplicação com a AwesomeAPI	37
Figura 29 - Exemplo da criação de um DER	38
Figura 30 - Exemplo da criação de um MER	39
Figura 31 - Exemplo de um código de banco de dados MySQL	40
Figura 32 - Resultado de uma consulta de uma tabela MySQL	41

Figura 33 - Exemplo de Wireframe de baixa fidelidade.....	41
Figura 34 - Exemplo de Wireframe de média fidelidade.....	42
Figura 35 - Exemplo de Wireframe de alta fidelidade.....	43
Figura 36 - Exemplo de Diagrama de Casos de Uso	44
Figura 37 - Exemplo de documentação de Caso de Uso	45
Figura 38 - Exemplo de Diagrama de Sequência.....	46
Figura 39 - Exemplo de Diagrama de Atividade	47
Figura 40 - Exemplo de Diagrama Máquina de Estados	48

LISTA DE ABREVIATURAS E SIGLAS

Application Program Interface (API)

Cascading Style Sheet (CSS)

Diagrama Entidade Relacionamento (DER)

Hypertext Markup Language (HTML)

Hypertext Transfer Protocol (HTTP)

JavaScript (JS)

Modelo Entidade Relacionamento (MER)

Node Package Manager (NPM)

Sistema Gerenciador de Banco de Dados (SGBD)

Unified Modeling Language (UML)

Uniform Resource Locator (URL)

User Experience (UX)

SUMÁRIO

1. INTRODUÇÃO	9
2. REFERENCIAL TEÓRICO	11
2.1. Dificuldade na pesquisa de compras online seguras e simplificadas.....	11
2.2. HTML.....	11
2.3. CSS	15
2.4. JavaScript	18
2.5. React.....	21
2.6. Python	23
2.7. NPM.....	24
2.8. Web Crawler	27
2.9. Web Scraping.....	27
2.10. Framework	27
2.10.1. Tailwind.....	28
2.10.2. Flask.....	31
2.10.3. Scrapy	34
2.11. APIs.....	36
2.11.1. AwesomeAPI	36
2.12. Banco de Dados	38
2.12.1. Modelagem de Dados	38
2.12.2. MySQL.....	39
2.13. UX.....	41
2.14. Wireframe	41
2.15. UML.....	43
3. DESENVOLVIMENTO.....	49
4. CONSIDERAÇÕES FINAIS	50
REFERÊNCIAS.....	51

1. INTRODUÇÃO

Não é novidade que a internet conquistou o mundo. Conforme o site E-Commerce Brasil (2024), o e-commerce está projetado para crescer de R\$ 349 bilhões em 2023 para R\$ 557 bilhões em 2027, evidenciando a transição dos padrões de consumo. Com esse aumento, a necessidade de ferramentas que tornem as compras online mais seguras e simplificadas tornou-se crucial. O Navigate Buy foca no desenvolvimento de um sistema web que busque assegurar e simplificar a experiência do consumidor nas compras online.

Apesar da facilidade e acessibilidade das compras online, os consumidores enfrentam dificuldades com a análise de melhores ofertas e a verificação da segurança dos sites de compra. Segundo Bolzani (2022), um levantamento feito pela Octadesk em conjunto com a Opinion Box mostrou que 73% dos consumidores declaram que os preços na internet são mais acessíveis. Porém, de acordo com Faustino e Lobato (2023), uma pesquisa apurada pelo Fórum Brasileiro de Segurança Pública (FBSP) indica que em 2022, houve uma elevação de 65,2% de fraudes em âmbito online.

Diante dessa problemática, a hipótese central é a implementação de um sistema web agregador de compras online que facilite a tomada de decisões dos consumidores, integrando a análise de lojas e seus respectivos produtos, apresentação de diferentes preços e comentários de avaliação para uma melhor comparação, conversão de moedas em tempo real e uma segurança mais eficiente em suas compras.

Assim, o objetivo geral é construir um sistema web que permita que os consumidores comparem produtos e preços entre diferentes lojas verificadas, buscando uma maior segurança em suas compras, analisando as avaliações vigentes de diversos produtos com base em conteúdos de sites de reclamação, sendo possível ver conversões de moedas em tempo real para maior acessibilidade. Dentre tais meios de abordagem, estão inclusos:

- Aprofundar as pesquisas e estudos sobre as dificuldades dos consumidores em seus processos de compra online;
- Analisar os principais requisitos de um sistema web que vise facilitar as dificuldades enfrentadas pelos compradores;

- Avaliar a eficácia da conversão de moedas em tempo real em respectivas compras importadas, examinando como isso facilita a transparência nos preços e reduz a incerteza do consumidor;
- Implementar uma eficiente aplicação web, fornecendo informações relevantes, ajudando a minimizar as adversidades enfrentadas pelos compradores;
- Avaliar o impacto da comparação de preços e comentários de avaliação na tomada de decisões dos consumidores.

Em vista disso, a metodologia adotada incluiu pesquisas exploratórias qualitativas e quantitativas para investigar as dificuldades dos consumidores e o impacto de um sistema web agregador de compras online seguras e simplificadas. Dados estatísticos foram coletados e analisados para o desenvolvimento que atendesse às necessidades identificadas, proporcionando uma ferramenta eficaz para a tomada de decisões nas compras online.

Nos capítulos sucessores, serão apresentadas todas as etapas de desenvolvimento e fundamentação teórica do Navigate Buy, que é sustentada por diversos autores relevantes na área, como Guedes, Mitchell, Pressman e Maxim.

2. REFERENCIAL TEÓRICO

Nesta seção, designa-se o problema de pesquisa que deu origem à necessidade do sistema web, assim como as tecnologias adotadas para seu desenvolvimento, proporcionando, portanto, o embasamento teórico essencial para a concepção do Navigate Buy.

2.1. Dificuldade na pesquisa de compras online seguras e simplificadas

Segundo Sé (2023), um estudo realizado pela consultoria Offerwise em janeiro de 2023 mostrou que nove em cada dez brasileiros pesquisam online antes de fazer uma compra. Isso reflete um aumento na adoção do ambiente online, com indivíduos dedicando tempo às pesquisas para encontrar o que desejam.

Diante disso, conforme abordado por Dias e Hemais (2015), apesar da crescente inclusão de pessoas no ambiente virtual, ainda existem receios relacionados a questões de segurança e privacidade. Portanto, o foco principal é atender às necessidades do consumidor e reduzir suas adversidades nas compras online, proporcionando mais facilidade na busca por ofertas e maior segurança.

2.2. HTML

Segundo Cardoso (1999), *Hypertext Markup Language* (HTML), que traduzido significa Linguagem de Marcação de Hipertexto, é uma linguagem baseada em arquivos-texto, onde pode ser editada por meio de um programa.

Conforme Freeman e Freeman (2006), um navegador tem a função de solicitar uma página HTML a um servidor, recebendo-a e exibindo-a em uma janela. O HTML diz ao navegador tudo sobre o conteúdo e a estrutura da página, constituída por *tags*.

Para Miletto e Bertagnolli (2014), a função das *tags* é representar os elementos que compõem uma página envolta por sinais (<) e (>). Essas *tags* podem aparecer em pares, porém elas sempre têm um começo e um fim determinado por (/).

Figura 1 - Exemplo de código em HTML para a criação de uma avaliação simples

```

1  <!DOCTYPE html>
2  <html Lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Avaliar || NavigateBuy</title>
7  </head>
8  <body>
9      <div class="container">
10         <div class="header">
11             <h1>Avaliação</h1><br>
12         </div>
13         <p>Informe os seus dados para fazer a avaliação</p><br>
14         <form id="meuForm">
15             <label for="nome">Nome: </label>
16             <input type="text" id="nome" name="nome">
17             <br><br>
18
19             <label for="email">Email:</label>
20             <input type="email" name="email" id="email">
21             <br><br>
22
23             <label for="comentarios">Comentários: </label><br>
24             <textarea name="comentarios" id="comentarios" cols="30" rows="10"></textarea>
25             <br><br>
26
27             <button type="submit">Enviar</button>
28         </form>
29     </div>
30 </body>
31 </html>

```

Fonte: Do próprio autor, 2024.

O código exibido acima resultará em uma interface de uma avaliação simples em HTML que será representada na próxima figura.

As principais *tags* utilizadas neste exemplo foram:

- `<!DOCTYPE html>`: é obrigatória do corpo do HTML, ele mostra ao navegador, qual é o tipo de linguagem que está sendo utilizada;
- `<html>`: declara o início e o fim para que o navegador funcione da melhor maneira;
- `Lang`: é um atributo que define o idioma de um elemento para os desenvolvedores saberem;
- `<head>`: armazena todas as informações do arquivo, como título e rotas para outros arquivos;
- `<meta>`: tem a função de fornecer informações para o navegador a

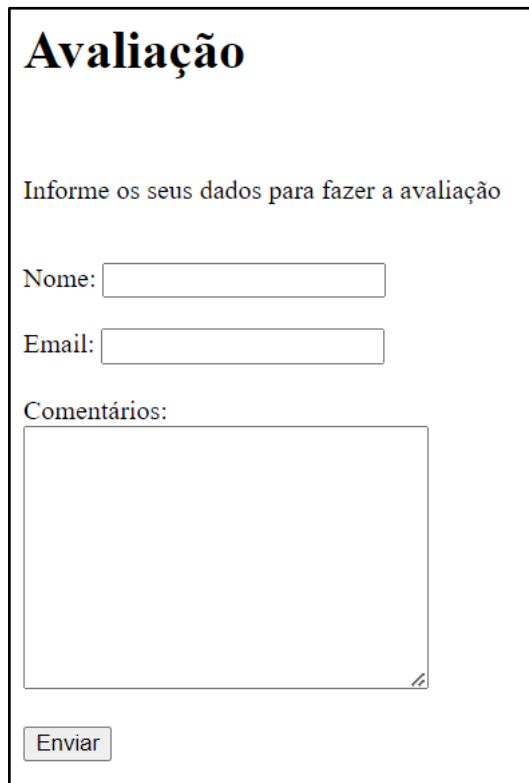
categorizar uma página, além de adaptar a página a diferentes tipos de tela automaticamente;

- *Charset* UTF-8: este atributo consiste em indicar qual é o formato de codificação para os navegadores, neste caso, o UTF-8;
- *Name*: o atributo “*name*” tem a função de identificar um elemento por um nome;
- *Viewport* e *Content*: os dois atributos tem a função de adaptar a página *web* feita para computador, às diferentes proporções de tela, como as de um celular;
- *<title>*: indica o que será exibido na aba do navegador da página;
- *<body>*: é fundamental para a estrutura do HTML, ela determina onde está localizado tudo visível para o usuário;
- *<div>*: as diversas divisões servem para dividir o conteúdo que está presente na página. No exemplo, ela está criando um container que abrange todo o corpo do HTML e ele também cria uma divisão para o cabeçalho;
- *Class*: este atributo tem a função de definir a qual classe aquela *tag* específica pertence, podendo ser reutilizada várias vezes;
- *id*: o atributo identificador determina uma referência, ele pode ser utilizado apenas uma vez para cada *tag*;
- Títulos e subtítulos: os títulos e subtítulos são representados pelas *tags* “*<h1>*”, “*<h2>*”, “*<h3>*”, “*<h4>*”, “*<h5>*”, “*<h6>*”, sendo o h1 a maior fonte e o h6, a menor;
- *<p>*: agrupa qualquer tipo de meio visual, como texto e imagens;
- *<form>*: representa uma seção do documento onde existem controles e ações que o usuário pode tomar para interagir com o formulário, neste caso a avaliação;
- *<label>*: cria um pequeno texto para indicar a categoria de um campo de texto, geralmente;
- *For*: o atributo “for” serve para referenciar à *tag* *<input>* qual é a identificação

daquela *tag* `<label>` específica;

- `<input>`: os campos de texto representam espaços a serem preenchidos na interface da página, ela serve para criar os controles interativos dos formulários. Além disso, eles são categorizados através do seu tipo (*type*);
- Tipo: o atributo “type” determina de qual tipo aquela *tag* “`<input>`” pertence, podendo variar entre os tipos email, nome, senha, entre outros. Quando o tipo é definido, a *tag* consegue modificar sua formatação;
- `
`: é utilizado para dividir os conteúdos, como textos, de uma linha para outra;
- `<textarea>`: tem a função de criar na interface da página uma caixa de texto similar ao campo de texto, porém com a finalidade de o usuário escrever um texto corrido;
- *Cols* e *Rows*: estes dois atributos definem a quantidade de colunas e linhas da *tag* “`<textarea>`”;
- `<button>`: é utilizado para criar um botão na interface da página, este botão é capaz de receber os dados digitados pelo usuário dentro da avaliação.

