PROGRAMA DE MESTRADO PROFISSIONAL EM ENSINO DE CIÊNCIAS E MATEMÁTICA

PRODUTO EDUCACIONAL

Construção de um Simulador Computacional para o Ensino de Física e Matemática em fenômenos que envolvem o uso de funções trigonométricas

MSc Rodrigo José Nolasco Silva

Prof. Dr. Osvaldo Canato Júnior

São Paulo (SP)
2024

Catalogação na fonte
Biblioteca Francisco Montojos - IFSP Campus São Paulo
Dados fornecidos pelo(a) autor(a)

F633a Silva, Rodrigo José Nolasco
Produto Educacional: Manual de Construção de um Simulador Computacional para o Ensino de Física e Matemática em fenômenos que envolvem o uso de Funções Trigonométricas / Rodrigo José Nolasco Silva. São Paulo: [s.n.], 2024.
33 f. il.

Orientador: Osvaldo Canato Júnior

() - Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, IFSP, 2024.

1. Pensamento Computacional. 2. Simulador Computacional. 3. Ensino de Física. 4. Ensino de Matemática. I. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo II. Título.

CDU

Produto Educacional apresentado como requisito à obtenção do grau de Mestre em Ensino de Ciências e Matemática pelo Programa de Mestrado Profissional em Ensino de Ciências e Matemática do Instituo Federal de Educação, Ciência e Tecnologia de São Paulo, campus São Paulo. Aprovado em banca de defesa de mestrado no dia 17/out./2024.

AUTORES

Rodrigo José Nolasco Silva: Licenciado em Matemática pela Universidade Braz Cubas e Mestre em Ensino de Ciências e Matemática pelo Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP). Atualmente é Professor de matemática e física do Ensino Médio da escola Laurinda Cardoso Mello Freire.

Osvaldo Canato Júnior: Possui graduação em Física pela Universidade Federal de São Carlos (1986), mestrado em Ensino de Ciências (Modalidade Física) (2003) e doutorado em Ensino de Ciências (Modalidade Física) (2014). Atualmente é professor do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP). Tem experiência na área de Educação, com ênfase em Ensino de Ciências e Matemática, atuando nos seguintes temas: formação de professores, educação para cidadania e também na função de coordenador de educação à distância.

Apresentação do Produto Educacional

Esse material, apresentado como Produto Educacional, é parte integrante de nossa pesquisa intitulada: **Construção de um simulador computacional para o ensino de Física e Matemática em fenômenos que envolvem o uso de funções trigonométricas**, desenvolvida no Programa de Mestrado Profissional em Ensino de Ciências e Matemática do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP), sob orientação do Professor Doutor Osvaldo Canato Júnior.

Nosso Produto Educacional consiste em 33 páginas.

Catalogação na fonte
Biblioteca Francisco Montojos - IFSP Campus São Paulo
Dados fornecidos pelo(a) autor(a)

s586m Silva, Rodrigo José Nolasco
Manual de Construção de um simulador computacional para o ensino de Física e Matemática em fenômenos que envolvem o uso de funções trigonométricas / Rodrigo José Nolasco Silva. São Paulo: [s.n.], 2024.
33 f. il.

Orientador: Osvaldo Canato Júnior

() - Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, IFSP, 2024.

1. Pensamento Computacional. 2. Simulador Computacional. 3. Ensino de Física. 4. Ensino de Matemática. I. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo II. Título.

CDD

ANEXO A – PRODUTO EDUCACIONAL

MANUAL PARA CRIAÇÃO DE UM SIMULADOR COMPUTACIONAL

APRESENTAÇÃO

O uso de simulações computacionais no Ensino de Ciências é hoje muito difundido e abordado em trabalhos acadêmicos, não sendo, no entanto, comum a discussão de como construir um simulador, sendo essa a intenção deste roteiro. Para tal, se apresenta a seguir um passo a passo de como clonar o simulador vinculado a este produto educacional, de forma que o leitor possa tê-lo como referência para a construção de seus próprios simuladores.

As simulações envolvidas neste roteiro abrangem conceitos específicos nas áreas de Física e Matemática, como o plano inclinado, a reflexão da luz e as funções trigonométricas.

1 – Primeiros Passos

Em primeiro lugar, é preciso escolher uma plataforma para hospedar o projeto. Optamos pela utilização do GitHub porque é um servidor na nuvem que armazena os códigos e possui uma interface amigável, além de ser gratuito. Diante disso, para iniciarmos o roteiro de criação, realize os comandos abaixo:

1.1 – Acesse: github.com (Figura 1).

Figura 1 – Página de acesso ao GitHub

1.2 – Clique em ***"Inscrever-se"***, entre com seu e-mail, crie uma senha e escolha um *"username"*. Em seguida, clique em *"Create acount"* para você receber um e-mail e fazer a verificação e inserção de um código recebido. O GitHub é gratuito para projetos abertos.

Fonte: Disponível em: github.com Acesso em: 27 de out. 2024

1.3 – Acesse: https://desktop.github.com/ (Figura 2).

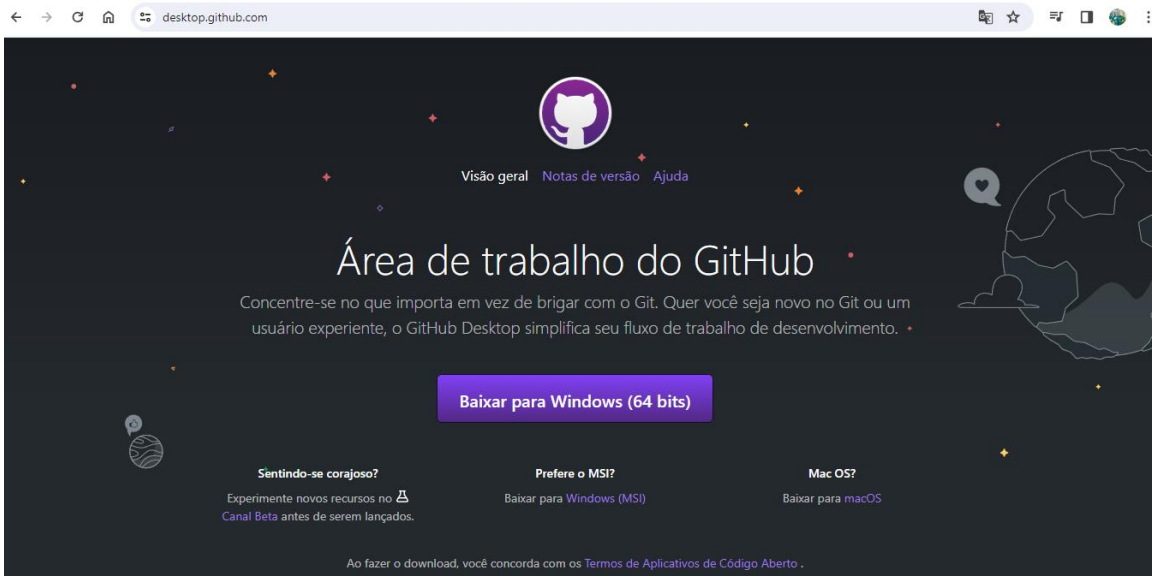

Figura 2 – Página para baixar o GitHub Desktop
Fonte: Disponível em: https://desktop.github.com/ Acesso em: 27 de out. 2024

1.4 – A partir daqui, vem a parte de criação do repositório local, ou seja, da pasta de arquivos na máquina onde vai trabalhar a partir do repositório remoto (GitHub). Faça o *download* do programa no seu computador. Ele irá reconhecer seu sistema operacional automaticamente.

1.5 – Na pasta ***"download"*** do seu computador, clique duas vezes sobre o ícone do arquivo, para iniciar a instalação (Figura 3).

