VELAFOCO

BRUNO CASTRO TOMAZ (10389988)

LUIZ GABRIEL PROFIRIO MENDES (10382703)

RAFAELA PERROTI ZYNGIER (10395424)

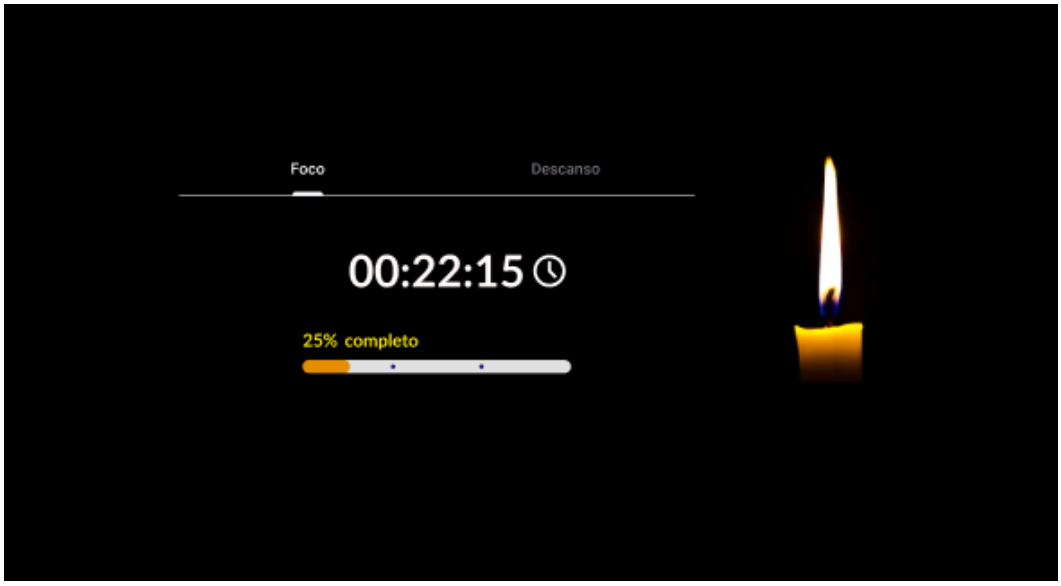
GUSTAVO SAAD MALUHY ANDRADE (10332747)