Figura 2 - Resultado do código HTML

A imagem mostra um formulário web simples com o título "Avaliação" em uma fonte serifada e em negrito. Abaixo do título, há uma instrução: "Informe os seus dados para fazer a avaliação". O formulário contém três campos de entrada: um para o nome, um para o e-mail e um campo de texto maior para comentários. Cada campo é precedido por um rótulo ("Nome:", "Email:", "Comentários:"). No canto inferior esquerdo do formulário, há um botão com o texto "Enviar".

Avaliação

Informe os seus dados para fazer a avaliação

Nome:

Email:

Comentários:

Fonte: Do próprio autor, 2024.

2.3. CSS

De acordo com Quierelli (2012), o *Cascading Style Sheet* (CSS), ou traduzindo para o português, Folha de Estilo em Cascata, edita a visualização do conteúdo de páginas *web*, ele adiciona cores ao fundo e estiliza textos, além de formatar imagens.

Como afirma Jobstraibizer (2009), a linguagem de estilo CSS é utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, assim como o HTML.

Dessa forma, Silva (2011) menciona sobre a regra CSS, disposta pelo seletor e sua declaração, e essa é formada por uma propriedade e um valor. Uma regra CSS pode conter várias declarações separadas por ponto e vírgula.

Figura 3 - Exemplo de código em CSS



```

1  /* Elemento: {
2      Propriedade: valor;
3  }
4  */
5
6  button {
7      color: black;
8  }

```

Fonte: Do próprio autor, 2024.

Conforme o exemplo, “*button*” é o elemento ou seletor referenciado e que irá receber a estilização, “*color*” é a propriedade que define a característica do seletor e, no exemplo, tem a função de definir uma cor, e “*black*” é o valor da propriedade, que neste caso é especificado como preto.

Como descreve Eis e Ferreira (2012), o *link* em HTML é um elemento que consegue importar objetos externos de um documento para outro, assim como no exemplo onde ele referencia um arquivo CSS na codificação da linguagem HTML.

Figura 4 - Conexão entre CSS e HTML



```

1  <!DOCTYPE html>
2  <html lang="pt-br">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Login || NavigateBuy</title>
7      <link rel="stylesheet" href="style.css">
8  </head>

```

Fonte: Do próprio autor, 2024.

Neste pequeno trecho da *tag* “<link>”, deve ser repassada a rota do arquivo onde está armazenado o CSS e consequentemente a ligação entre os dois arquivos estará feita.

Figura 5 - Exemplo de código em CSS para a criação de uma estilização

```
1  @charset "UTF-8";
2
3  * {
4      font-family: Arial, Helvetica, sans-serif;
5      margin: 0;
6      padding: 0;
7  }
8
9  body {
10     background-color: #0E023B;
11     display: flex;
12     justify-content: center;
13     align-items: center;
14 }
15
16 .container {
17     background-color: white;
18     color: #0C8249;
19     border-radius: 10px;
20     box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2);
21     padding: 20px;
22     margin: 60px;
23     width: 300px;
24     text-align: center;
25 }
26
27 .container p {
28     color: #0E0238;
29 }
30
31 .header {
32     color: #0E023B;
33 }
34
35 button {
36     background-color: #0C8249;
37     color: white;
38     border: none;
39     border-radius: 5px;
40     padding: 10px 20px;
41 }
```

Fonte: Do próprio autor, 2024.

O código exibido acima resultará em uma interface de uma avaliação simples em HTML e CSS que será representado na próxima figura.

As propriedades utilizadas neste exemplo foram:

- *Font-family*: define a fonte que vai ser utilizada nos textos;
- *Margin*: altera a margem ao redor do elemento;
- *Padding*: introduz o espaço entre o conteúdo da página e as bordas;
- *Display*: determina como os elementos vão ser distribuídos na página;
- *Justify-content*: justifica que todos os itens devem ser distribuídos igualmente;
- *Align-items*: alinha os elementos verticalmente;

- *Background-color*: define a cor que vai ser utilizada no fundo dos elementos;
- *Color*: altera a cor dos textos daquele elemento;
- *Border-Radius*: serve para arredondar as bordas do elemento;
- *Box-shadow*: cria uma sombra ao redor do elemento;
- *Width*: determina o tamanho do elemento na página;
- *Text-Align*: alinha verticalmente os textos deste elemento.

Figura 6 - Resultado do código CSS

A imagem mostra um formulário web com o título "Avaliação" em uma fonte grande e escura. Abaixo do título, há um texto instrutivo: "Informe os seus dados para fazer a avaliação". O formulário contém três campos de entrada: "Nome:" e "Email:" são campos de texto simples, e "Comentários:" é um campo de texto maior com uma borda arredondada. Um botão verde com o texto "Enviar" está localizado na base do formulário. O formulário inteiro está centralizado dentro de um retângulo azul escuro com bordas arredondadas.

Fonte: Do próprio autor, 2024.

2.4. JavaScript

Conforme descreve Flanagan (2013), o JavaScript (JS) é uma linguagem de programação bastante utilizada, pois ela se adapta a diversos meios interpretadores, como computadores e smartphones, possuindo uma vasta biblioteca de utilitários.

Segundo Stefanov (2011), sendo essencial para a interatividade de elementos, o JS opera entre o lado do cliente, a qual é a parte visível na página, e o lado do servidor, responsável pelo processamento de informações de um sistema.

Destacado por Groner (2019), o *GitHub*, o maior host de códigos do mundo, hospeda mais de 400 mil repositórios em JavaScript, podendo assim notar uma das linguagens mais populares e proeminentes do mundo e da internet.

Figura 7 - Código construído com a linguagem JavaScript

```

1  function inserirDados() {
2      const form = document.getElementById("meuForm")
3      const nomeInput = document.getElementById("nome")
4      const emailInput = document.getElementById("email")
5      const comentariosInput = document.getElementById("comentarios")
6
7      form.addEventListener('submit', function(event) {
8          event.preventDefault(); // Evita o envio padrão do formulário
9
10         const nome = nomeInput.value;
11         const email = emailInput.value;
12         const comentarios = comentariosInput.value;
13
14         const mensagemNome = `Nome: ${nome}\nEmail: ${email}\nComentários: ${comentarios}\n`;
15         alert(`${mensagemNome}`);
16     })
17 }
18
19
20 inserirDados();

```

Fonte: Do próprio autor, 2024.

O código exibido acima resultará em uma interface de uma avaliação simples em HTML e CSS, porém a implementação do JavaScript fornece funcionalidade, fazendo com que um alerta apareça assim que o botão “Enviar” seja pressionado, confira nas duas próximas figuras.

As principais partes deste exemplo são:

- Função `inserirDados()`: esta linha cria uma função, chamada de “inserirDados”, capaz de receber dados da avaliação e inseri-los em uma mensagem de alerta. Ela abrange a maior parte da codificação;
- `Const`: é uma declaração que define uma variável com um nome específico no espaço de memória do dispositivo;
- `document.getElementById`: é autoexplicativa pelo seu nome, ela consegue buscar um conteúdo de um elemento HTML através de um identificador;
- `AddEventListener`: adiciona um manipulador de eventos para a avaliação e recebe um objeto como parâmetro, ele é utilizado todas as vezes que o botão “enviar” for pressionado;
- `event.preventDefault()`: serve para evitar o envio de um formulário padrão do navegador, o que iria conflitar com o exemplo do código quando a página fosse recarregada;

- *.value*: a finalidade desta propriedade é obter o valor do elemento HTML e armazenar o conteúdo em uma variável constante;
- *Alert*: antes de ser acionado, cria-se uma linha *string* para formatar as informações que estiverem presentes na última variável constante da codificação. Esta formatação será exibida em um alerta na página.

Figura 8 - Resultado do código com HTML, CSS e JavaScript

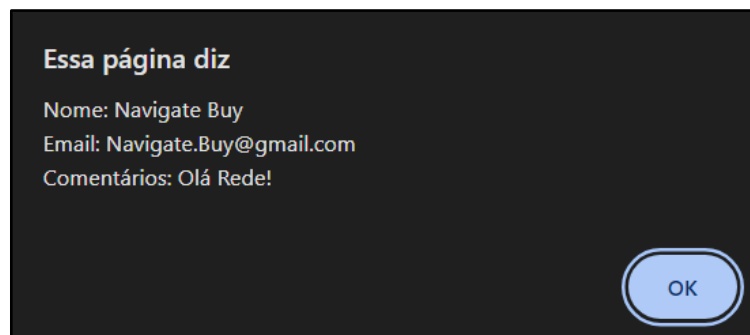


O formulário, intitulado "Avaliação", solicita os dados do usuário para a realização da avaliação. Ele contém três campos de entrada: "Nome" com o valor "Navigate Buy", "Email" com o valor "Navigate.Buy@gmail.com", e "Comentários" com o texto "Olá Rede!". Um botão verde "Enviar" está posicionado na base do formulário.

Fonte: Do próprio autor, 2024.

Assim que o botão “Enviar” é pressionado, o seguinte alerta se apresenta na tela:

Figura 9 – Resultado de caixa de alerta com JavaScript



Fonte: Do próprio autor, 2024.

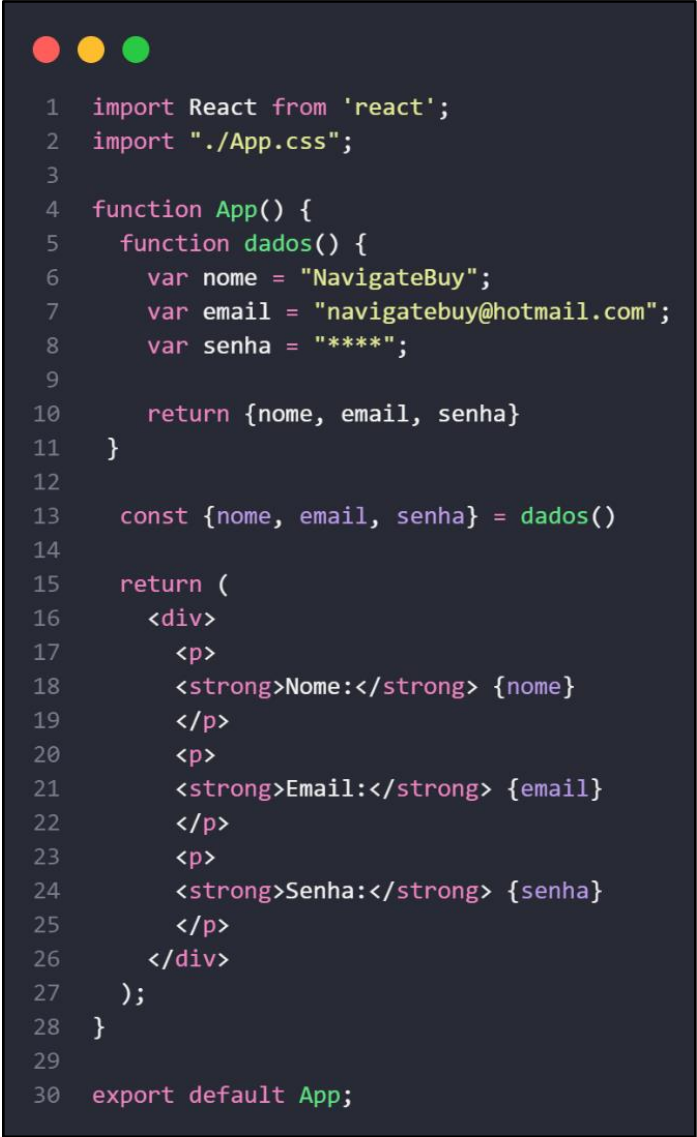
2.5. React

Em concordância com Soares (2023), o React é uma biblioteca herdada do JavaScript, tendo como propósito principal o desenvolvimento *Front-End*, trazendo consigo uma diminuição nas linhas de código em aplicações.

Conforme dito por Silva (2021), a biblioteca React possui uma facilidade e otimização no desenvolvimento de criação de interfaces interativas para o usuário e de alto desempenho.

Segundo Stefanov (2016), não há imposição de uma estrutura de diretórios fixa ao utilizar o React, possuindo a liberdade de utilizar ou renomear um diretório novo em seu desenvolvimento da aplicação em React.