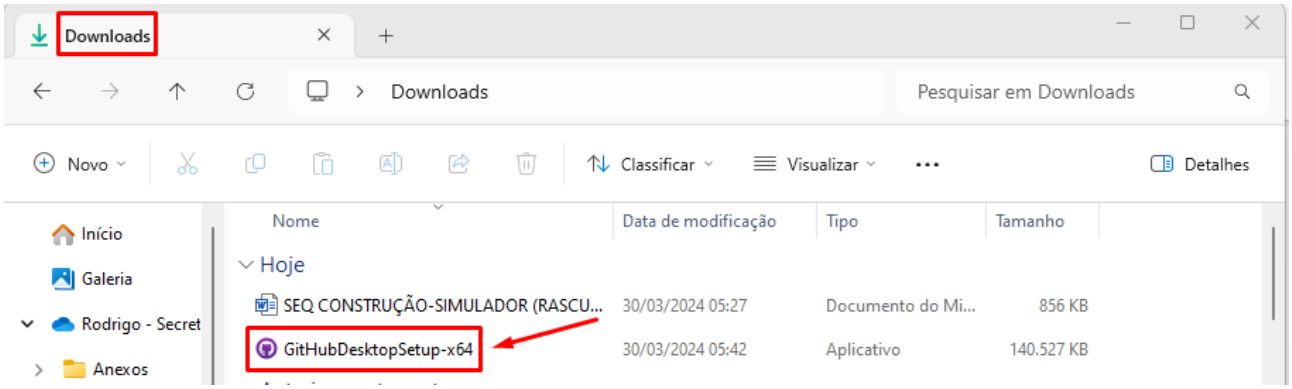

Figura 3 – Pasta de downloads no computador de trabalho
Fonte: Autoria própria

1.6 – Após a instalação, irá aparecer uma tela para você confirmar seu e-mail e clicar em ***"Finish"*** (Figura 4).

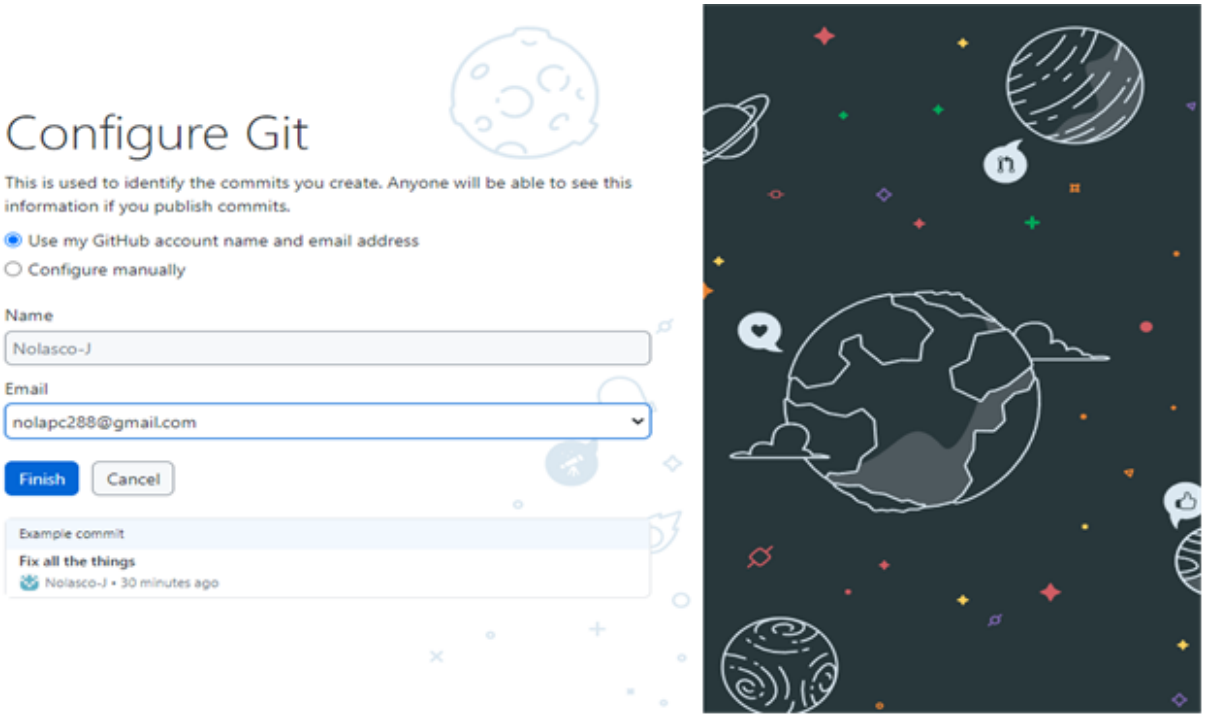

Figura 4 – Página para inserir e-mail de utilização no GitHub
Fonte: Autoria própria

Feito isso, você será direcionado para outra tela que será possível ter acesso à configuração do Git. O sistema fará o sincronismo automático com sua conta do GitHub. Para verificar, clique em ***"file"*** e depois em ***"Options"***, para você verificar que a sincronização ocorreu (Figura 5).

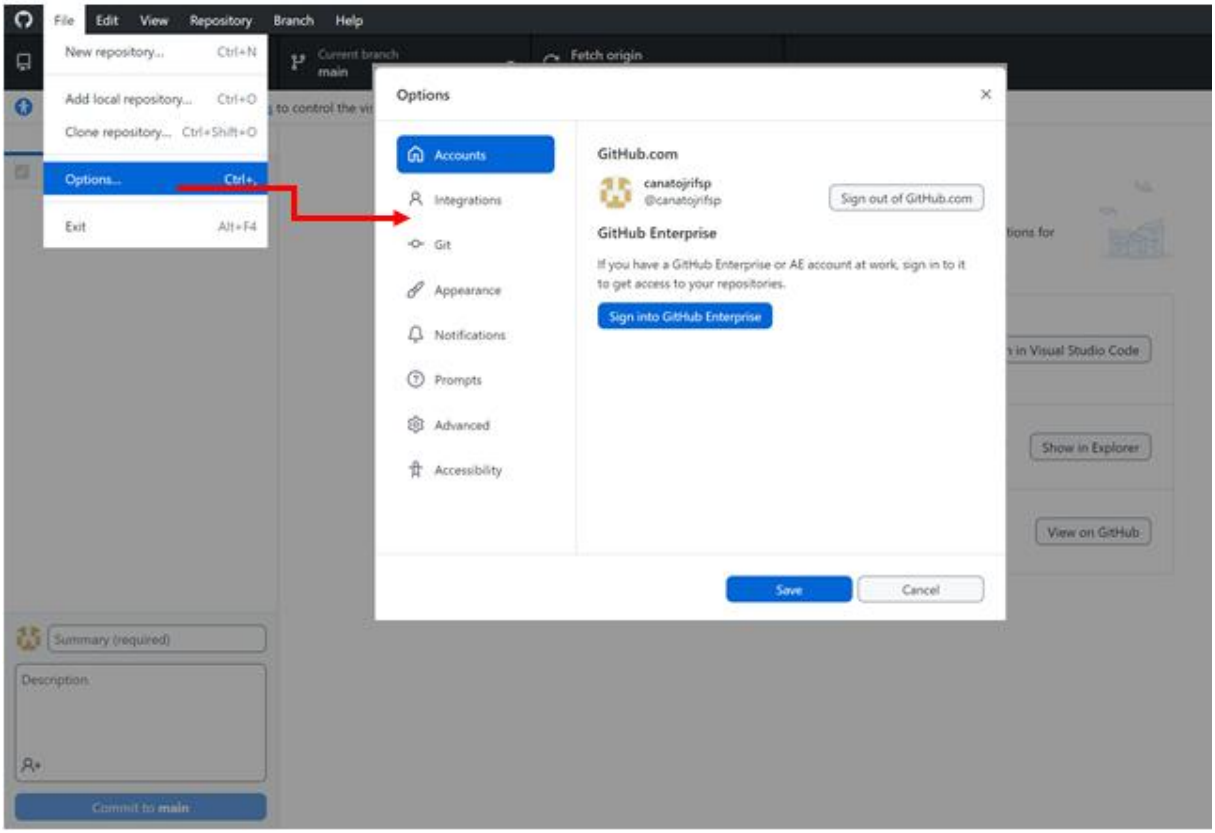

Figura 5 – Verificação da tela de sincronização
Fonte: Autoria própria

2 – Criando o Repositório

Repositório é o nome que o GitHub dá para o seu projeto, é uma pasta de arquivo que conterá todos os seus códigos de programação.

2.1 – Na tela inicial do GitHub Desktop, clique em ***"file"*** e depois em ***"New repository"*** para criarmos uma pasta com os arquivos do projeto (Figura 6).