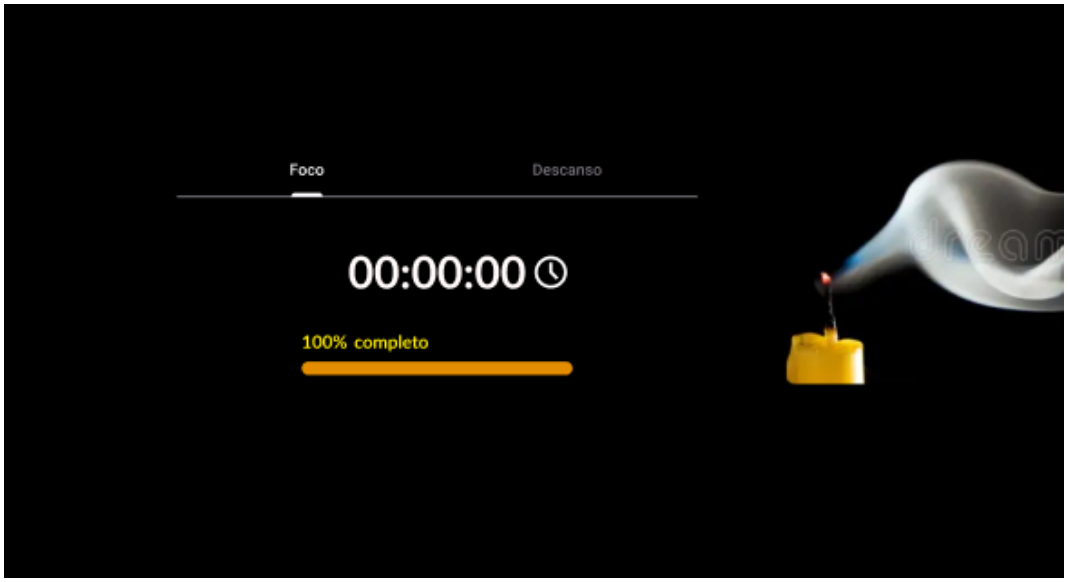
IDEAÇÃO

Com o objetivo de impulsionar a produtividade e o bem-estar da comunidade, nosso projeto inovador apresenta um website interativo que integra a consagrada técnica Pomodoro. A experiência do usuário será enriquecida por uma interface intuitiva, onde a gestão do tempo se torna uma jornada visualmente agradável. Acompanhe em tempo real o progresso com uma vela virtual que gradualmente se apaga, simbolizando a passagem do tempo e o aumento do foco. Ao final de cada ciclo Pomodoro, desfrute de pausas revigorantes de 5 minutos, ideais para relaxar e recarregar as energias.

Além disso, uma barra de progresso dinâmica oferece uma visão clara e precisa do seu progresso, permitindo que você otimize seu tempo com eficiência. Este projeto visa promover a organização e o foco, reduzir a procrastinação e o estresse, e auxiliar na gestão eficaz do tempo.

PROTÓTIPO INICIAL





https://www.figma.com/design/izJe3nBoFRYeaXA2Dh2b9L/velafoco?node-id=4-1364&t=WonHWnKfV2NtJmcE-0

Link do github do projeto:
https://github.com/BrunoCastroTomaz/VelaFoco

TUTORIAL

HTML5

O código em HTML5 é composto de tags que formam a estrutura da página web. Essas tags devem ser utilizadas de maneira semântica, isto é, de uma forma que deixe bem claro qual é a função daquele elemento na página. Ao longo deste tutorial, você verá como estruturamos o código HTML da nossa página seguindo esses princípios e as melhores práticas em desenvolvimento web. Neste primeiro tutorial, desconsideraremos aspectos estéticos.

Estrutura básica

Toda página HTML possui a seguinte estrutura básica de tags:

```
<!DOCTYPE html>
<html lang="pt-BR">
   <head>
   </head>
   <body>
   </body>
</html>
```

<!DOCTYPE html>

Informa ao navegador que o documento está escrito em HTML5.

<html lang=”pt-BR”></html>

É a tag raiz do documento HTML. Todos os elementos devem estar dentro dela. Podemos especificar o idioma da página, como fizemos em `<html lang="pt-BR">`.

<head></head>

Contém metadados e configurações da página

<body></body>

Contém todo o conteúdo visível da página, como textos, imagens, botões, etc.

Note que a delimitação do escopo de uma tag **<>** é dado pelo seu fechamento, que é representado por **</>**.

Head - Metadados e Configurações da Página

O head é muito importante, pois nele estarão contidas as configurações e metadados que garantirão o bom funcionamento da sua página web. Para o CandleFocus, temos o seguinte head:

```
<head>
   <meta charset="UTF-8"/>
   <title>CandleFocus</title>
</head>
```

<meta charset=”UTF=8”/>

Define o encoding da página para UTF-8. Dessa forma, garantimos que caracteres especiais sejam exibidos corretamente.

<title>CandleFocus</title>

Definimos que o título da página é CandleFocus. Dessa forma, o nome CandleFocus passa a aparecer na aba do navegador.

Body - Conteúdo da Página

```
    <body>


    </body>
```

O body é uma parte fundamental do código, pois é nela que colocaremos nossos textos, imagens, vídeos, tabelas, formulários, etc. Ou seja, todo o conteúdo da nossa página web estará contido dentro do body.

Estrutura Básica

Não é uma regra absoluta, mas é comum ver páginas web que são divididas entre 3 grandes seções: header, main e footer. O CandleFocus não é diferente e nós escreveremos nosso código HTML se baseando na seguinte estrutura:

```
    <body>
       <header>
       </header>


       <main>
```

```
        </main>


        <footer>
        </footer>
    </body>
```

<header></header>

O header representa um container para conteúdos introdutórios da sua página ou um recurso de navegação. Nele podemos ter tags de título, como <h1> - <h6>, por exemplo.

<main></main>

A main representa o conteúdo principal da página. O ideal é que tenhamos apenas 1 main por documento.

<footer></footer>

O footer é o rodapé da sua página. Normalmente possui informações sobre os autores daquela página, copyright, informações para contato e outros links úteis para navegação.