Figura 10 - Exemplo de código para aplicação de um login simples em React



```
1  import React from 'react';
2  import './App.css';
3
4  function App() {
5    function dados() {
6      var nome = "NavigateBuy";
7      var email = "navigatebuy@hotmail.com";
8      var senha = "****";
9
10     return {nome, email, senha}
11   }
12
13   const {nome, email, senha} = dados()
14
15   return (
16     <div>
17       <p>
18         <strong>Nome:</strong> {nome}
19       </p>
20       <p>
21         <strong>Email:</strong> {email}
22       </p>
23       <p>
24         <strong>Senha:</strong> {senha}
25       </p>
26     </div>
27   );
28 }
29
30 export default App;
```

Fonte: Do próprio autor, 2024.

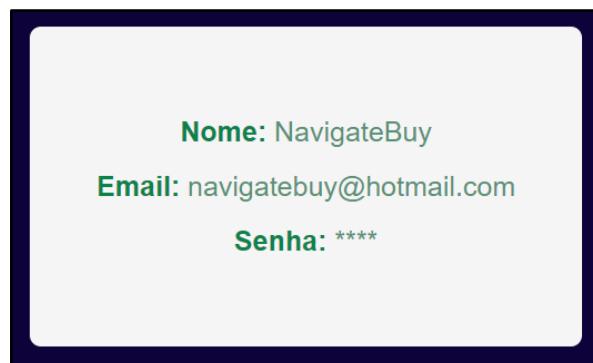
Tendo em vista que o React é uma biblioteca que herda do JavaScript, dessa maneira o React também pode ser trabalhado com o layout da página com o HTML e o CSS. Com base no exemplo, irei dividi-lo, para uma explicação mais objetiva:

- *Function*: está criando duas funções “App” e “dados” ambas sem parâmetro, sendo “dados” uma função para armazenar os dados inseridos, já a função “App” é a função principal do código basicamente tudo que for feito no código tem que estar dentro dessa função;
- *Var*: no exemplo do código, temos as variáveis “nome”, “email” e “senha” onde serão armazenados os dados;
- *Return*: uma função destinada a retornar algum valor quando for chamada, que na situação do exemplo o “return” está voltando 3 variáveis do mesmo tipo sendo elas “String” que quer dizer que são do tipo texto;
- *Const*: no exemplo está sendo usado para chamar as variáveis da função “dados” de forma constante;
- *Export default*: está fazendo a exportação de todo esse conteúdo colocado na função “App”, para a execução do código.

Porém, todo esse código para ter funcionalidade dentro do React são feitas importações para conseguir manusear diversas linguagens de programação em uma só biblioteca JavaScript, que nesse exemplo são:

- *Import React*: está fazendo a importação de um objeto “React” para um módulo chamado “react” que é um pacote que contém a biblioteca React;
- *Import “./App.css”*: uma importação de um arquivo css para poder ter a estilização da aplicação. Entre as aspas é colocada a rota que esse arquivo se encontra.

Figura 11 - Resultado da aplicação de um login simples feito em React



Fonte: Do próprio autor, 2024.


2.6. Python

Conforme afirma Mckinney (2018), Python é uma linguagem de programação usada em funções de *scripts*, conduzindo eficiência com sua estrutura de código simples e tipagem dinâmica.

Ressaltando o que foi dito por Borges (2014), que possui um imenso conjunto de módulos e *frameworks*, tendo uma fácil execução na compilação dos códigos, possuindo recursos de outras linguagens modernas.

De acordo com Menezes (2010), o Python está em crescimento, sendo muito utilizado na construção de *websites* e aplicações *mobile*, podendo tratar a parte do *beck-end*, isto é, nos tratamentos dos dados.

Figura 12 - Exemplo de código para aplicação de cálculo de média em Python



```
1  # Função para calcular a média da nota
2  def Calculando_Media():
3      nota1 = float(input("Digite a nota 1: "))
4      nota2 = float(input("Digite a nota 2: "))
5      nota3 = float(input("Digite a nota 3: "))
6
7      media = (nota1 + nota2 + nota3) / 3
8
9      if media >= 8:
10         print("Você está com a média:", media, "\nAprovado!!")
11     elif media >= 5:
12         print("Você está com a média:", media, "\nRecuperação")
13     else:
14         print("Você está com a média:", media, "\nReprovado")
15
16     return nota1, nota2, nota3
17
18 valores = Calculando_Media()
```

Fonte: Do próprio autor, 2024.

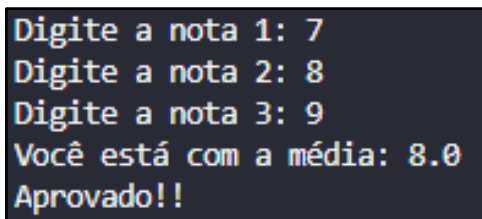
Dado o exemplo acima, ele será desmembrado para que os comandos sejam explicados:

- *Def*: define um objeto de função, seu objetivo é efetuar uma ação que foi designada, ou seja, na situação do exemplo acima, temos uma função que é, “Calculando_Media”;

- *Input*: essa é uma função que faz o recebimento dos valores para as três variáveis para, desse jeito, fazer o cálculo da média das três notas;
- *media*: uma variável que está armazenando o cálculo que será feito para tirar a média das três notas registradas em três variáveis que se chamam “nota1”, “nota2” e “nota3”;
- *If*: uma estrutura de condição para que o código seja mais eficiente podendo ampliar essa condição para outras com o comando “*Elif*”, nessa situação o “*If*” está sendo usado para dar uma condição para o programa que basicamente é, se “media” for igual ou maior que 8 mostrará uma mensagem que foi aprovado, caso contrário entra na estrutura “*Elif*” mostrando uma mensagem de recuperação, e em caso de uma nota muito baixa entra na estrutura “*Else*” informando que foi reprovado;
- *Print*: para imprimir qualquer tipo de dado ou conteúdo, seja de uma variável, até uma complexa função, sendo o comando mais utilizado não só em Python, mas em outras linguagens de programação também, no caso do exemplo acima está printando três mensagens uma de aprovado ou recuperação ou reprovado que aparecera para a pessoa dependendo da estrutura “*If*”;
- *valores*: uma variável criada para receber a função “Calculando_Media” fazendo a sua chamada sem passagem de parâmetro.

Ao rodar a aplicação, a pessoa irá fornecer as notas e assim será feito o cálculo da média entre elas.

Figura 13 - Resultado do código do cálculo da média em Python pelo terminal



```
Digite a nota 1: 7
Digite a nota 2: 8
Digite a nota 3: 9
Você está com a média: 8.0
Aprovado!!
```

Fonte: Do próprio autor, 2024.

2.7.NPM

Conforme define Pereira (2014), o *Node Package Manager* (NPM) significa gerenciador de pacotes do Node.js, como uma ferramenta para gerenciar módulos JS em projetos, facilitando a instalação e atualização das dependências de um módulo.

De acordo com Ihrig (2014), Node.js é um ambiente virtual para criar servidores *web* capazes de impulsionar projetos e aplicações. Sua instalação pode ser feita através do gerenciador de pacotes de um sistema operacional.

Powers (2017) destaca que, com o Node, o NPM vem juntamente instalado e os seus módulos podem ser adquiridos global ou localmente. Sendo capaz de descobrir todas as dependências presentes em um módulo e respectivamente instalar todas.

Assim, parafraseando Moraes (2021), o NPX, uma extensão do NPM, possui a função de executar módulos sem precisar de uma instalação prévia. Ele cria pastas temporárias para adicionar as dependências do pacote e logo após as remove.

Os principais comandos do NPM são:

- `npm install nome_do_módulo`: instala um módulo no projeto;
- `npm install -g nome_do_módulo`: instala um módulo global;
- `npm list`: lista todos os módulos do projeto;
- `npm list -g`: lista todos os módulos globais;
- `npm remove nome_do_módulo`: desinstala um módulo do projeto;
- `npm remove -g nome_do_módulo`: desinstala um módulo global;
- `npm update nome_do_módulo`: atualiza para a versão mais recente de um módulo;
- `npm update -g nome_do_módulo`: atualiza para a versão mais recente de um módulo global;
- `npm -v`: exibe a versão atual do NPM instalada na máquina.

Figura 14 - Exemplo de inicialização de um projeto NPM

```
Microsoft Windows [versão 10.0.22631.3447]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\viniv>mkdir ExemploNpm

C:\Users\viniv>cd ExemploNpm

C:\Users\viniv\ExemploNpm>npm init
```

Fonte: Do próprio autor, 2024.

A imagem acima exemplifica a criação de um projeto em um *prompt* de comando do sistema operacional através do comando:

- `mkdir ExemploNpm`

Após isso, a pasta é acessada com o comando:

- `cd ExemploNpm`

E por fim, o prompt interativo do NPM é inicializado com o comando:

- `npm init`

Figura 15 - Exemplo de inserção dos dados exigidos pelo NPM

```
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help init' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (exemplonpm)
version: (1.0.0)
description: Este projeto é apenas um exemplo
entry point: (index.js)
test command:
git repository:
keywords: exemplo, tcc, projeto
author:
license: (ISC)
```

Fonte: Do próprio autor, 2024.

Após a execução do último comando, uma saída semelhante a essa será exibida no prompt de comando. O NPM solicitará automaticamente algumas informações do responsável pelo projeto, incluindo:

- *name*: nome do projeto;
- *version*: versão inicial do pacote Node.js;
- *description*: descrição escrita do projeto para que outras pessoas o encontrem, caso ele seja publicado na internet;
- *entry point*: arquivo principal que será a porta de entrada do projeto, que por padrão é o “*index.js*”;
- *test command*: comandos pré-definidos para testes feitos pelo NPM;
- *git repository*: se caso houver um repositório criado na plataforma de hospedagem, *github*, o NPM salvará as atualizações do projeto neste repositório automaticamente;
- *keywords*: palavras-chave do projeto para que outras pessoas o encontrem.
- *author*: através de alguns dados, como um *e-mail* e um *site*, é possível determinar um autor para o projeto;
- *license*: define as permissões e limitações legais que outras pessoas vão encontrar ao utilizar o módulo do projeto.

2.8. Web Crawler

Conforme descreve Mitchell (2019), o *Web Crawler*, que traduzido para a língua vernácula significa rastreador da *web* é essencialmente um algoritmo de rastreamento com um elemento central de recursão.

Consoante Souza e Café (2018), esses capturadores são programas que coletam dados de forma contínua de fontes específicas, que podem ser desenvolvidos em Python e os dados armazenados em um banco de dados.

Assim, de acordo com Machado *et al.* (2015), o Web Crawler, também denominado *Spider* ou *Bot*, é um programa empregado para coletar informações disponíveis em páginas da web, como links e palavras-chave.

2.9. Web Scraping

Segundo Farias, Angeluci e Passarelli (2021), o Web Scraping, ou raspagem de dados, é o processo de criação de scripts, geralmente em Python, para extrair dados de forma automatizada e estruturada para análises ou outros fins.

Conforme demonstra Assis e Gomide (2021), essa técnica de mineração de dados online envolve buscar, extrair e organizar dados desestruturados em uma forma estruturada.

Portanto, segundo Rodrigues *et al.* (2021), essa coleta automática de dados pode ser útil para diversas aplicações, como na busca de documentações, relatórios e estatísticas em repositórios digitais.

2.10. Framework

Para Gamma (2008), o *framework* é basicamente um conjunto de ferramentas que os desenvolvedores podem usar para criar e estender suas aplicações de forma mais eficiente.

Conforme expressado por Nascimento, Capanema e Pereira (2019), um *framework* pode acabar fazendo diversos processamentos, possuindo a capacidade de armazenar códigos e gerenciar recursos em uma aplicação.

Assim, de acordo com Bendoraitis e Kronika (2020), um dos aspectos importantes de um *framework* é a sua possibilidade de integrar sistemas que possam auxiliar nos projetos.