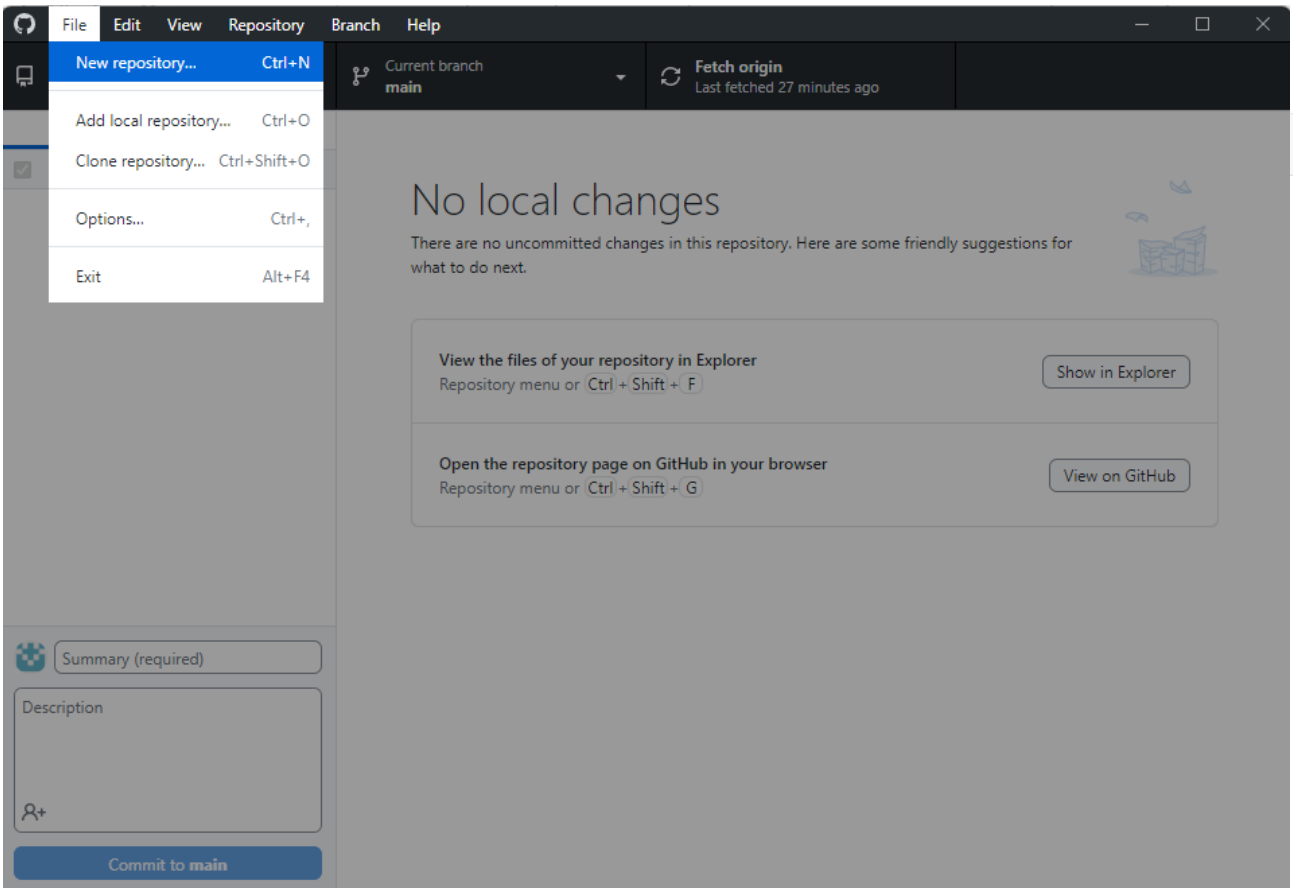

Figura 6 – Criando Repositório
Fonte: Autoria própria

2.2 – Digite o nome do seu projeto na caixa marcada abaixo; Descrição é opcional.

Selecione ***"README"*** e depois clique em ***"Create repository"*** (Figura 7).

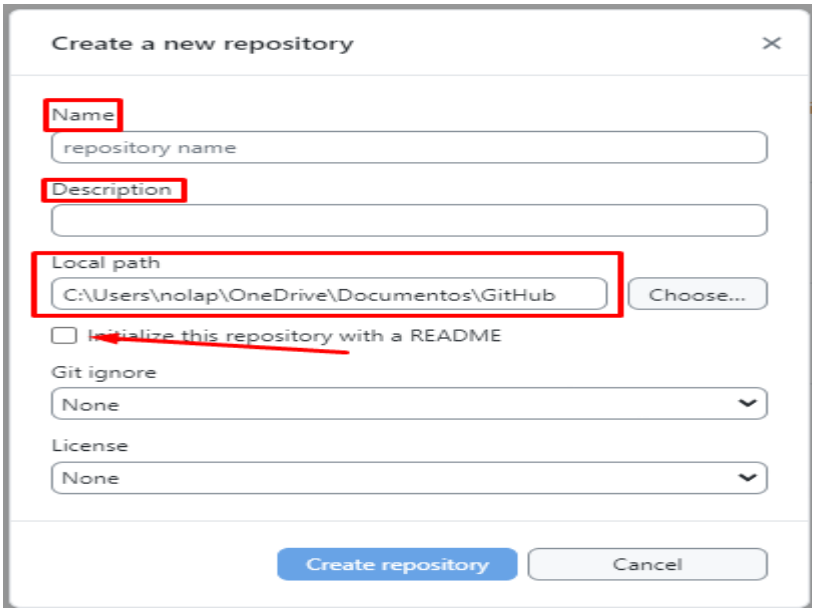

Figura 7 – Nomeando seu projeto
Fonte: Autoria própria

Os demais campos, não alterar. Em ***"Local path"***, o próprio sistema irá criar uma pasta *"GitHub"*, que estará em ***"Documentos"***, para armazenar os arquivos *"local"*, que são os arquivos que estarão salvos no seu computador para posterior envio à *"nuvem"* do repositório.

2.3 – Após a criação do repositório, você terá que publicar seu repositório, clicando no botão azul, ***"Publish repository"*** (Figura 8).

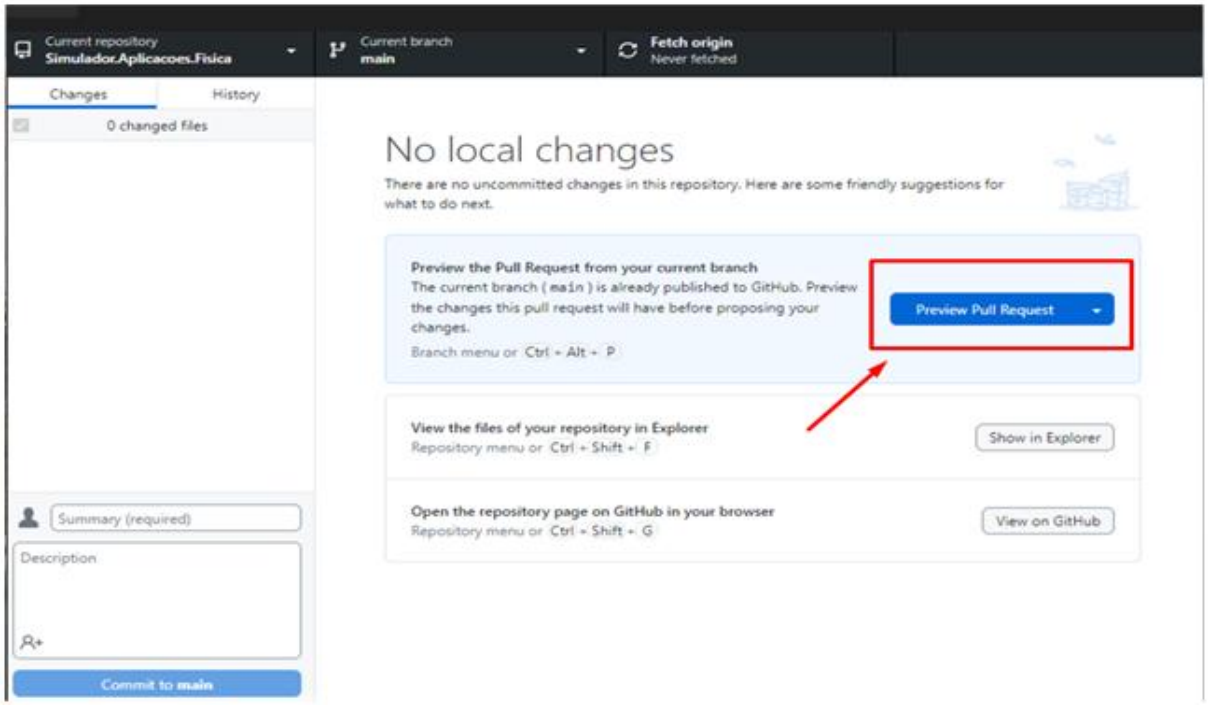

Figura 8 – Local de publicação do Repositório
Fonte: Autoria própria

2.4 – Ao finalizar a criação da pasta do repositório local, ela ficará visualmente disponível no site do GitHub (Figura 9).