Em seguida veremos tags importantes que foram usados nesse projeto:

<nav>

```
<nav>
                    <ul>
                        <li><button id="fase-foco-pomodoro" class="foco">Foco</li>
                        <li><button id="fase-descanso-pomodoro"
class="descanso">Descanso</li>
                    </ul>
                </nav>
```

A tag <nav> define uma seção de navegação da página. No projeto, ela é usada para agrupar os botões que permitem alternar entre os modos "Foco" e "Descanso".

Os elementos <ul> e <li> com <button> dentro criam uma lista de navegação com dois botões: "Foco" e "Descanso". Cada botão tem um id e uma class para poder ser estilizado com CSS e controlado com JavaScript.

<button>

```
<button id="start" class="start" onclick="start()">START</button>
```

A tag <button> cria um botão clicável. O botão com id="start" inicia o cronômetro quando o usuário clica, ativando a função start() definida no JavaScript (explicado adiante).

<img>

```
<img src="imagens/vela.png" alt="Vela acesa" class="vela-image">
```

A tag <img> exibe uma imagem na página. Aqui, mostra uma vela acesa, com o caminho da imagem especificado em srce uma descrição alternativa em alt.

<article>

```
<article class="timer">
                    <time><p id="timer"></p></time>
                    <article class="progress-bar">
                    <article class="progress-fill" id="progress-fill"></article>
                    </article>
                </article>
```

A tag <article> é usada para agrupar conteúdos independentes e com significado próprio dentro de uma página. No projeto, ela organiza a seção do temporizador: o <article class="timer"> envolve tanto o contador quanto a barra de progresso. Dentro dele, a tag <time> indica que o conteúdo está relacionado ao tempo e contém um parágrafo <p id="timer"> onde o tempo é exibido dinamicamente. Logo abaixo, outro <article> com a classe progress-bar serve como a estrutura da barra de progresso, e dentro dele há um <article class="progress-fill" id="progress-fill">, que representa a parte da barra que vai sendo preenchida conforme o tempo passa.

CSS

Se você quer deixar a sua página web mais colorida, bonita e divertida, o CSS (Cascading Style Sheets) vai ser o seu aliado. Enquanto o HTML estrutura o conteúdo (como títulos, parágrafos e imagens), o CSS define como esse conteúdo deve aparecer — por exemplo, as cores, fontes, tamanhos, espaçamentos e o layout da página. Vamos começar com um exemplo simples:

```
p {
  color: blue;
  font-size: 18px;
}
```

Esse código faz com que todos os parágrafos (<p>) da página apareçam em azul e com fonte de 18 pixels.

Você também pode usar classes para aplicar a mesma estilização a todos os elementos. Por exemplo, se voce quer que um texto esteja em destaque, você pode usar um html com a classe destaque:

```
        <p class="destaque">Olá, mundo!</p>
```

E no css deixar todos os parágrafos em destaque em vermelho e negrito:

```
.destaque {
   color: red;
   font-weight: bold;
 }
```

Flexbox - o layout facilitado

Para usar o Flexbox, primeiro você precisa transformar um contêiner em flexível usando display: flex. No código deste projeto, você vai ver isso em vários elementos:

```
body {
   display: flex;
}

.container {
   display: flex;
}



.content {
   display: flex;
}

ul {
   display: flex;
}
```

Mas por que fazemos isso?

Sem o display: flex, o HTML organiza os elementos apenas de forma vertical e empilhada, um embaixo do outro (o comportamento padrão). Ele entende quem está dentro de quem, mas não como posicionar os elementos uns em relação aos outros — por exemplo, lado a lado, centralizados ou espaçados igualmente.

Com o Flexbox, conseguimos controlar o layout de forma muito mais precisa: colocar elementos na horizontal, centralizar, alinhar, distribuir espaço e adaptar o layout para diferentes tamanhos de tela.

Quando um container se torna flex, ‘e possivel acionar diversas opções que alteram o layout no container pai e no container filho

No container pai:

- display: flex - Ativa o Flexbox no elemento.
- flex-direction - Define a direção dos itens (por exemplo: row, column...).
- justify-content - Alinha os itens na horizontal (esquerda, centro, espaçados...).

- align-items - Alinha os itens na vertical (topo, centro, base...).
- flex-wrap - Permite que os itens quebrem linha se necessário.
- gap - Define o espaço entre os itens dentro do contêiner.