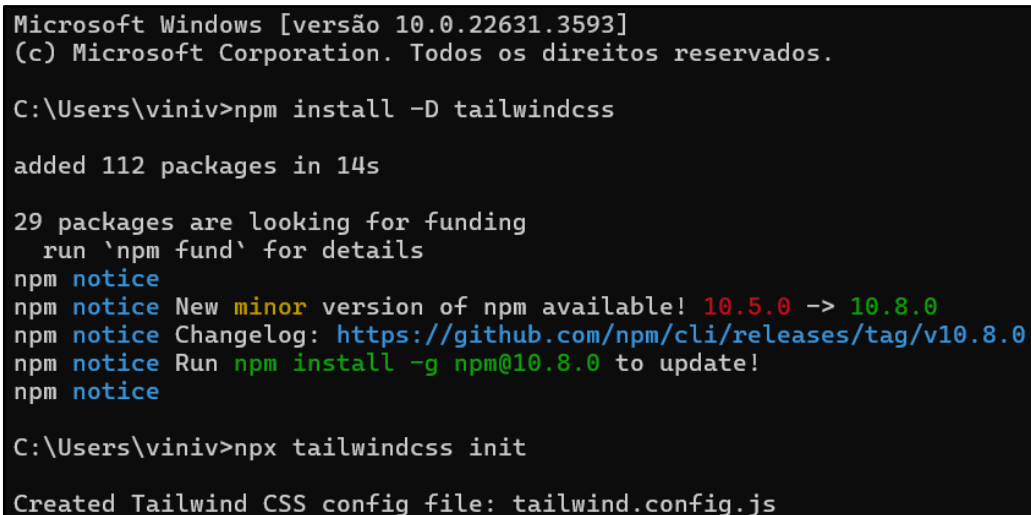
2.10.1. Tailwind

Conforme descreve Abba (2023), o Tailwind é um *framework* CSS que simplifica a estilização de páginas web, possibilitando a escrita direta de CSS no HTML, por meio de suas classes predefinidas.

De acordo com Oliveira (2023), é possível criar componentes independentes com configurações que o próprio *framework* oferece, tornando a leitura do código mais "limpa" e de fácil compreensão para o desenvolvedor.

Portanto, Neves (2023) ressalta que a sua abordagem baseada em componentes otimiza o tempo e esforço do programador, já que poucas linhas de código precisam ser escritas para construir um *design* moderno.

Figura 16 - Exemplo de instalação do Tailwind



```
Microsoft Windows [versão 10.0.22631.3593]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\viniv>npm install -D tailwindcss

added 112 packages in 14s

29 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New minor version of npm available! 10.5.0 -> 10.8.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.0
npm notice Run `npm install -g npm@10.8.0` to update!
npm notice

C:\Users\viniv>npx tailwindcss init

Created Tailwind CSS config file: tailwind.config.js
```

Fonte: Do próprio autor, 2024.

No Terminal, deve-se imprimir o seguinte comando:

- `npm install -D tailwindcss`

E logo após:

- `npx tailwindcss init`

Depois da instalação do Tailwind, será gerado um arquivo denominado de "tailwind.config.js", nele deve ser inserido o caminho do seu arquivo HTML, como mostrado no exemplo:

Figura 17 - Exemplo de criação da conexão entre HTML e o Tailwind

```

1  /** @type {import('tailwindcss').Config} */
2  module.exports = {
3    content: ["./Elementos/HTML/Exemplo.html"],
4    theme: {
5      extend: {},
6    },
7    plugins: [],
8  }

```

Fonte: Do próprio autor, 2024.

Agora, com a criação de dois arquivos em CSS, um principal e um utilitário, deve ser adicionado os diretivos do Tailwind no arquivo utilitário com os seguintes comandos:

Figura 18 - Acrescentando diretivos do Tailwind

```

1  @tailwind base;
2  @tailwind components;
3  @tailwind utilities;

```

Fonte: Do próprio autor, 2024.

Depois disso, deve existir uma conexão entre os dois arquivos CSS em seus respectivos caminhos com este comando no terminal:

- `npx tailwindcss -i ./src/utilitário.css -o ./src/principal.css --watch`

Figura 19 – Exemplo executando comando Tailwind

```

PS C:\Users\viniv\TCC> npx tailwindcss -i ./Elementos/CSS/utilitario.css -o ./Elementos/CSS/style.css --watch
Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
Rebuilding...
Done in 170ms.

```

Fonte: Do próprio autor, 2024.

Por fim, um link entre o CSS principal e o arquivo HTML deve ser feito e que será mostrado na próxima figura, na linha 7:

Figura 20 - Código HTML com a implementação do Tailwind

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Login || NavigateBuy</title>
7   <link rel="stylesheet" href="style.css"> </head>
8 <body class="bg-azul flex justify-center">
9   <div class="container bg-white rounded-lg text-verde p-5 m-6 w-96">
10     <div class="header text-azul text-4xl text-center">
11       <h1>Login</h1>
12     </div>
13     <p class="text-azul text-lg underline text-center m-4">Informe seus dados!</p>
14     <form id="meuForm" class="flex flex-col items-center m-4">
15       <label for="nome" class="mt-4">Email:</label>
16       <input type="email" id="Email" name="Email" class="py-1.5 pl-1 bg-gray-300 border-2 border-verde">
17       <label for="senha" class="mt-4">Senha:</label>
18       <input type="password" name="senha" id="senha" class="py-1.5 pl-1 bg-gray-300 border-2 border-verde">
19       <button type="submit" class="bg-verde text-white rounded-md p-3 mt-4">Entrar</button>
20     </form>
21   </div>
22 </body>
23 </html>

```

Fonte: Do próprio autor, 2024.

O código exibido acima resultará em uma interface de um login simples em HTML e Tailwind CSS que será representado na próxima figura.

As principais propriedades de Tailwind neste exemplo são:

- “bg-azul” e “bg-white”: conseguem definir qual cor vai ser utilizada no fundo do elemento;
- “flex” e “justify-center”: são capazes de alinhar o conteúdo da página ao centro da tela horizontalmente de maneira flexível;
- *rounded-lg*: compõe as bordas arredondadas ao elemento em um tamanho pré-definido pelo Tailwind;
- text-verde: é possível modificar a cor que o texto irá ser escrito na tela pela propriedade “text-(cor)”. E no exemplo, foi definida a cor verde;
- “m-6” e “p-5”: são responsáveis pelas margens e preenchimento do corpo visual por um tamanho pré-definido pelo Tailwind, enquanto “mt-4” aplica uma margem para o elemento em relação aos elementos acima dele. Já “py” e “pl” são duas versões da propriedade para preenchimento, o primeiro modifica a área ocupada nos lados inferior e superior do elemento, já o segundo integra o lado esquerdo do elemento;
- “w-96”: consegue alterar o tamanho, pré-definido pelo Tailwind, da tamanho

do elemento na tela;

- “*text-4x1*”: aumenta o tamanho do texto 4 vezes maior do que o padrão;
- “*text-center*”: organiza o texto de maneira centralizada;
- “*underline*”: impõe uma linha abaixo do texto, sublinhando a mesma;
- “*flex-col*”: faz com que os elementos contidos nele se organizem de maneira flexível em coluna;
- “*items-center*”: permite que os elementos se organizem verticalmente dentro de um container;
- “*border-2*” e “*border-verde*”: criam uma borda e dão uma cor para ela, com um tamanho pré-definido pelo Tailwind.

Figura 21 - Resultado do código HTML com a implementação do Tailwind



Fonte: Do próprio autor, 2024.

2.10.2. Flask

Segundo Grinberg (2018), o Flask é um *microframework* escrito em Python, porém ressalta que mesmo sendo um *framework* pequeno, não quer dizer que possua menos recursos que outros *frameworks*. Levando em conta a facilidade de entender o código.

De acordo com Silva (2019), como uma ferramenta auxiliar na criação de um sistema web, o Flask traz grandes benefícios quando utilizado, sendo um dos melhores frameworks para se trabalhar com Python.

Consoante Pereira (2018), por ser considerado um *microframework*, o Flask é ideal para desenvolver ferramentas que apoiam aplicações, oferecendo a flexibilidade necessária para adaptar-se a diferentes necessidades e escalas de projetos.

Figura 22 - Exemplo de código para aplicação de cálculo de média com o Flask



```
1  from flask import *
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def home():
7      return render_template('Exemplo_para_TCC.html')
8
9  @app.route('/Exemplo_para_TCC.html', methods=['POST'])
10 def Calcular():
11     nota1 = float(request.form.get('nota1'))
12     nota2 = float(request.form.get('nota2'))
13     nota3 = float(request.form.get('nota3'))
14     media = (nota1 + nota2 + nota3) / 3
15
16     if media >= 8:
17         print("Sua média é:", media, "\nAprovado!")
18     elif media >= 5:
19         print("Sua média é:", media, "\nRecuperação")
20     else:
21         print("Sua média é:", media, "\nReprovado")
22
23     return redirect('/')
24
25 if __name__ in "__main__":
26     app.run(debug=True)
```

Fonte: Do próprio autor, 2024.

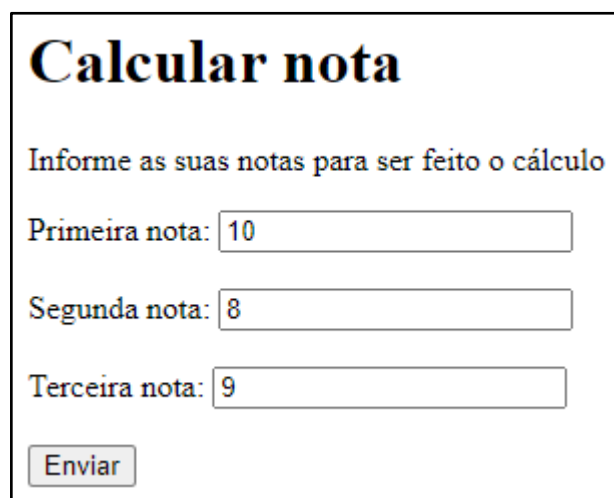
Dado o exemplo acima, ele será desmembrado para que os comandos sejam explicados:

- *From*: o “*From*” está indicando o *framework* que será importado;
- *Import*: um comando para importar as bibliotecas e *frameworks*, logo em seguida tem um caractere que é o asterisco “*” que tem uma função de informar para o comando “*Import*” que é para importar tudo do módulo Flask;

- *App.route*: utilizado para fazer a passagem de parâmetro através da rota que for fornecida. No caso do exemplo, é o “Exemplo_para_TCC.html” que, por meio do método “*POST*”, passará a capturar as variáveis “nota1”, “nota2” e “nota3”, que o valor será colocado pelo usuário;
- *Render_template*: é utilizado para renderizar o modelo HTML para *Hypertext Transfer Protocol* (HTTP), que no caso do código é o “Exemplo_para_TCC.html”;
- *Methods*: determina o tipo de método HTTP que será aplicado no código, na situação do exemplo é o método “*POST*”;
- *Request*: é um objeto fornecido pelo Flask que inclui os dados das notas enviados pelo usuário;
- *Redirect*: simplesmente é uma função de redirecionamento de páginas, especificada pela *Uniform Resource Locator* (URL), que no caso do código é a barra ‘/’;
- *App.run*: determina que quando a verificação for feita que o código está sendo executado direito, será realizado o método “*run*” no objeto “app”, que é uma instância do Flask, já o “debug=True” é opcional já que ele detalha sobre os erros e recarrega automaticamente a aplicação quando o código for alterado.

Para ser inserido os dados das notas, o usuário digitará na página web criada com HTML:

Figura 23 - Tela de inserimento dos dados em HTML

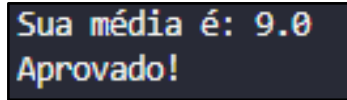


A imagem mostra uma interface web com o título "Calcular nota" em uma fonte serifada e em negrito. Abaixo do título, há uma instrução: "Informe as suas notas para ser feito o cálculo". Seguem três linhas de entrada de texto, cada uma com um rótulo à esquerda e um campo de entrada à direita: "Primeira nota:" com o valor "10", "Segunda nota:" com o valor "8", e "Terceira nota:" com o valor "9". No canto inferior esquerdo da interface, há um botão com o texto "Enviar".

Fonte: Do próprio autor, 2024.

Com o desenvolvimento do Flask, quando os dados forem inseridos, o resultado da conta irá aparecer no terminal.

Figura 24 - Resultado da codificação com o Flask no terminal



Sua média é: 9.0
Aprovado!

Fonte: Do próprio autor, 2024.

2.10.3. Scrapy

De acordo com Duke (2018), Scrapy é um *framework* que extrai um conjunto de informações como textos e números para manipular e tratar os mesmos. Tudo que estiver disponível na internet pode ser obtido.

Conforme expressado por Gomes (2024), o processo de extração acontece de forma automatizada, e o framework utiliza a técnica Web Scraping juntamente ao Python, por ser uma linguagem de programação de fácil uso.

E para finalizar, Didática Tech (2022) nos mostra que as vantagens de utilizar o Scrapy são sua eficiência para lidar com abundância de coletas de dados e a fácil manipulação da codificação por um desenvolvedor.

Figura 25 - Exemplo de codificação em Python com o Scrapy