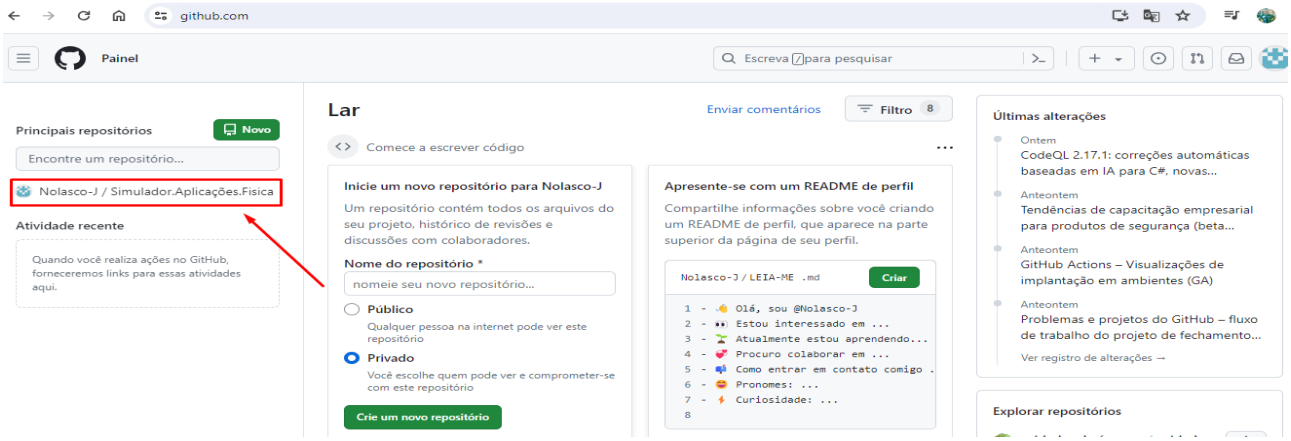

Figura 9 – Visualização da pasta criada no site do GitHub
Fonte: Autoria própria

Em seguida, o GitHub irá perguntar se o seu repositório será Público ou Privado. Aconselha-se aqui a marcar a opção ***"Keep this code private"***, ou seja, mantenha-o Privado.

3 – Aplicando os Códigos

3.1 – Volte ao GitHub e clique no seu repositório, **conforme passo 2.4** (Figura 09). Feito isso, você entrará na pasta ***"local"***, criada pelo GitHub no seu computador, onde será possível visualizar os arquivos, ***"README" e "index"***, que serão necessários para a programação e descrição do projeto no GitHub (Figura 10).

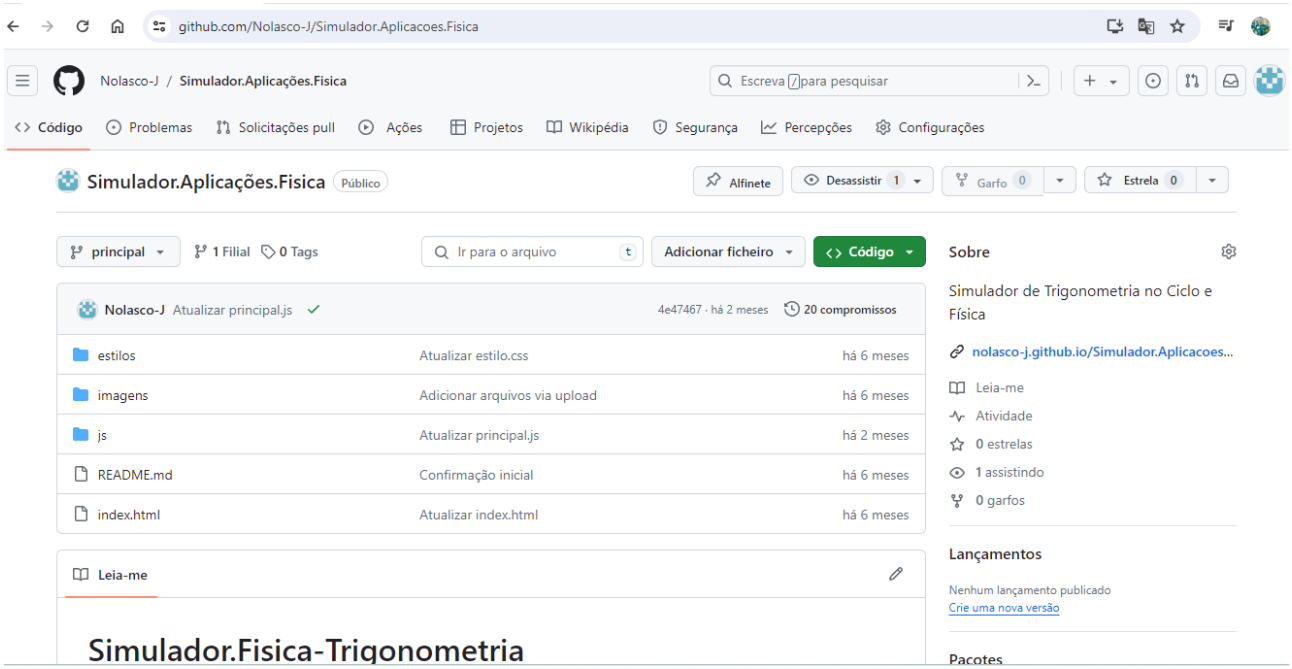

Figura 10 – Pasta *"local"* criada pelo GitHub no computador
Fonte: Autoria própria

3.2 – Vá em ***Documentos***, no seu computador, clique na pasta do GitHub e depois no título do seu repositório criado, para iniciarmos o processo de inserção dos códigos (Figura 11).

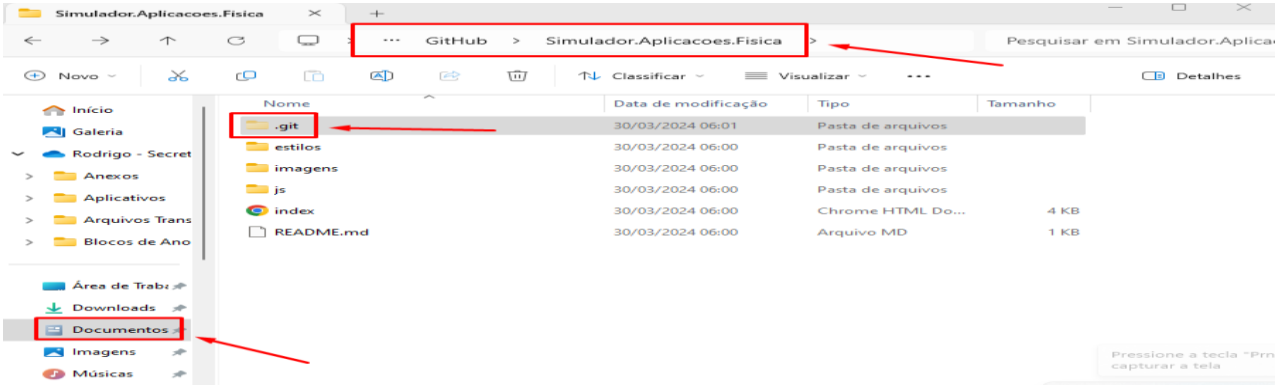

Figura 11 – Pasta *Documentos* no computador
Fonte: Autoria própria

3.3 – Todos os códigos do seu projeto serão desenvolvidos/salvos nesta pasta do seu computador. **Conforme você observou na imagem do passo 3.2** (Figura 11), você poderá utilizar todos os códigos já criados e desenvolvidos para este roteiro de criação do simulador computacional, clicando aqui[9]. Quando você executar o download da pasta de códigos no seu computador, elas deverão constar dentro do seu repositório, conforme **descrito no passo 3.2** (Figura 11).

3.4 – **Volte ao GitHub e atualize a página**, para que os arquivos baixados no seu computador sejam migrados e sincronizados com a nuvem.

3.5 - Neste momento, iremos precisar atualizar a base de códigos do GitHub para **o GitHub Desktop**. Isso é extremamente importante, pois o GitHub sempre irá atualizar qualquer alteração dos códigos a partir do seu computador, conforme **descrito no passo 3.2** (Figura 11) (Figura 12).