No container filho:

- flex - Define o quanto o item cresce ou encolhe em relação aos outros.
- order - Altera a ordem de exibição do item (sem mudar o HTML).
- align-self - Alinha um item individual de forma diferente dos outros.

Vamos analizar mais a funto um dos exemplos do código desse projeto:

```css
.container {
   display: flex;
   text-align: center;
   align-items: center;
   gap: 100px;
   flex: 1;
   flex-wrap: wrap;
   justify-content: center;
}
```

A classe .container usa Flexbox para organizar dois blocos — o do timer e o da imagem — lado a lado. A propriedade display: flex ativa o Flexbox. O justify-content: center centraliza os blocos na horizontal e o align-items: center alinha verticalmente. O gap: 100px cria um espaço entre eles, e o flex-wrap: wrap permite que os blocos quebrem de linha em telas menores, o que ajuda na responsividade.

Dois casos especiais: “#” e “:”

```css
li:hover{
   color: white;
   border-bottom: 1px solid white;;
}
.start:hover {
   background-color: #00befe;
}

footer#rodape {
   clear: both;
   border-top: 1px solid;
}

footer#rodape p {
   font-family: ArialRounded;
```

```
    text-align: center;

}
```

No CSS, o # é utilizado para selecionar um elemento pelo seu id. Quando você vê algo como #rodape, significa que o estilo será aplicado ao elemento HTML que possui o atributo id="rodape". Portanto, o # é um seletor de id.

Já o : em start:hover é utilizado para aplicar estilos em um estado específico de um elemento. O :hover é um pseudo-classe que se ativa quando o usuário passa o mouse sobre o elemento. No exemplo, start:hover significa que o estilo será aplicado ao elemento com a classe start quando o mouse estiver sobre ele, alterando, por exemplo, a cor de fundo do botão, como foi feito no código com background-color: #00befe;.

Media Query

Uma media query no CSS é uma regra que aplica estilos diferentes dependendo das características do dispositivo, como a largura da tela, a altura da tela, a resolução, etc.

```
    @media (max-width: 768px) {
        .container {
            flex-direction: column;
            gap: 20px;
        }

        .vela-image {
            order: 1;
        }

        ul {
            display: flex;
            flex-direction: row; /* Ensure buttons always stay side by side */
            justify-content: center; /* Center them horizontally */
            align-items: center;
            gap: 10px; /* Add space between them */
            list-style: none;
            padding: 0;
        }


    }
```

Essa media query afeta o layout da página em telas pequenas (com largura de 768px ou menos). Ela faz com que:

- A imagem da vela (.vela-image) seja movida para abaixo do timer (graças ao order: 1).
- Os itens dentro do .container (o timer e a imagem) sejam empilhados verticalmente, com um espaçamento de 20px entre eles.
- Os botões de navegação (<li>) serão centralizados horizontalmente, com um espaçamento de 10px entre eles. Eles ficarão lado a lado, sem que se sobreponham ou fiquem com espaçamento desorganizado. A remoção dos marcadores de lista (list-style: none;) também melhora a aparência, fazendo com que a lista de botões pareça mais limpa.

JavaScript

O código em javascript é essencial no funcionamento do pomodoro. Após adicionar a estrutura da página em HTML e os estilos em CSS, é essencial o javascript para poder adicionar a funcionalidade do site.

O usuário pode iniciar, pausar, retomar ou trocar entre os modos de foco e descanso. Há ainda uma barra de progresso que mostra quanto tempo já passou.

A funcionalidade principal consiste em:

- Iniciar, Pausar, Retomar o timer
- Alternar entre modos de descanso e foco
- Feedback visual de tempo e progresso

Elementos HTML Usados

```
let timerDisplay = document.getElementById('timer');
let startButton = document.getElementById('start');
let focusButton = document.getElementById('fase-foco-pomodoro');
let breakButton = document.getElementById('fase-descanso-pomodoro');
let progressBar = document.getElementById('progress-fill');
```

Estes elementos são obtidos do HTML via `getElementById` e serão usados para mostrar o tempo, iniciar o timer, mudar de fase e atualizar a barra de progresso.