```

1  from scrapy.spiders import CrawlSpider, Rule
2  from scrapy.linkextractors import LinkExtractor
3
4  class MinhaaranhaSpider(CrawlSpider):
5      name = "Minhaaranha"
6      allowed_domains = ["books.toscrape.com"]
7      start_urls = ["https://books.toscrape.com"]
8
9      rules = (
10         Rule(LinkExtractor(allow="catalogue/category")),
11         Rule(LinkExtractor(allow="catalogue", deny="category"), callback="parse_item")
12     )
13
14     def parse_item(self, response):
15         title = response.css(".product_pod h3 a::attr(title)").get()
16         price = response.css(".price_color::text").get()
17
18         yield {
19             'title': title,
20             'price': price,
21         }

```

Fonte: Do próprio autor, 2024.

O código acima demonstra um exemplo de codificação escrita na linguagem Python, onde podem ser obtidas informações de uma página através da técnica de *web scraping*.

As principais partes desta codificação são:

- *Import*: com este comando é possível importar classes do Scrapy, e no exemplo estão sendo importados o *CrawlSpider* responsável pelas páginas *web* que terão seus dados obtidos através do *crawl*, o *Rule* que ajuda a definir limites por onde o *spider* deve percorrer e o *linkextractors* que permite especificar critérios para a extração de informações da página *web*;
- *Class*: determina o nome da classe que recebe o *CrawlSpider*;
- *Name*: identifica um nome para o *spider*;
- *Allowed_domains*: restringe o *spider* para realizar o *crawl* apenas nas páginas da *web* permitidas;
- *Start_urls*: especifica quais são as URLs das páginas *web* que o *spider* deve chegar para vasculhar as informações contidas nele;
- *Rules*: neste trecho, são armazenadas as regras como caminhos por onde o *spider* vai passar e filtrar os links que forem identificados e permitidos a ele através da URL especificada anteriormente;
- *Parse_item*: esta função tem o objetivo de processar o conteúdo da página *web* e explorar tudo que foi permitido pela última regra definida, logo extraíndo as informações dele. No exemplo, está sendo usado o método *css* do objeto *response*, que recebe o conteúdo da página *web*, para selecionar os títulos e preços dos livros e o método “*get()*” extrai os valores dos elementos selecionados pelo *css*;
- *Yield*: na última parte do código, a função *parse_item* cria um dicionário que irá receber tudo que foi extraído para ser armazenado e visualizado posteriormente.

Figura 26 - Resultado da codificação com o Scrapy



```

1  [
2  {"title": "In Her Wake", "price": "£12.84"},
3  {"title": "I Am Pilgrim (Pilgrim #1)", "price": "£27.09"},
4  {"title": "My Mrs. Brown", "price": "£28.90"},
5  {"title": "Mr. Mercedes (Bill Hodges Trilogy #1)", "price": "£10.60"},
6  {"title": "The Edge of Reason (Bridget Jones #2)", "price": "£29.82"},
7  {"title": "The Lonely Ones", "price": "£26.33"},
8  {"title": "The Thing About Jellyfish", "price": "£43.59"},
9  {"title": "The Wild Robot", "price": "£48.77"},
10 {"title": "The Whale", "price": "£52.87"},

```

Fonte: Do próprio autor, 2024.

2.11. APIs

Segundo dito por Muniz *et al.* (2023), uma *Application Program Interface* (API) possui uma execução de um sistema de forma padronizada e possibilita que duas partes de software se comuniquem.

Conforme mostrado por Stevens (2008), as APIs podem ser utilizadas em diversos ramos na tecnologia, seja para o desenvolvimento de uma aplicação até a construção de um sistema de rede.

Consoante Silva (2009), elas providenciam diversas formas de registros e finalidades, desempenhando um papel fundamental na interconexão e na integração de algum sistema.

2.11.1. AwesomeAPI

Segundo Butewicz (2022), é possível criar um sistema de conversões monetárias com a linguagem de programação Python e a AwesomeAPI. Ele destaca que usar essa API é tão simples quanto qualquer outra biblioteca do Python.

Figura 27 - Exemplo de código com a AwesomeAPI



```

1  import requests
2  import json
3
4  link = 'https://economia.awesomeapi.com.br/last/USD-BRL'
5
6  req = requests.get(link)
7
8  print("Conversão de Dolar americano para Real brasileiro: \n", json.dumps(req.json(), indent=4))

```

Fonte: Do próprio autor, 2024.

Dado o exemplo, será separado em partes para uma explicação mais aprofundada:

- *Requests*: está importando pelo meio “*import*” o *requests* que é uma biblioteca do Python muito utilizada para fazer requisições com API;
- *Json*: outra importação que está sendo feita é do módulo *Json* para manipular os dados que a API fornece;
- *Link*: é uma variável que está armazenando o *link* da API de conversão monetária, quando podemos perceber que no *link* está digitado “USD-BRL” que significa que a API irá em busca da conversão de moedas do dólar americano para o real brasileiro;
- *req*: outra variável, mas esta variável está guardando a biblioteca *requests* que, usando o método “*get*” pegará a conversão que a API irá fornecer;
- *Dumps*: é uma função do módulo “*Json*” que converte um objeto Python em uma *String* *Json* formatada;
- *Json*: com esse método, está convertendo os dados que API está mandando para um formato *Json* e um objeto Python no do comando “*req.json()*”;
- *Indent*: está especificando que a saída *Json* deve ser estilizada com uma indentação de 4 espaços para tornar a visualização do resultado mais legível.

Figura 28 - Resultado da aplicação com a AwesomeAPI

```
{
  "USDBRL": {
    "code": "USD",
    "codein": "BRL",
    "name": "Dólar Americano/Real Brasileiro",
    "high": "5.1451",
    "low": "5.1304",
    "varBid": "0",
    "pctChange": "0",
    "bid": "5.144",
    "ask": "5.1445",
    "timestamp": "1716555589",
    "create_date": "2024-05-24 09:59:49"
  }
}
```

Fonte: Do próprio autor, 2024.

No resultado, é mostrada a conversão do dólar americano para o real brasileiro, informa a quantia de ambos, além de mostrar a data e a hora em que foi feita essa execução.

2.12. Banco de Dados

Segundo Alves (2021), desde os primórdios, o homem sentiu a necessidade de armazenar informações, assim, com o crescimento do meio computadorizado essa necessidade resultou na criação de bancos de forma bem mais prática e rápida.

Portanto, conforme descreve Zhao (2023), o banco de dados é um conjunto de dados organizados, que tem como objetivo armazenar informações em um sistema computacional.

De acordo com Date (2004), um Sistema Gerenciador de Banco de Dados (SGBD) é usado para gerenciar um ou mais bancos de dados. Ele permite a modelagem de dados, tornando os SGBDs cruciais para a organização e manipulação dos dados.

2.12.1. Modelagem de Dados

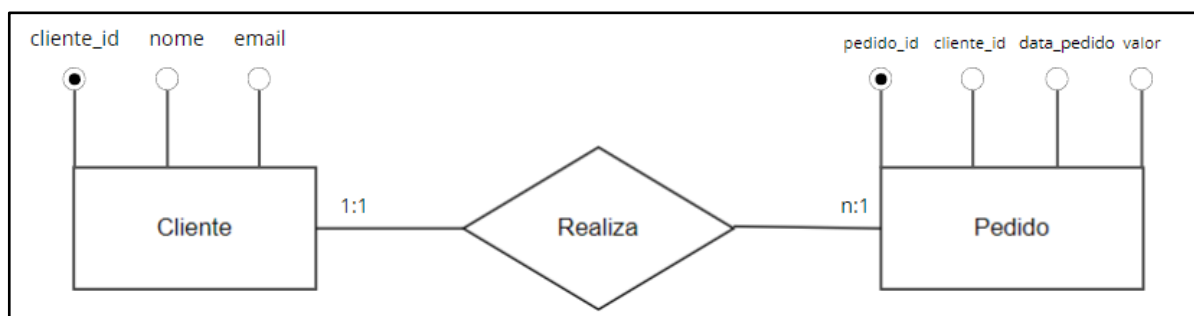
Segundo Heuser (2009), a modelagem de dados é um processo que busca descrever de maneira abstrata os dados a serem armazenados, facilitando a compreensão e interpretação.

Consoante Machado (2020), construir uma modelagem de dados corretamente requer uma compreensão essencial das operações lógicas e da organização dos dados nos quais serão armazenados.

DER

De acordo com Cardoso e Cardoso (2012), o Diagrama Entidade Relacionamento (DER) é o primeiro passo na representação visual de um banco de dados, enfatizando sua importância para evitar erros em projetos de banco de dados.

Figura 29 - Exemplo da criação de um DER



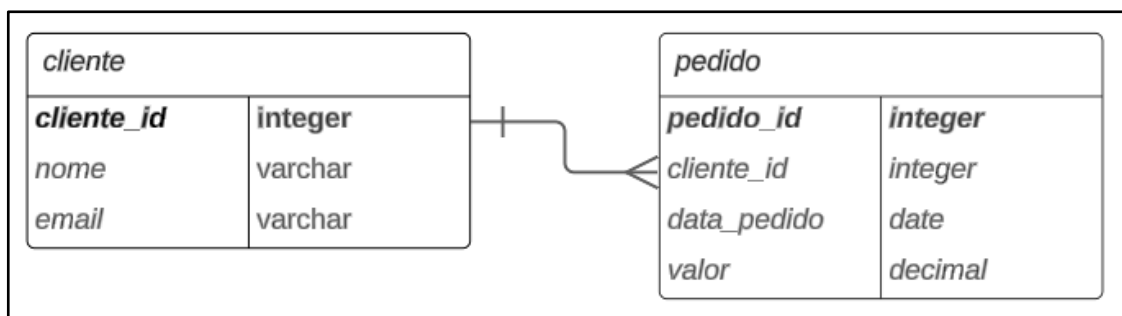
Fonte: Do próprio autor, 2024.

Na imagem acima, é apresentado um exemplo simplificado de um DER, onde podemos visualizar os dados de um banco de forma clara. Nele, são destacadas as entidades "cliente" e "pedido", juntamente com seus atributos correspondentes.

MER

Conforme demonstra Sordi (2019), o Modelo Entidade Relacionamento (MER) tem como base a representação dos dados em um sistema, proporcionando uma visão clara das entidades e de seus respectivos atributos e relacionamentos.

Figura 30 - Exemplo da criação de um MER



Fonte: Do próprio autor, 2024

A imagem acima destaca um exemplo de MER, onde inclui duas entidades, que são: “cliente” e “pedido”, demonstrando cada entidade com seus atributos específicos, como nome e email para cliente, e campos adicionais como pedido_id, data_pedido e valor para pedido. O relacionamento é estabelecido através do campo cliente_id, indicando que cada pedido está associado a um cliente específico.

2.12.2. MySQL

Conforme caracteriza Milani (2006), O MySQL é um SGBD, que supre de maneira vantajosa o armazenamento de dados e as informações de um meio que hoje está cada vez mais popularizado.

De acordo com Lobo (2008), há vários bancos de dados disponíveis, mas é essencial escolher o software apropriado para o sistema no qual será desenvolvido. O MySQL é uma opção competitiva, robusta e vantajosa em termos de licenciamento.

Dessa forma, segundo Alves (2017), esse gerenciador de banco de dados relacional e de código aberto é considerado muito recorrente entre os desenvolvedores para aplicações *web*.

Figura 31 - Exemplo de um código de banco de dados MySQL