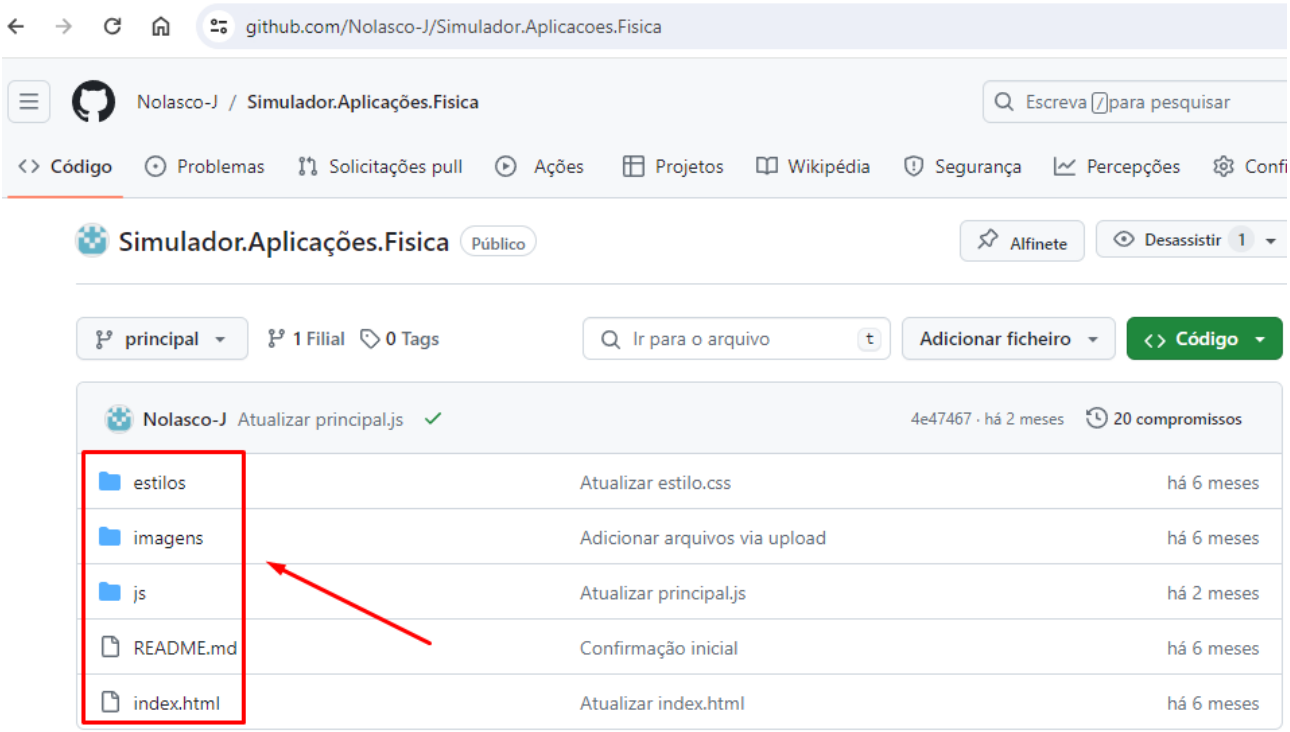

Figura 12 – Visualização dos códigos de Programação no Desktop
Fonte: Autoria própria

[9] Códigos do Simulador. Disponível em: https://drive.google.com/drive/folders/1mCOitBrQTh9Ny_fITQB5eiI0eT5RKRpW?usp=sharing Acesso em: 27 de out. 2024

4 – Clonando os Códigos (Commit to main / Push origin)

Nota: Apenas como informação ao usuário/leitor deste roteiro, existe a possibilidade de inserção dos códigos manualmente. Uma ferramenta que pode auxiliar os projetos de versionamento que são gerenciados pelo Git, é o ***Git Bash***, que é uma plataforma de controle de versões local podendo ser utilizada até em modo offline. Você pode acessá-la em: https://www.git-scm.com/downloads (Figura 13).

Figura 13 – Página de instalação do Git Bash
Fonte: Disponível em: https://www.git-scm.com/downloads Acesso em: 27 de out. 2024

Tela terminal de comandos do GIT, o ***Git Bash***, é o local onde são digitadas as linhas do código de programação. Apenas ilustramos que existem outras possibilidades de inserção dos códigos de programação, além da clonagem (Figura 14).

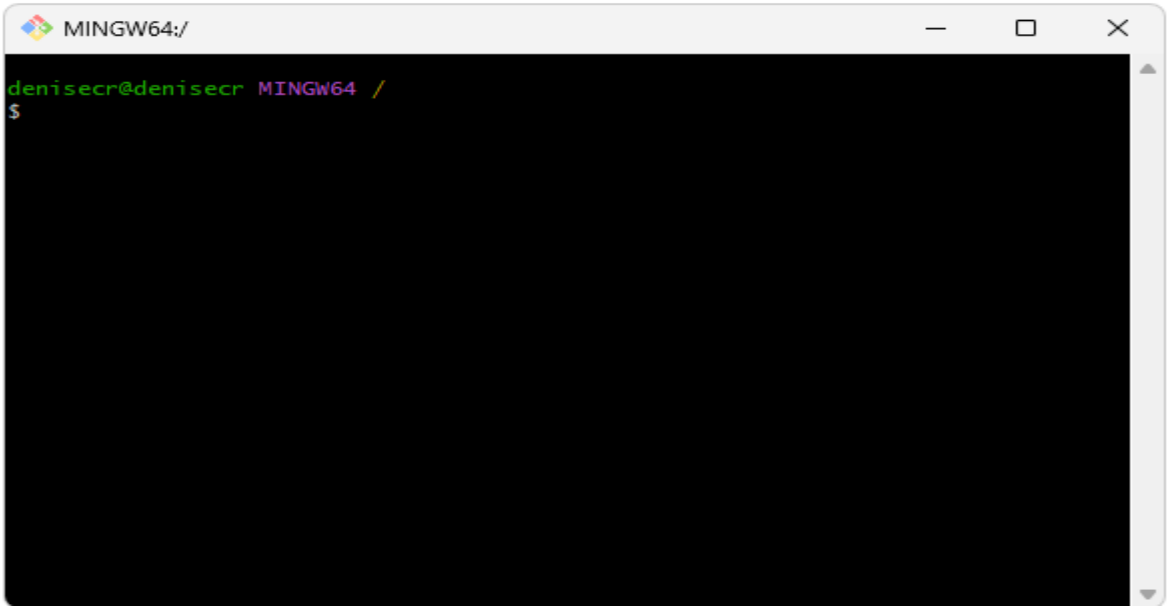

Figura 14 – Tela de comandos do Git Bash
Fonte: Autoria própria

4.1 – **No GitHub Desktop**, o comando ***"Commit to main"*** cria as versões do código, atualizando todas as suas pastas locais, do seu computador, preparando-as para serem publicadas por meio do ***"Push/origin"*** (Figura 15).

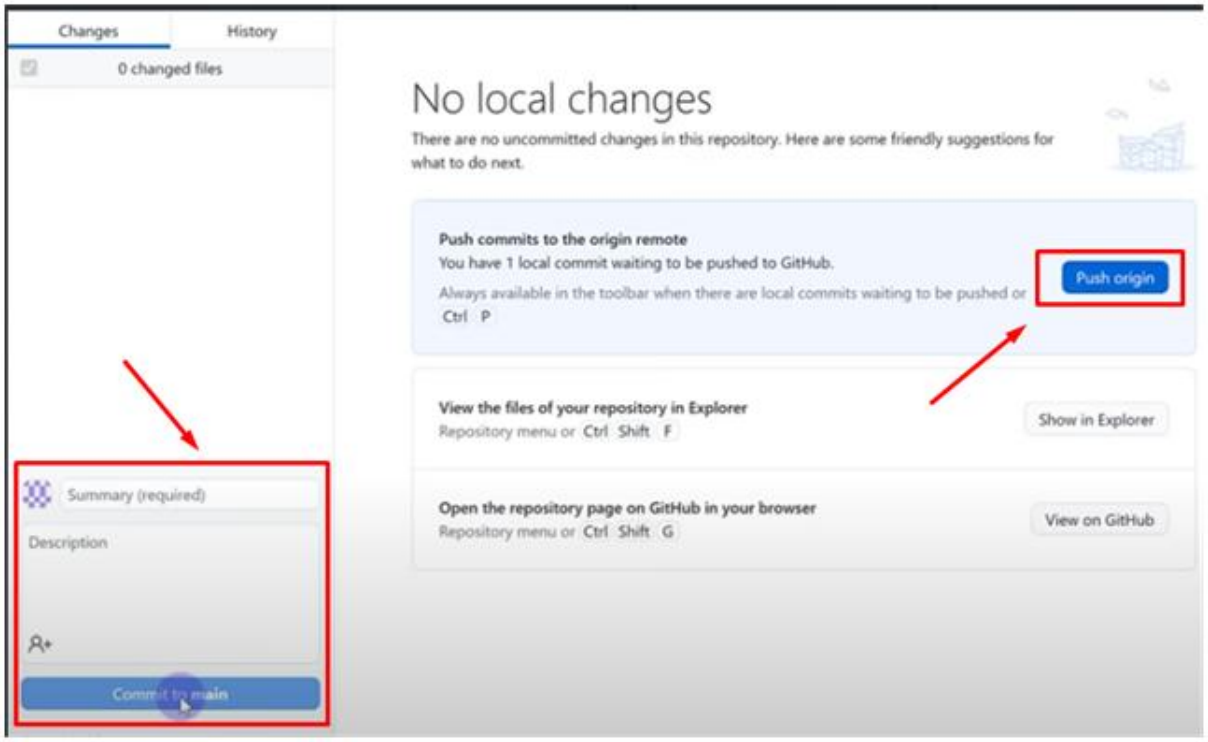

Figura 15 – Preparação para publicação
Fonte: Autoria própria

A partir deste momento da codificação e com o auxílio do **Ambiente de Desenvolvimento Integrado - IDE**, cuja finalidade é um editor de código fonte e no uso do Git, para manter o repositório remoto no GitHub atualizado.