Variáveis de Tempo

```
let defaultFocusTime = 25 * 60; // 25 minutos em segundos
let defaultBreakTime = 5 * 60; // 5 minutos em segundos
let timeLeft = defaultFocusTime;
let timerInterval = null;
let totalTime = defaultFocusTime;
```

- `defaultFocusTime` e `defaultBreakTime` definem os tempos padrão.
- `timeLeft` é o tempo que ainda resta.
- `totalTime` é usado para calcular a porcentagem da barra de progresso.
- `timerInterval` guarda o `setInterval` ativo.

Função `formatTime(seconds)`

```
function formatTime(seconds) {
    const minutes = Math.floor(seconds / 60);
    const remainingSeconds = seconds % 60;
    return `${String(0).padStart(2, '0')}:${String(minutes).padStart(2,
'0')}:${String(remainingSeconds).padStart(2, '0')}`;
}
```

Essa função formata o tempo em `HH:MM:SS`, transforma o tempo em segundos em minutos. As horas são sempre 0.

Exemplo:

```
formatTime(90); //equivale a "00:01:30"
```

Função `updateTimerDisplay()`

```
function updateTimerDisplay() {
    timerDisplay.textContent = formatTime(timeLeft);
}
```

Atualiza o conteúdo de `timerDisplay` com o tempo restante formatado.

Função `startTimer()`

```
function startTimer() {
    // Limpa qualquer intervalo anterior
    if (timerInterval) clearInterval(timerInterval);

    let startTime = Date.now();
    let endTime = startTime + timeLeft * 1000;

    timerInterval = setInterval(() => {
        let now = Date.now();
        timeLeft = Math.max(0, Math.round((endTime - now) / 1000));

        updateTimerDisplay();
        updateProgressBar();

        if (timeLeft === 0) {
            clearInterval(timerInterval);
            alert("Tempo concluído!");
            timeLeft = totalTime; // reseta
            updateTimerDisplay();
            startButton.textContent = "START";
            startButton.onclick = start;
        }
    }, 1000);

    startButton.textContent = "PAUSE";
    startButton.onclick = pauseTimer;
}
```

Essa função:

1. Calcula o tempo final (`endTime`)
2. Atualiza `timeLeft` a cada segundo
3. Ao chegar em 0:
    - Para o timer
    - Mostra um alerta
    - Reseta o botão

Função `pauseTimer()`

```
function pauseTimer() {
    clearInterval(timerInterval);
    timerInterval = null;
```

```
    startButton.textContent = "RESUME";
    startButton.onclick = startTimer;
}
```

- Interrompe o temporizador atual.
- Muda o botão para “RESUME”, indicando no HTML que o temporizador foi pausado.

Função `start()`

```
function start() {
    updateTimerDisplay();
    startTimer();
}
```

Serve como função de inicialização padrão ao clicar no botão START. Atualiza a tela e chama `startTimer`.

Função `updateProgressBar()`

```
function updateProgressBar() {
    let progressPercentage = ((totalTime - timeLeft) / totalTime) * 100;
    progressBar.style.width = progressPercentage + "%";
}
```

Calcula a porcentagem completada e altera o `style.width` da barra de progresso.

Exemplo: Se já se passaram 15 dos 25 minutos:

((1500 - 600) / 1500) * 100 = 60%

Trocar entre Foco e Descanso

Botão de Foco:

```
focusButton.addEventListener("click", () => {
    clearInterval(timerInterval);
    timeLeft = defaultFocusTime;
    totalTime = defaultFocusTime;
    updateTimerDisplay();
```

```
    updateProgressBar();
    startButton.textContent = "START";
    startButton.onclick = start;
});
```

Botão de Descanso:

```
breakButton.addEventListener("click", () => {
    clearInterval(timerInterval);
    timeLeft = defaultBreakTime;
    totalTime = defaultBreakTime;
    updateTimerDisplay();
    updateProgressBar();
    startButton.textContent = "START";
    startButton.onclick = start;
});
```

Esses botões reiniciam o temporizador, mudando o tempo de foco ou descanso, limpando o intervalo ativo e atualizando o display.

Inicialização

```
updateTimerDisplay();
startButton.onclick = start;
```

Ao carregar a página:

- O display mostra `25:00`
- O botão está pronto para chamar a função "start()" para iniciar o temporizador