```

1  -- Criação do Banco de dados
2  Create Database Exemplo_TCC;
3  Use Exemplo_TCC;
4
5  -- Criação da tabela clientes
6  Create Table clientes (
7      id INT AUTO_INCREMENT PRIMARY KEY,
8      nome VARCHAR(100),
9      email VARCHAR(100)
10 );
11
12 -- Inserção dos clientes na respectiva tabela
13 Insert Into clientes (nome, email) Values ('Navigate Buy', 'navigatebuy@hotmail.com');
14
15 -- Seleção de todos os clientes (caso queira ver todos os que foram registrados)
16 Select * From clientes;
17
18 -- Exemplo de Atualização do email do cliente (caso queira mudar algum dado específico que já foi registrado)
19 Update clientes Set email = 'navigatebuy_novo@hotmail.com' Where nome = 'Navigate Buy';
20
21 -- Exemplo de Seleção do cliente (caso queira ver um dado específico)
22 Select * From clientes Where nome = 'Navigate Buy';
23
24 -- Exemplo de Exclusão do cliente na tabela (caso queira excluir um dado em específico)
25 Delete From clientes Where nome = 'Navigate Buy';

```

Fonte: Do próprio autor, 2024.

Na imagem acima é configurado um código de criação de um banco de dados no MySQL e sua respectiva tabela como exemplo, logo em seguida são apresentados os comandos principais de uma aplicação MySQL, que são:

- *Create*: em português significa criar, seu uso é feito para a criação de um banco que é acompanhado do *Database* e para a criação de uma tabela que é acompanhado do *Table*.
- *Insert*: traduzindo para a língua nacional significa inserir, é usado para inserir dados em uma tabela, junto ao comando *Into*, que vem antes do nome da tabela e o *Values* que passa os valores que vão ser inseridos na tabela.
- *Select*: na língua vernácula quer dizer selecionar, é usado para verificar vários ou um dado que é acompanhado do *From* que determina em qual tabela ele irá selecionar e do *Where* caso queira selecionar de alguma informação específica.
- *Update*: é usado para atualizar um dado que já foi registrado anteriormente é seguido pelo *set* que determina o que vai ser mudado e pelo *Where* que define de qual lugar é esse dado que vai ser transformado.
- *Delete*: transpondo para o português significa deletar, empregado para excluir dados que foram registrados anteriormente.

Figura 32 - Resultado de uma consulta de uma tabela MySQL

id	nome	email
abc Filter...	abc Filter...	abc Filter...
1	Navigate Buy	navigatebuy@hotmail.com

Fonte: Do próprio autor, 2024.

2.13. UX

Conforme descreve Furtado (2023), *User Experience* (UX), que traduzindo significa Experiência do Usuário, refere-se à toda usabilidade que engloba as reações e ações de uma pessoa ao experimentar ou interagir com um produto.

Parafraseando Grant (2019), é evidente que os profissionais dessa área detêm percepções valiosas por meio de estudos e práticas, contudo, é necessário empatia e objetividade como principais habilidades para uma boa criação UX.

Segundo Podmajersky (2019), existe a redação UX, a qual é empregada como diálogo entre a experiência por meio de elementos visuais e palavras e o usuário, que responde interagindo com esses recursos.

Assim, de acordo com Levy (2021), quando implementada uma estratégia de UX adequada, isso traz com que possamos revolucionar o mercado com um produto que seja único ou até uma melhor alternativa dos que existem, por meio da inovação.

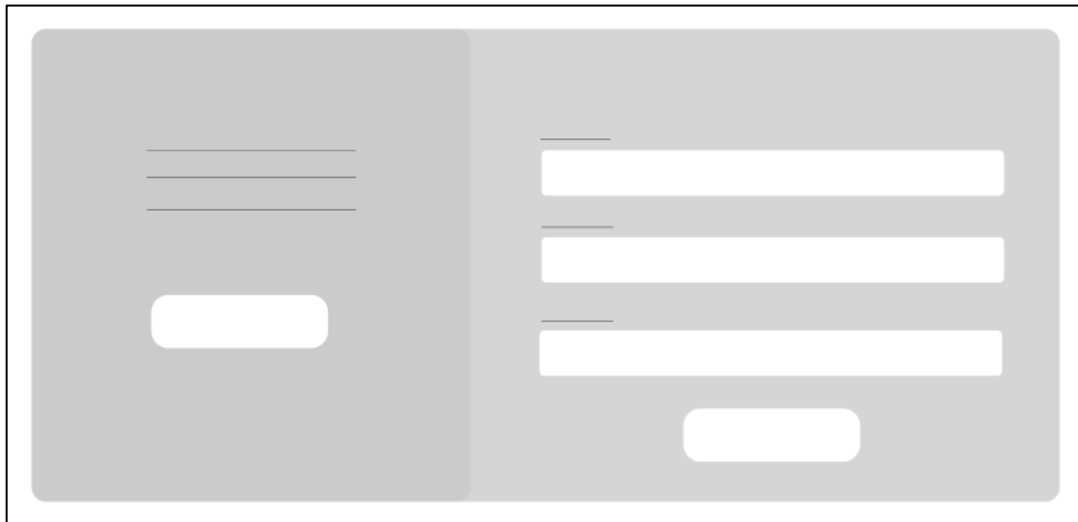
2.14. Wireframe

De acordo com Teixeira (2014), o *Wireframe* é crucial para o desenvolvimento de uma aplicação, pois desenha protótipos que representam como uma interface visual vai ser e ajuda na organização do conteúdo dentro do projeto.

Conforme dito por Memória (2006), o desenvolvimento de um *Wireframe* sempre começa com baixa fidelidade e vai ganhando forma conforme a proposta avança. Dessa maneira, os detalhes da interface vão se tornando cada vez mais refinados.

Em concordância com Pereira (2018), acrescenta e traz que tal é um esboço mais detalhado, possuindo um propósito de comunicar as melhores decisões de uma interface de alguma aplicação, seja *web* ou *mobile*.

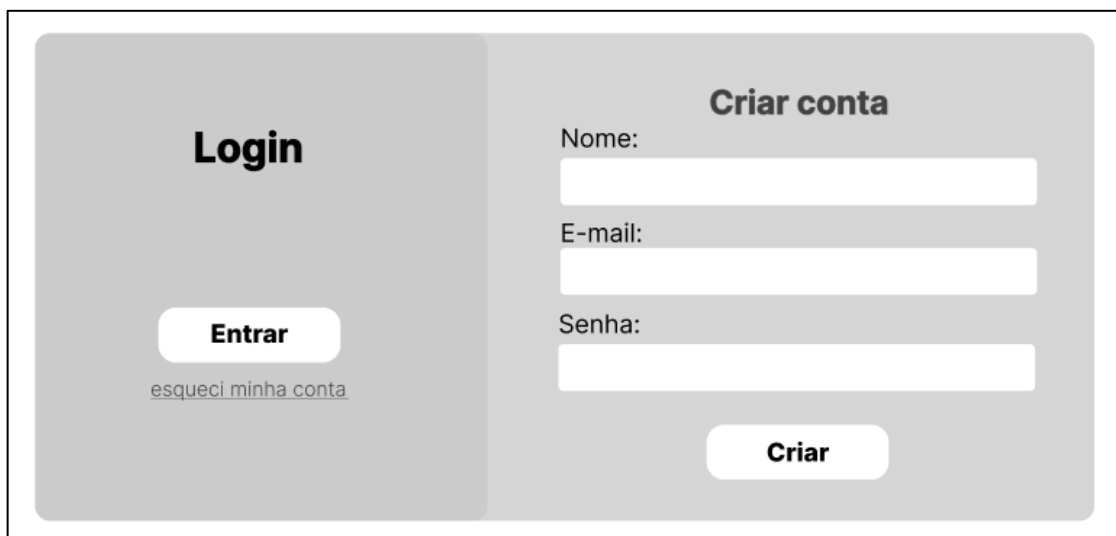
Figura 33 - Exemplo de Wireframe de baixa fidelidade.



Fonte: Do próprio autor, 2024.

Na figura anterior temos o wireframe de baixa fidelidade onde foi colocado o escopo visual inicial de uma página web. Já a seguir é um *wireframe* de média fidelidade, possuindo mais informações do que o de baixa fidelidade.

Figura 34 - Exemplo de Wireframe de média fidelidade.



Fonte: Do próprio autor, 2024.

Na próxima imagem temos o *wireframe* de alta fidelidade mais detalhado que reflete o escopo final do visual da respectiva página web, possuindo cores, textos mais trabalhados e entre outros elementos.

Figura 35 - Exemplo de Wireframe de alta fidelidade.

O wireframe apresenta duas colunas. A coluna da esquerda, com fundo escuro, contém o título 'Login' em verde, o texto 'Bem vindo!', o link 'Acesse sua conta', um botão verde 'Entrar' e o link 'esqueci minha conta'. A coluna da direita, com fundo verde, contém o título 'Criar conta' em escuro, três campos de entrada brancos rotulados 'Nome:', 'E-mail:' e 'Senha:' com exemplos de texto ('user...', 'user@...', 'user123...'), e um botão escuro 'Criar' na base.

Fonte: Do próprio autor, 2024.

2.15. UML

De acordo com Guedes (2018), A *Unified Modeling Language* (UML) é uma linguagem visual e de modelagem baseada no paradigma de orientação a objetos, composta por diagramas com o objetivo de definir características de um sistema a ser aplicado.

Conforme afirmam Pressman e Maxim (2021), essa linguagem se tornou o padrão internacional para a criação de *software* de todos os tipos. Assim, compreender seus elementos permite especificar e entender um sistema de forma fácil.

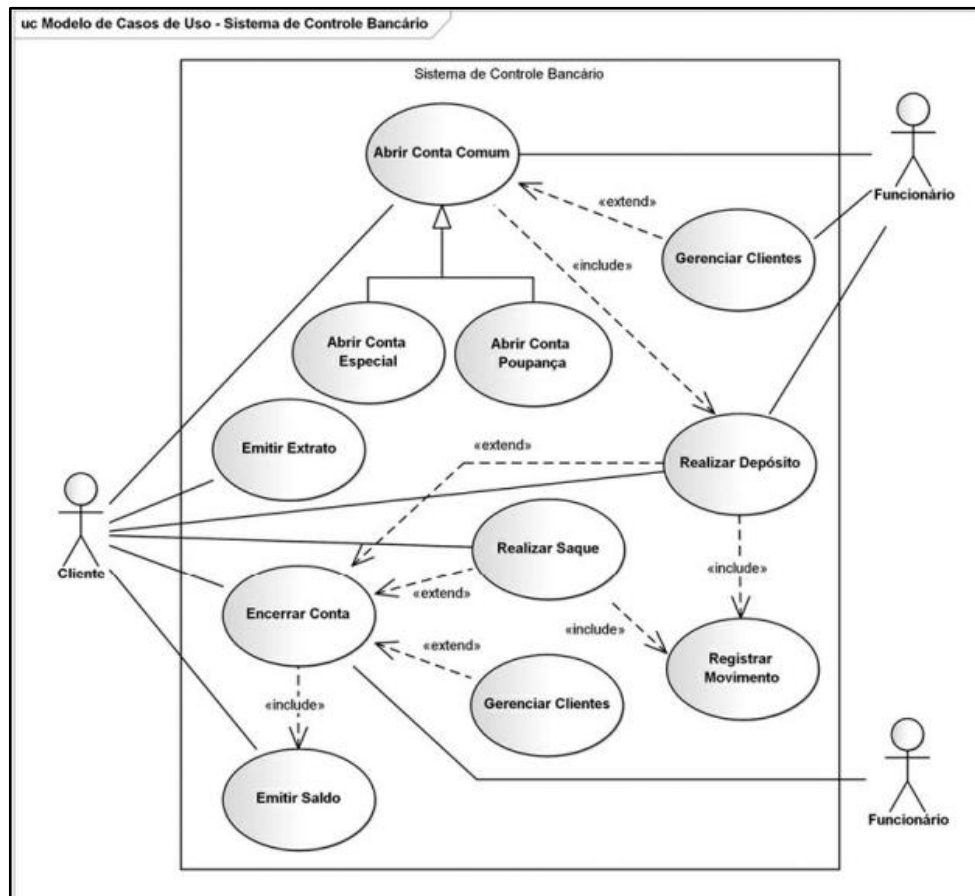
Segundo Fowler (2005), ao usar a UML, as equipes de desenvolvimento podem melhorar a comunicação e a compreensão entre os membros, tornando o processo de desenvolvimento mais eficiente.

Portanto, consoante Booch, Rumbaugh e Jacobson (2006), a UML é amplamente utilizada em diversos contextos, abrangendo itens estruturais, comportamentais, de agrupamento, descritivos, relacionamentos e diagramas.

Diagrama de Casos de Uso

Conforme descreve Fowler (2005), os diagramas de caso de uso são representações gráficas que têm por objetivo fornecer uma visão das funcionalidades de um sistema, identificando os atores e os serviços que ele oferece.

Figura 36 - Exemplo de Diagrama de Casos de Uso



Fonte: (Guedes, 2018)

De acordo com Guedes (2018), esse é um exemplo de diagrama de casos de uso de um sistema de controle bancário, que define as funcionalidades, bem como seus atores caracterizando interações como abertura e encerramento de conta.

Documentação de Caso de Uso

Segundo Pressman e Maxim (2021), as documentações de caso de uso são usadas para descrições formais de um caso de uso específico, que inclui elementos como atores, resumo do escopo, pré-condição, cenário e exceções.

Figura 37 - Exemplo de documentação de Caso de Uso

Nome do Caso de Uso		UC06 – Emitir Saldo	
Ator Principal	Cliente		