Optamos pela utilização do **Visual Studio Code**, por se tratar de uma ferramenta da Microsoft, além de ser gratuita.

Abra o seu navegador de internet e digite: https://code.visualstudio.com (Figura 16).

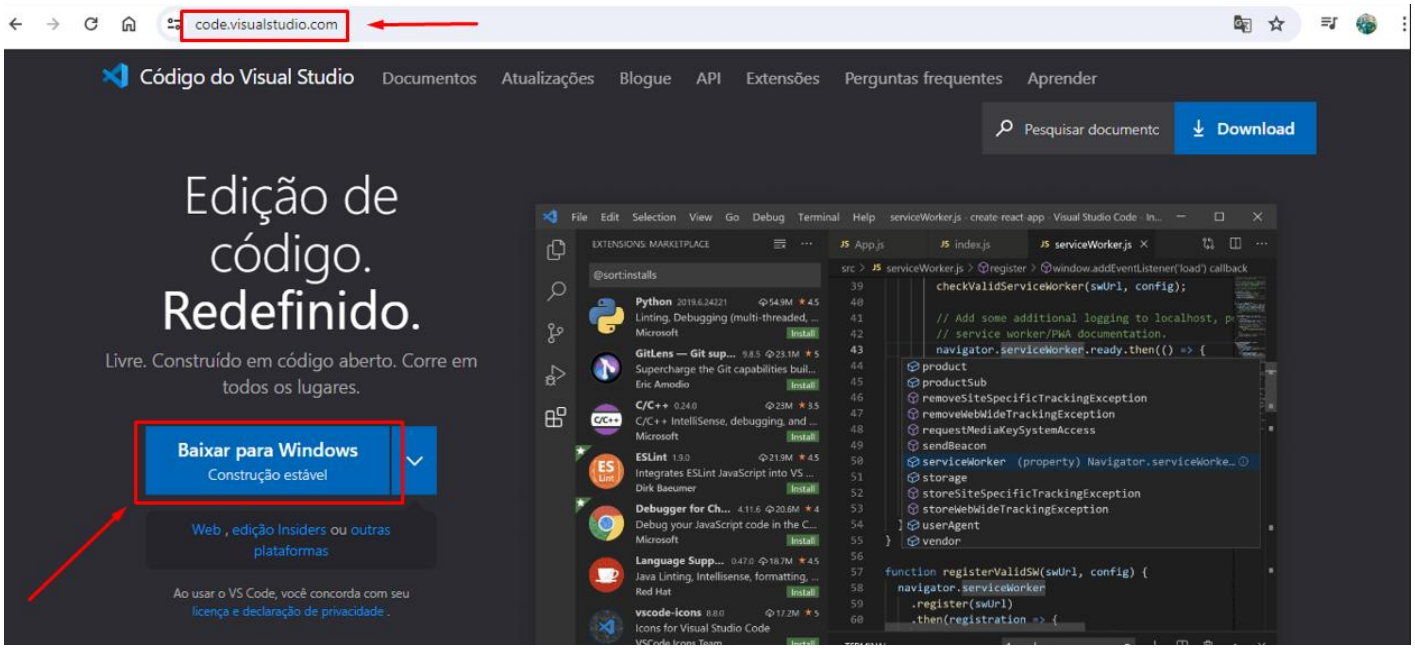

Figura 16 – Página de download do Visual Studio Code
Disponível em: https://code.visualstudio.com Acesso em: 27 de out. 2024

4.2 - **Clique no botão azul** para fazer o download para Windows. O programa reconhece automaticamente qual é o seu sistema operacional. Será uma instalação simples, como qualquer outro software da Microsoft. Além do ícone que será criado na área de trabalho, você também poderá acessá-lo através do **Menu Iniciar/Botão Windows**.

No **Visual Studio Code**, clique em ***File***, depois em ***Open Folder*** (Figura 17).

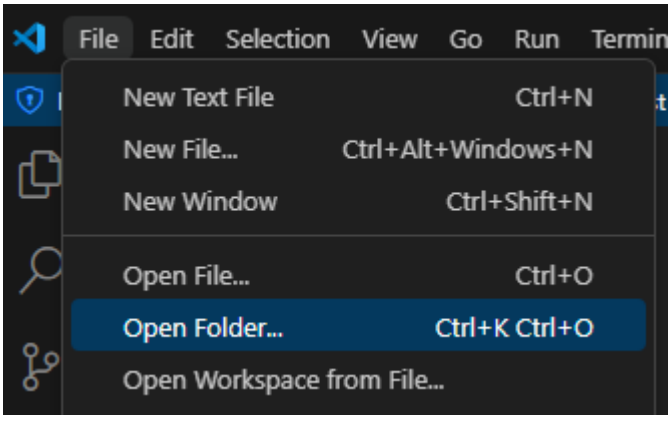

Figura 17 – Acessando o repositório
Fonte: Autoria própria

4.3 - Escolha o repositório local (Figura 18).

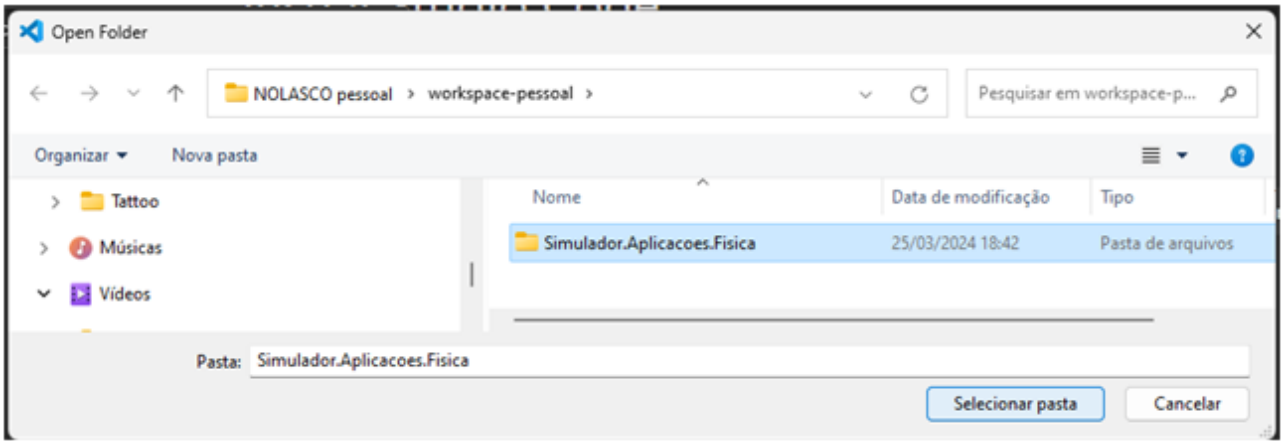

Figura 18 – Repositório local
Fonte: Autoria própria

4.4 – Após clicar no repositório local, você será direcionado para uma tela de visualização de todos os códigos incorporados ao simulador (Figura 19).

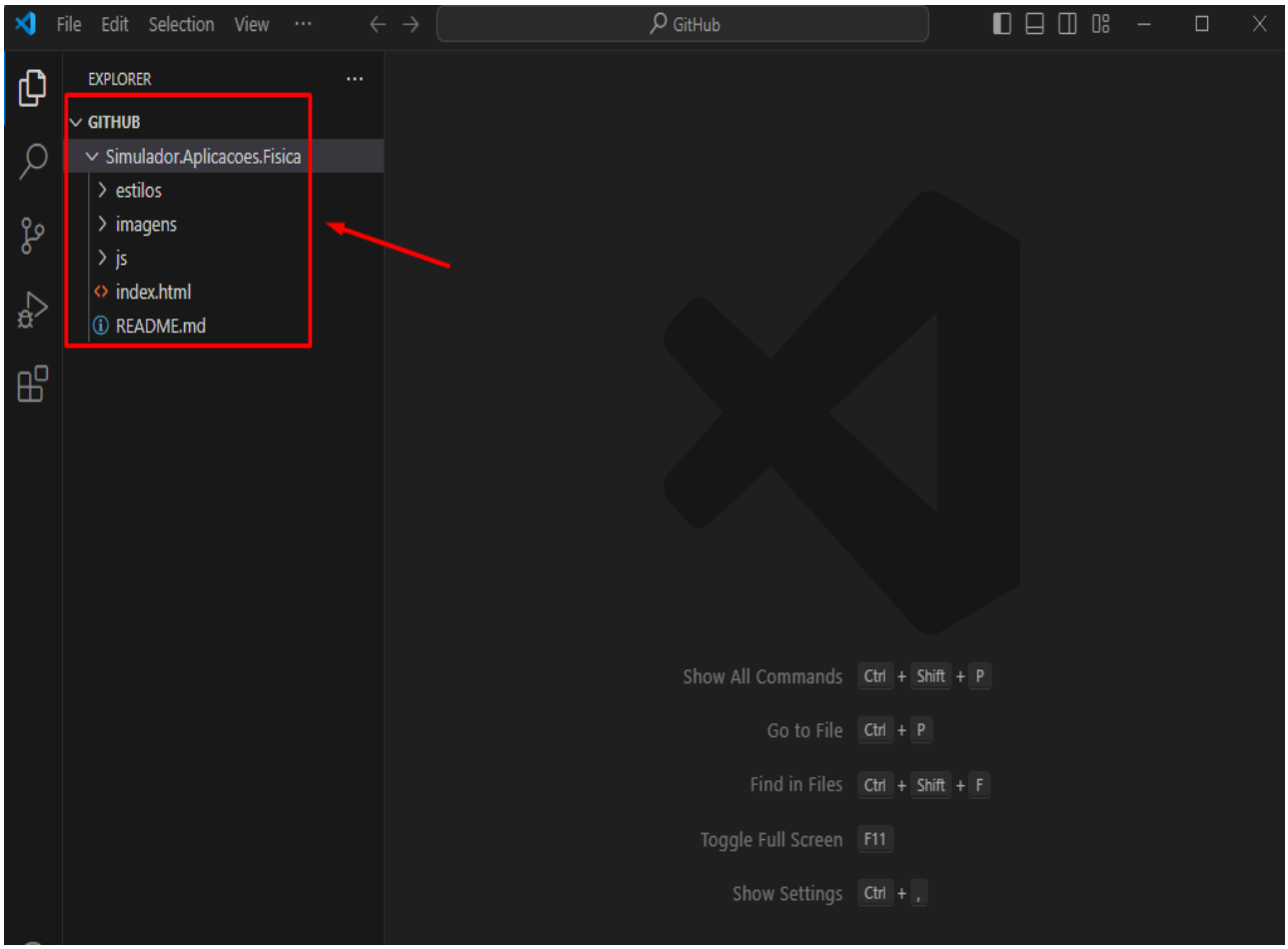

Figura 19 – Códigos do simulador
Fonte: Autoria própria

Perceba, caro usuário/leitor do roteiro, que na Figura 19, existem três pastas, ***"estilos"***, ***"imagens"*** e ***"js"***, e dois arquivos, ***"index.html"*** e o ***"README.md"***. Nestas pastas e arquivos estão contidos todos os códigos de programação, estilos e imagens usadas na criação do simulador.

5 - Codificação do Simulador em HTML 5

Ao abrir o conjunto dos códigos no Visual Studio Code, você tem acesso à edição de todos os arquivos e pastas do simulador. Primeiramente, iremos focar na configuração do arquivo ***"index.html"*** (Figura 20).

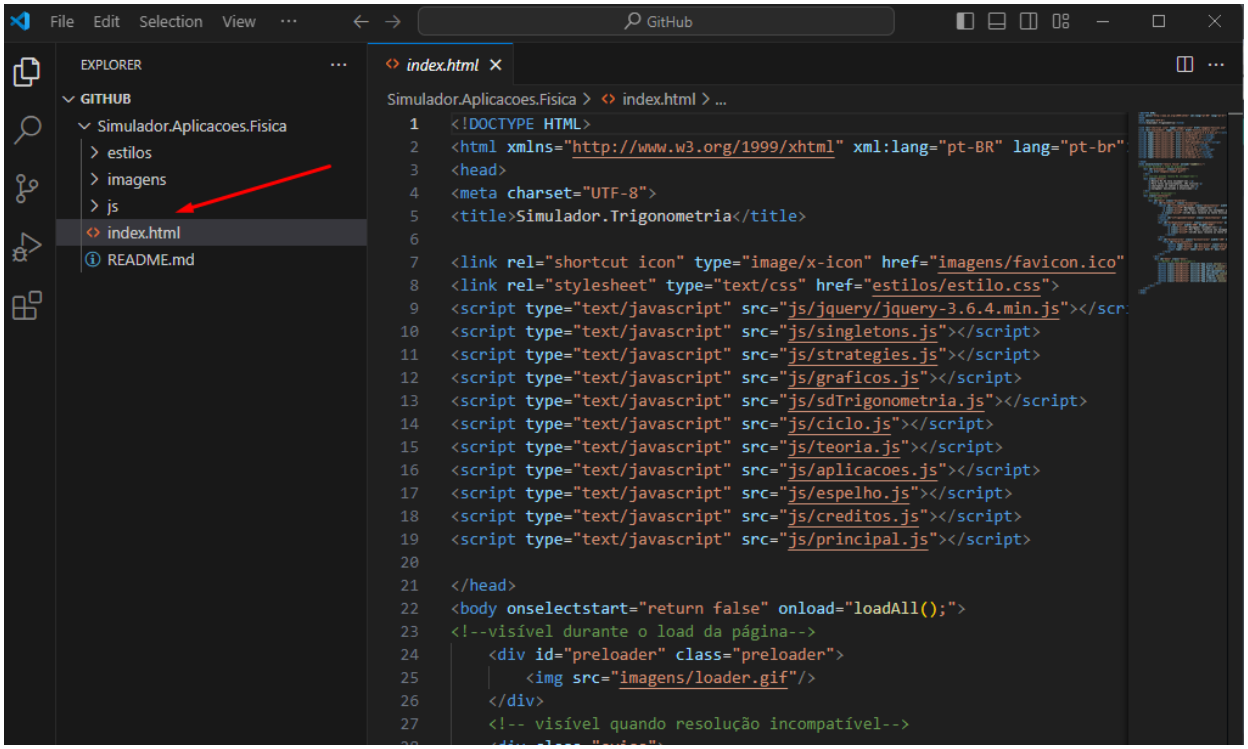

Figura 20 – Arquivo *index.html* do simulador
Fonte: Autoria própria

Página HTML é a base para a exibição do conteúdo. O HiperText Markup Language não é uma linguagem de programação, e sim uma linguagem de marcação e com ele criamos marcações que são interpretadas pelo browser para renderização de elementos na tela, como parágrafos, tabelas, imagens, etc. Ao

manipularmos os objetos nas simulações, o papel da renderização é transformar tudo em uma imagem, para que possamos enxergar o resultado final do projeto.

Desta forma, precisamos conhecer a estrutura mais básica de uma página HTML (Figura 21) (Figura 22).

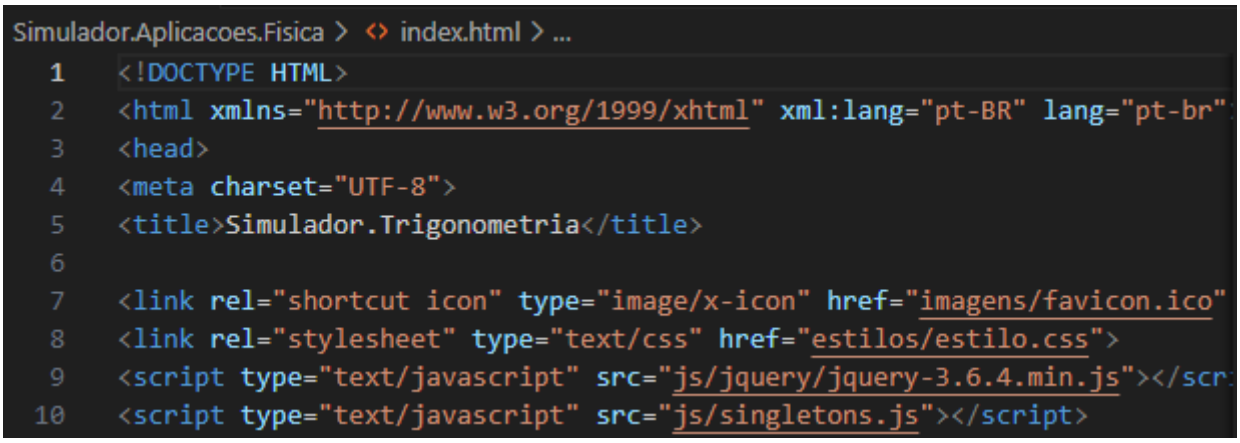

Figura 21 – Codificação em HTML do simulador
Fonte: Autoria própria

Observe que todo documento HTML é iniciado com a "***tag***" ***<html>***. Aliás, em programação é muito comum o uso de ***"tags"***, ou seja, indicadores do que o programa deve executar.