Atores Secundários			
Resumo	Descreve os passos necessários para um cliente obter o saldo referente a uma determinada conta		
Pré-condições			
Pós-condições			
Cenário Principal			
Ações do Ator		Ações do Sistema	
1. Informar o número da conta		2. Verificar a existência da conta	
		3. Solicitar a senha da conta	
4. Informar a senha		5. Verificar se a senha está correta	
		6. Emitir o saldo	
Restrições/Validações	1. A conta precisa existir e estar ativa		
	2. A senha precisa estar correta		
Cenário de Exceção I – Conta não encontrada			
Ações do Ator		Ações do Sistema	
		1. Comunicar ao cliente que o número da conta informada não foi encontrado	
Cenário de Exceção II – Senha inválida			
Ações do Ator		Ações do Sistema	
		1. Comunicar ao cliente que a senha fornecida não combina com a senha da conta	

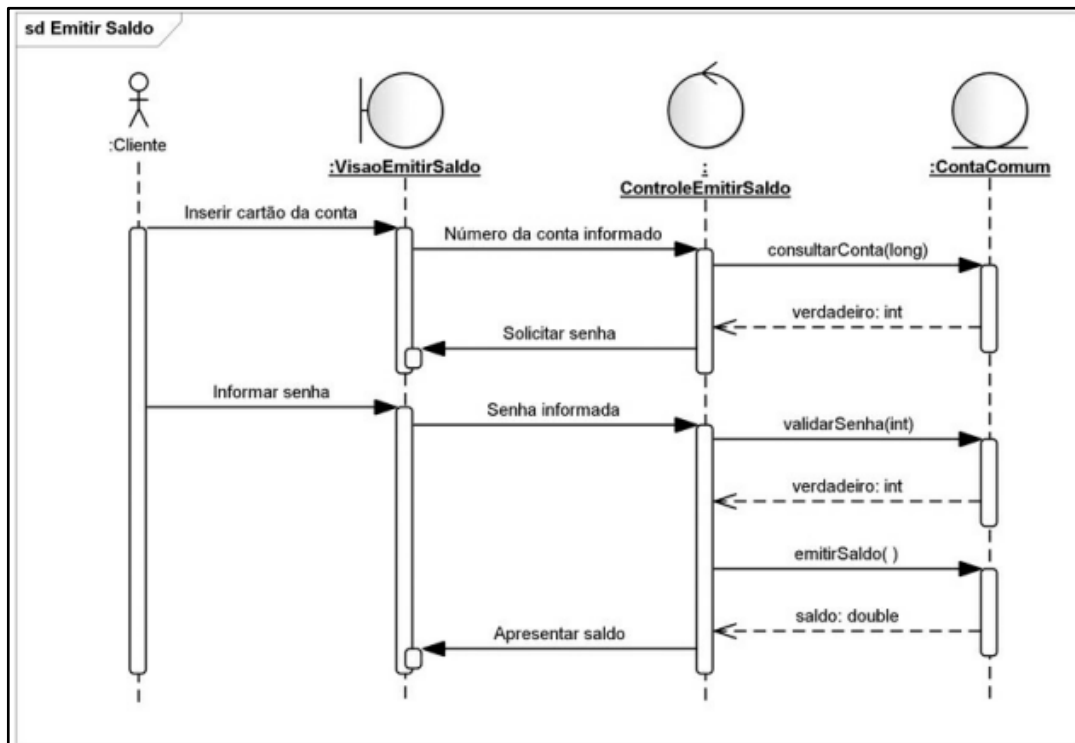
Fonte: (Guedes, 2018)

Consoante Guedes (2018), a documentação de caso de uso do processo de emissão de saldo descreve o resumo, as ações do sistema, exceções e os atores que interagem com esse caso de uso específico.

Diagrama de Sequência

De acordo com Booch, Rumbaugh e Jacobson (2006), um diagrama de sequência foca na ordem temporal das mensagens entre papéis. Ele exhibe os papéis e as mensagens trocadas entre suas instâncias, destacando a visão dinâmica do sistema.

Figura 38 - Exemplo de Diagrama de Sequência



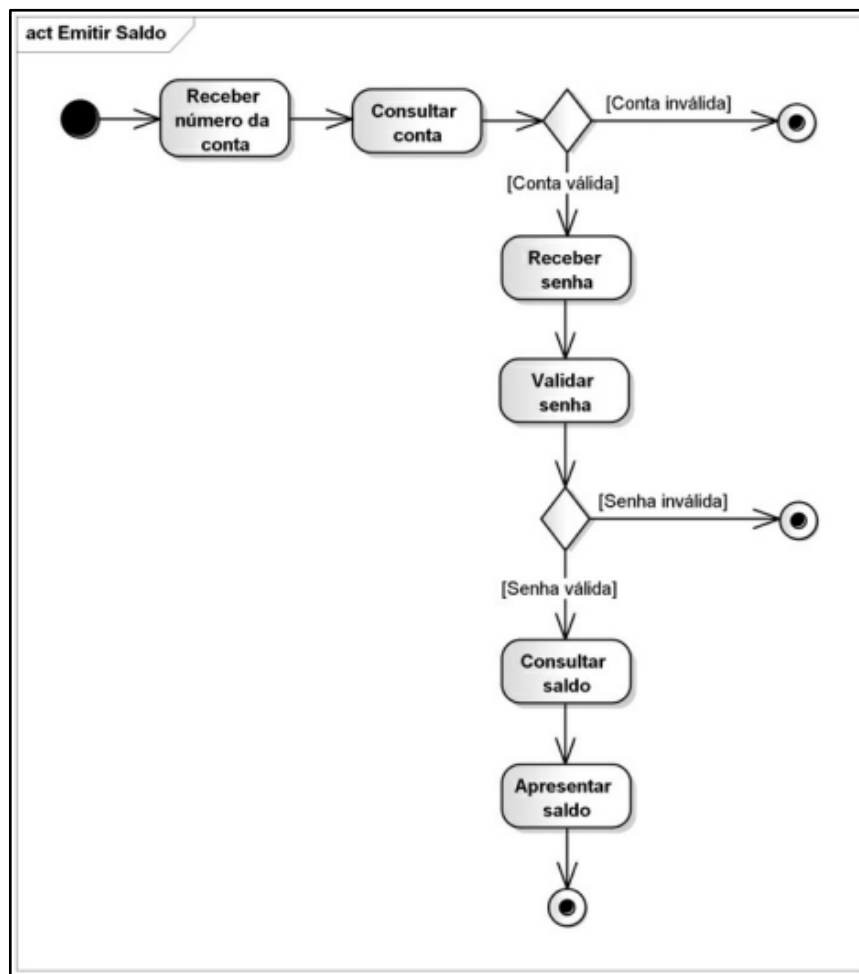
Fonte: (Guedes, 2018)

Evidenciado por Guedes (2018), o exemplo do diagrama de sequência do sistema de controle bancário revela as sequências para a emissão de saldo por meio suas respectivas instâncias.

Diagrama de Atividade

Conforme aponta Fowler (2005), um diagrama de atividades demonstra o fluxo de trabalho entre ações em um sistema, sendo representado através de nós de ação, setas e outros elementos visuais.

Figura 39 - Exemplo de Diagrama de Atividade



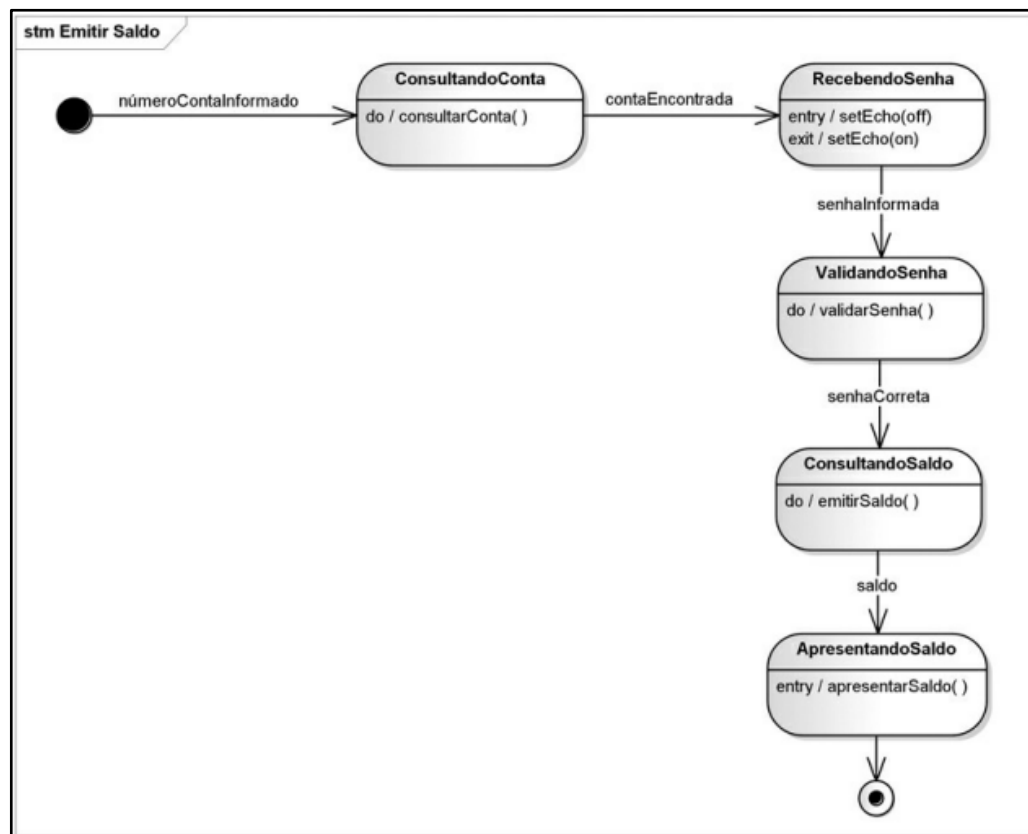
Fonte: (Guedes, 2018)

Segundo Guedes (2018), nesse diagrama de atividade de emissão de saldo temos todas as ações que moldam essa atividade que vão desde receber o número da conta, nó de ação e nó de decisão até de fato apresentar o saldo.

Diagrama de Máquina de Estados

Consoante Booch, Rumbaugh e Jacobson (2006), o diagrama de Máquina de Estados representa a visão dinâmica de um sistema, mostrando uma máquina de estados composta por estados, transições, atividades e eventos.

Figura 40 - Exemplo de Diagrama Máquina de Estados



Fonte: (Guedes, 2018)

Conforme demonstra Guedes (2018), nesse exemplo de diagrama de máquina de estados, é utilizado o caso de uso de emissão de saldo, representando os estados desse processo, bem como suas transições e eventos.

3. DESENVOLVIMENTO

4. CONSIDERAÇÕES FINAIS

REFERÊNCIAS

ALVES, William Pereira. **Construindo uma aplicação web completa com PHP e MySQL**. 1. ed. São Paulo: Novatec Editora, 2017. E-book.

ALVES, William Pereira. **Banco de dados: Teoria e Desenvolvimento**. 2. ed. São Paulo: Erica, 2021.

BENDORAITIS, Aidas; KRONIKA, Jake. **Desenvolvimento Web com Django 3 Cookbook: Soluções práticas para problemas comuns no desenvolvimento Web com Python**. 1 ed. São Paulo: Novatec Editora, 2020. E-book.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML: guia do usuário**. 1. ed. Rio de Janeiro: Elsevier Editora, 2006.

BORGES, Luiz Eduardo. **Python para desenvolvedores**. 1 ed. São Paulo: Novatec Editora. 2014. E-book.

CARDOSO, Carlos. **HTML 4 - Série Curso Básico & Rápido**. 1 ed. Rio de Janeiro: Axcel Books, 1999.

CARDOSO, Virgínia; CARDOSO, Giselle. **Sistemas de banco de dados: uma abordagem introdutória e aplicada**. 1. ed. São Paulo: Saraiva, 2012.

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

EIS, Davi.; FERREIRA, Elcio. **HTML5 e CSS3 com Farinha e Pimenta**. 1 ed. São Paulo: Tableless, 2012.

FLANAGAN, David. **JavaScript: O guia definitivo**. 6 ed. Porto Alegre: Bookman, 2013.

FOWLER, Martin. **UML Essencial: um breve guia para a linguagem-padrão de modelagem de objetos**. 3. ed. [S.l]: Bookman Editora, 2005. E-book.

FREEMAN, Elisabeth; FREEMAN, Eric. **Use a Cabeça! HTML com CSS e XHTML**. 2. ed. Rio de Janeiro: Alta Books, 2006.

FURTADO, Daniel. **Princípios de UX**: Entendendo o Design Centrado no Usuário. 1. ed. Santa Catarina, Joinville: Brauer, 2023.

GAMMA, Erich. **Padrões de projeto**: Soluções reutilizáveis de software orientado a objetos. São Paulo: Bookman Editora, 2008. E-book

GRANT, Will. **UX Design**: Guia definitivo com as melhores práticas de UX. 1. ed. São Paulo: Novatec Editora, 2019.

GRINBERG, Miguel. **Desenvolvimento web com Flask**: Desenvolvendo aplicações web com Python. 1 ed. São Paulo: Novatec Editora. 2018. E-book.

GRONER, Loiane. **Estruturas de Dados e Algoritmos com JavaScript**: Escreva um Código JavaScript Complexo e Eficaz Usando a Mais Recente ECMAScript. 2 ed. São Paulo: Novatec Editora, 2019.

GUEDES, Gilleanes T. A. **UML 2 - Uma Abordagem Prática**. 3. ed. São Paulo: Novatec Editora, 2018.

HEUSER, Carlos Alberto. **Projeto de banco de dados**. 6. ed. Porto Alegre: Bookman, 2009. E-book.

IHRIG, Colin J. **Pro Node.js para Desenvolvedores**. 1 ed. Rio de Janeiro: Editora Ciência Moderna Ltda, 2014.

JOBSTRAIBIZER, Flávia. **Criação de sites com CSS**. 1 ed. São Paulo: Digerati Books, 2009.

LEVY, Jaime. **Estratégia de UX**: Técnicas de estratégia de produto para criar soluções digitais inovadoras. 2. ed. São Paulo: Novatec Editora, 2021.

LOBO, Edson Junior Rodrigues. **Curso prático de MySQL**. 1. ed. São Paulo: Universo dos Livros, 2008. E-book.

MACHADO, Felipe Nery Rodrigues. **Banco de dados**: Projeto e Implementação. 4. ed. São Paulo: Erica, 2020.

MCKINNEY, Wes. **Python para análise de dados**: Tratamento de dados com Pandas, NumPy e IPython. 1 ed. São Paulo: Novatec Editora. 2018. E-book.

MEMÓRIA, Felipe. **Design para a internet: Projetando a experiência perfeita**. São Paulo: Elsevier Editora, 2006. E-book.

MENEZES, Nilo Ney Coutinho. **Introdução à programação com Python: Algoritmos e lógicas de programação para iniciantes**. 1 ed. São Paulo: Novatec Editora. 2010. E-book.

MILANI, André. **MySQL - Guia do Programador**. 1. ed. São Paulo: Novatec Editora, 2006.

MILETTO, Evandro Manara.; BERTAGNOLLI, Silvia de Castro. **Desenvolvimento de Software II: Introdução ao Desenvolvimento Web com HTML, CSS, JavaScript e PHP**. Porto Alegre: Bookman, 2014.

MITCHELL, Ryan. **Web Scraping com Python: Coletando mais dados da web moderna**. 2. ed. Brasil: Novatec Editora, 2019.

MORAES, William B. **Construindo aplicações com NodeJS**. 3 ed. São Paulo: Novatec Editora, 2021.

MUNIZ, Antonio *et al.* **Jornada API na prática: Unindo conceitos e experiências do Brasil para acelerar negócios com a tecnologia**. 1 ed. Rio de Janeiro: Brasport Editora, 2023. E-Book.

PEREIRA, Caio R. **Node.js: Aplicações web real-time com Node.js**. 1 ed. São Paulo: Casa do Código. 2014. E-book.

PEREIRA, Eduardo. **Trilhas Python: Programação multiparadigma e desenvolvimento Web com Flask**. São Paulo: Casa do Código Editora, 2018. E-book.

PEREIRA, Rogério. **User experience design**. São Paulo: Casa do Código Editora, 2018. E-book.

PODMAJERSKY, Torrey. **Redação Estratégica para UX: Aumente engajamento, conversão e retenção com cada palavra**. 1. ed. São Paulo: Novatec, 2019.

POWERS, Shelley. **Aprendendo Node: Usando JavaScript no servidor**. 1 ed. São Paulo: Novatec Editora, 2017.

PRESSMAN, Roger S; MAXIM, Bruce R. **Engenharia de software**: uma abordagem profissional. 9. ed. Porto Alegre: AMGH, 2021. E-book.

QUIERELLI, Davi A. **Criando Sites com Html-css-php**: Construindo um projeto iniciante. 1 ed. São Paulo: Clube de Autores, 2012.

SILVA, Maurício Samy. **Ajax com jQuery**: Requisições Ajax com a simplicidade de jQuery. 1 ed. São Paulo: Novatec Editora, 2009. E-book.

SILVA, Maurício Samy. **CSS3**: Desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3. 1 ed. São Paulo: Novatec Editora, 2011.

SILVA, Tiago. **Flask de A a Z**: Crie aplicações web mais completas e robustas em Python. São Paulo: Casa do código Editora, 2019. E-book.

SILVA, Mauricio Samy. **React Aprenda Praticando**: Desenvolva aplicações web reais com uso da biblioteca React e de seus módulos auxiliares. 1 ed. São Paulo: Novatec Editora, 2021. E-book.

SOARES, Adriano Mesquita (org.). **Tópicos especiais em engenharia**: Inovações e avanços tecnológicos. 1 ed. São Paulo: AYA Editora, 2023. E-book.

SORDI, José Osvaldo de. **Modelagem de dados**: estudos de casos abrangentes da concepção lógica à implementação. 1. ed. São Paulo: Erica, 2019.

STEFANOV, Stoyan. **Padrões JavaScript**. 1 ed. São Paulo: Novatec Editora, 2011.

STEFANOV, Stoyan. **Primeiros passos com React**: Construindo aplicações web. 1 ed. São Paulo: Novatec Editora, 2016. E-book.

STEVENS, W. Richard. **Programação de rede Unix**: API para soquetes de rede. 3 ed. Porto Alegre: Bookman Editora, 2008. E-Book.

TEIXEIRA, Fabricio. **Introdução e boas práticas em UX Design**. São Paulo: Casa do Código Editora, 2014. E-book.

ZHAO, Alice. **SQL - Guia Prático**: um guia para o uso de SQL. 4. ed. São Paulo: Novatec Editora, 2023.

ASSIS, Wendel Vilaça de; GOMIDE, João Victor Boechat. Web scraping em dados públicos: método para extração de dados dos gastos públicos dos vereadores da Câmara Municipal de Belo Horizonte. **Informação & Informação**, Londrina, v. 26, n. 4, p. 319-341, out./dez. 2021. Disponível em: <https://ojs.uel.br/revistas/uel/index.php/informacao/article/view/44123>. Acesso em: 30 mai. 2024.

DIAS, Juan Pablo da Silva; HEMAIS, Marcus Wilcox. Consumidores de baixa renda e compras on-line: Receios em consumir pela internet. **REGE - Revista de Gestão**, São Paulo, v. 22, n. 2, p. 115-132, jan./mar. 2015. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1809227616300960>. Acesso em: 03 mai. 2024.

FARIAS, Marcello Tenorio de; ANGELUCI, Alan César Belo; PASSARELLI, Brasilina. Web scraping e ciência de dados na pesquisa aplicada em comunicação: um estudo sobre avaliações online. **Revista Observatório**, Palmas, v. 7, n. 3, p. 1-22, jul./set. 2021. Disponível em: <https://repositorio.usp.br/item/003093096>. Acesso em: 29 mai. 2024.

MACHADO, Cristian Cleder *et al.* Um Web Crawler para Projeções e Análise de Vulnerabilidades de Segurança e Consistência Estrutural de Páginas Web. **Revista de Empreendedorismo, Inovação e Tecnologia**, [S.L.], v. 2, n. 2, p. 3-12, 2015. Disponível em: <https://seer.atitus.edu.br/index.php/revistas/article/view/869>. Acesso em: 18 mai. 2024.

NASCIMENTO, João P. B.; CAPANEMA, Daniel de O.; PEREIRA, Adriano C.M. Projeto e análise de desempenho de um algoritmo iterativo para grandes grafos em um ambiente distribuído. **RBCA – Revista Brasileira de Computação Aplicada**, Minas Gerais, v. 11, n. 1, p. 36-47, mar. 2019. Disponível em: <https://seer.upf.br/index.php/rbca/article/view/8738/114114519>. Acesso em: 31 mai. 2024.

RODRIGUES, Quemuel Baruque de Freitas *et al.* Webscraping em R: uma abordagem para investigação em ciências sociais. **Simbiótica**, Vitória, v. 8, n. 4, p. 191-215, set./dez. 2021. Disponível em: <https://periodicos.ufes.br/simbiotica/article/view/37351>. Acesso em: 31 mai. 2024.

SOUZA, Renato Rocha; CAFÉ, Lígia Maria Arruda. Análise de sentimento aplicada ao estudo de letras de música. **Informação & Sociedade: Estudos**, João Pessoa, v. 28, n. 3, p. 275-286, set./dez. 2018. Disponível em: <https://periodicos.ufpb.br/index.php/ies/article/view/34884>. Acesso em: 30 mai. 2024.

ABBA, Ihechikara. **Como Usar o Tailwind CSS para Desenvolver Rapidamente os Sites da Snazzy**. 2023. Disponível em: <https://kinsta.com/pt/blog/tailwind-css/>. Acesso em: 23 abr. 2024.

BOLZANI, Isabela. **61% dos brasileiros compram mais pela internet do que em lojas físicas, aponta estudo**. G1 globo, 2022. Disponível em: <https://g1.globo.com/economia/noticia/2022/12/14/61percent-dos-brasileiros-compram-mais-pela-internet-do-que-em-lojas-fisicas-aponta-estudo.ghtml>. Acesso em: 24 mar. 2024.

BUTEWICZ, Thanael. **Utilidades Python: Obtendo conversão de moedas**. Casa do desenvolvedor. 2022. Disponível em: <https://forum.casadodesenvolvedor.com.br/topic/45430-utilidades-python-obtendo-conversao-de-moedas>. Acesso em: 23 mai. 2024.

DIDÁTICA TECH. **Tutorial de Scrapy para iniciantes em Python**. 2022. Disponível em: <https://didatica.tech/tutorial-de-scrapy-para-iniciantes-em-python/>. Acesso em: 31 maio 2024.

DUKE, Justin. **Como Fazer Crawling em uma Página Web com Scrapy e Python 3**. 2018. Disponível em: <https://www.digitalocean.com/community/tutorials/como-fazer-crawling-em-uma-pagina-web-com-scrapy-e-python-3-pt>. Acesso em: 26 de maio de 2024.

E-COMMERCE BRASIL. **O impulso do e-commerce no Brasil: perspectivas até 2027**. 2024. Disponível em: <https://www.ecommercebrasil.com.br/artigos/o-impulso-do-e-commerce-no-brasil-perspectivas-ate-2027>. Acesso em: 04 jun. 2024.

FAUSTINO, Marco; LOBATO, Gisele. **Golpes virtuais crescem 65% com brasileiros passando mais tempo na internet**. Terra, 2023. Disponível em: <https://www.terra.com.br/noticias/checamos/golpes-virtuais-crescem-65-com-brasileiros-passando-mais-tempo-na-internet,5b5a67f228cff03faf6084e2fc39eaa60q7kkt7m.html>. Acesso em: 28 mar. 2024.

GOMES, Ana Maria. **O que é Web Scraping e Como Utilizar com Python**. 2024. Disponível em: <https://hub.asimov.academy/tutorial/o-que-e-web-scraping-e-como-utilizar-com-python/>. Acesso em: 31 de maio de 2024.

NEVES, Vinícios. **Composição elegante: Descubra o poder do Tailwind CSS**. 2023. Disponível em: <https://marcosviniciosneves.medium.com/composi%C3%A7%C3%A3o-elegante-descubra-o-poder-do-tailwind-css-4aae5306bf91>. Acesso em: 23 abr. 2024.

OLIVEIRA, Rafael. **Criando componentes com Tailwind CSS**. 2023. Disponível em: <https://www.treinaweb.com.br/blog/criando-componentes-com-tailwind-css>. Acesso em: 23 abr. 2024.

SÉ, Letícia. **Nove em dez brasileiros pesquisam online antes de comprar, diz Google**. Globo, 2023. Disponível em: <https://revistapegn.globo.com/negocios/noticia/2023/03/nove-em-dez-brasileiros-pesquisam-online-antes-de-comprar-diz-google.ghtml>. Acesso em: 27 mar. 2024.