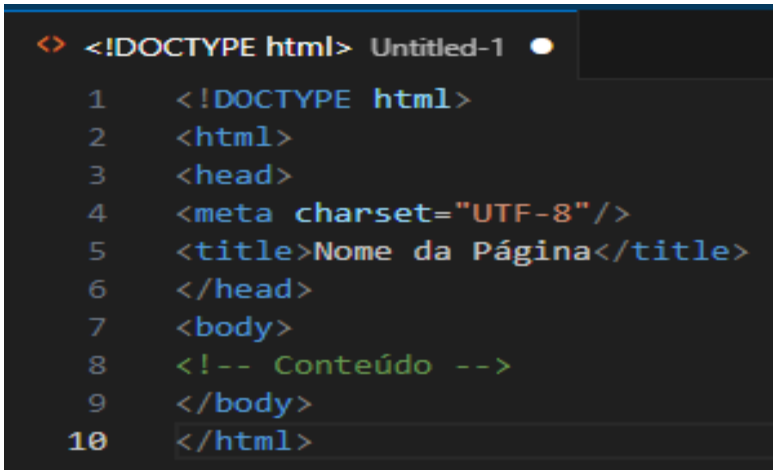

Figura 22 – Exemplo genérico de codificação em HTML
Fonte: Autoria própria

Outra "***tag***" muito comum em documentos HTML é a ***tag <canvas>***, componente do HTML na versão 5, define uma área para desenhar elementos diversos. Com ela, podemos criar imagens variadas e, com alguns recursos de programação, podemos criar animações interativas, como simuladores (Figura 23) e (Figura 24).

```
39              <canvas id="cvTrigonometrando" class="objectCanvas" widt
40                  <p class="avisoP">Navegador incompatível!</p>
41                  <p class="avisoP">Por favor, atualize seu navegador
42                 <p class="avisoP">versão mais recente ou tente utiliz
43              </canvas>
44              <canvas id="cvTrigonometrando2" class="objectCanvas" wid
45              </canvas>
46              <div id="divEspelhoControles" class="espelhoControles" w
47                  <canvas id="plot" width="500" height="450">
48                      <p class="avisoP">Navegador incompatível!</p>
49                      <p class="avisoP">Por favor, atualize seu navega
50                      <p class="avisoP">versão mais recente ou tente u
51                  </canvas>
```

Figura 23 – Códigos do simulador de elementos diversos, *tag* <*canvas*>
Fonte: Autoria própria

```
7    <body>
8    <canvas></canvas>
9    </body>
```

Figura 24 – Exemplo genérico da *tag* <*canvas*>
Fonte: Autoria própria

No caso do simulador aqui apresentado, no arquivo "***index.html***" há diversas outras "***tags***", como exemplificado na imagem a seguir (Figura 25).

```
52                </div>
53                <div id="divControles" class="divControles" width="100" h
54                    <form id="form-counters">
55                        <button type="button" id="btn-minus" class="btn-
56                        <button type="button" id="btn-plus" class="btn-p
57                        <input type="text" name="posy" min="0" max="150"
58                    </form>
59                </div>
60            </div>
61            <div id="menu" class="menu">
62                <!--Botões do menu principal-->
63                <button class="menuButton" onclick="App.ciclo.inicio()">
64                <button class="menuButton" onclick="App.graficos.inicio(
65                <button class="menuButton" onclick="App.teoria.inicio();
66                <button class="menuButton" onclick="App.sdTrigonometria.
67                <button class="menuButton" onclick="App.aplicacoes.inici
68                <button class="menuButton" onclick="App.espelho.inicio()
69                <button class="menuButton" onclick="App.creditos.inicio(
70                <button class="menuButton" onclick="App.principal.inicio
```

Figura 25 – Outras *tags* de codificação utilizadas no simulador
Fonte: Autoria própria

6 - CSS 3 Estilização da página HTML

Outro arquivo componente dos códigos do simulador aqui apresentado é o arquivo "***estilo.css***".

O Cascading Style Sheets ou Folha de Estilos em Cascata, é onde se define estilos para elementos em uma página. O CSS possui uma sintaxe própria, e a forma de incluí-lo na página pode variar em três: Inline, na página ou em arquivo externo.

Inline: o estilo é definido diretamente dentro da ***tag* HTML** onde ele deve ser aplicado. No exemplo abaixo, na ***tag p***, que define um parágrafo, é aplicado o estilo **color: blue**, que define a cor do texto para azul (Figura 26).

```
1   <p style="color: blue;">Lorem ipsum dolor sit amet[...]</p>
```

Figura 26 – Exemplo de estilo de definição de cor
Fonte: Autoria própria

Neste modelo, o estilo definido para a ***tag*** só será aplicado para o conteúdo dentro desse elemento, não atingindo nada externo nem sendo possível reaproveitar declaração do mesmo estilo.

CSS na página: os estilos são declarados de forma global dentro do contexto do documento HTML em questão, podendo ser reaproveitados para diferentes ***tags***, sem necessidade de repetição. No exemplo abaixo, a demonstração de criação do estilo, dentro do elemento ***HEAD*** do HTML, especificado pela ***tag style***, onde é definido que a cor verde deve ser aplicada ao texto de todos os parágrafos criados na página (Figura 27).

```
1   <html>
2       <head>
3           <style>
4               p{
ource Control (Ctrl+Shift+G)
6               }
7           </style>
8       </head>
9       <body>
10          <p>Lorem ipsum dolor sit amet[...]</p>
11          <p>Nunc vulputate interdum sem, eu semper[...]</p>
12      </body>
13  </html>
```

Figura 27 – Exemplo de criação de estilo
Fonte: Autoria própria

Assim, os dois parágrafos definidos no exemplo terão o texto na cor verde. Caso seja necessário que elementos diferentes definidos pela mesma ***tag*** tenham estilos diferentes, existem alguns recursos para isso.

Utilizando o mesmo exemplo, mas diferenciando os dois parágrafos, fazendo com que o primeiro continue em verde, mas o segundo tenha o texto na cor vermelha (Figura 28).

```
<html>
    <head>
        <style>
            .paragrafo1{
                color: green;
            }
            .paragrafo2{
                color: red;
            }
        </style>
    </head>
    <body>
        <p class="paragrafo1">Lorem ipsum dolor sit amet[...]</p>
        <p class="paragrafo2">Nunc vulputate interdum sem, eu semper[...]</p>
    </body>
</html>
```

Figura 28 – Exemplo de criação de estilos de parágrafos
Fonte: Autoria própria

Neste exemplo, definimos **classes de estilo** para serem aplicadas a qualquer elemento onde elas sejam necessárias.

Ainda há outras formas de diferenciar e direcionar blocos de estilo a elementos específicos ou conjuntos de elementos específicos, utilizando especificações como ***name*** da ***tag***, ***id*** da ***tag***, entre outros elementos comuns no uso de identificação de componentes específicos.

CSS em arquivo externo: fica num arquivo com extensão ***.css***, separado da página HTML, e deve ser importado nela para ser aplicado (Figura 29).

```
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="aqui entra o link do seu arquivo externo">
    </head>
```

Figura 29 – Exemplo de arquivo *.css*
Fonte: Autoria própria

A sintaxe neste caso é exatamente igual ao modelo criado dentro da página, mas sem utilizar a ***tag style***, apenas os blocos de estilo.

Algumas vantagens do uso do **CSS** externo são a organização e limpeza do código, sendo que HTML fica separado do **CSS**, ambos mais limpos e mais fáceis de gerenciar, e também o mesmo arquivo ***.css*** pode ser importado em diferentes páginas HTML, aproveitando tudo o que for necessário, além de que mais de um ***.css*** pode ser utilizado numa mesma página (Figura 30).

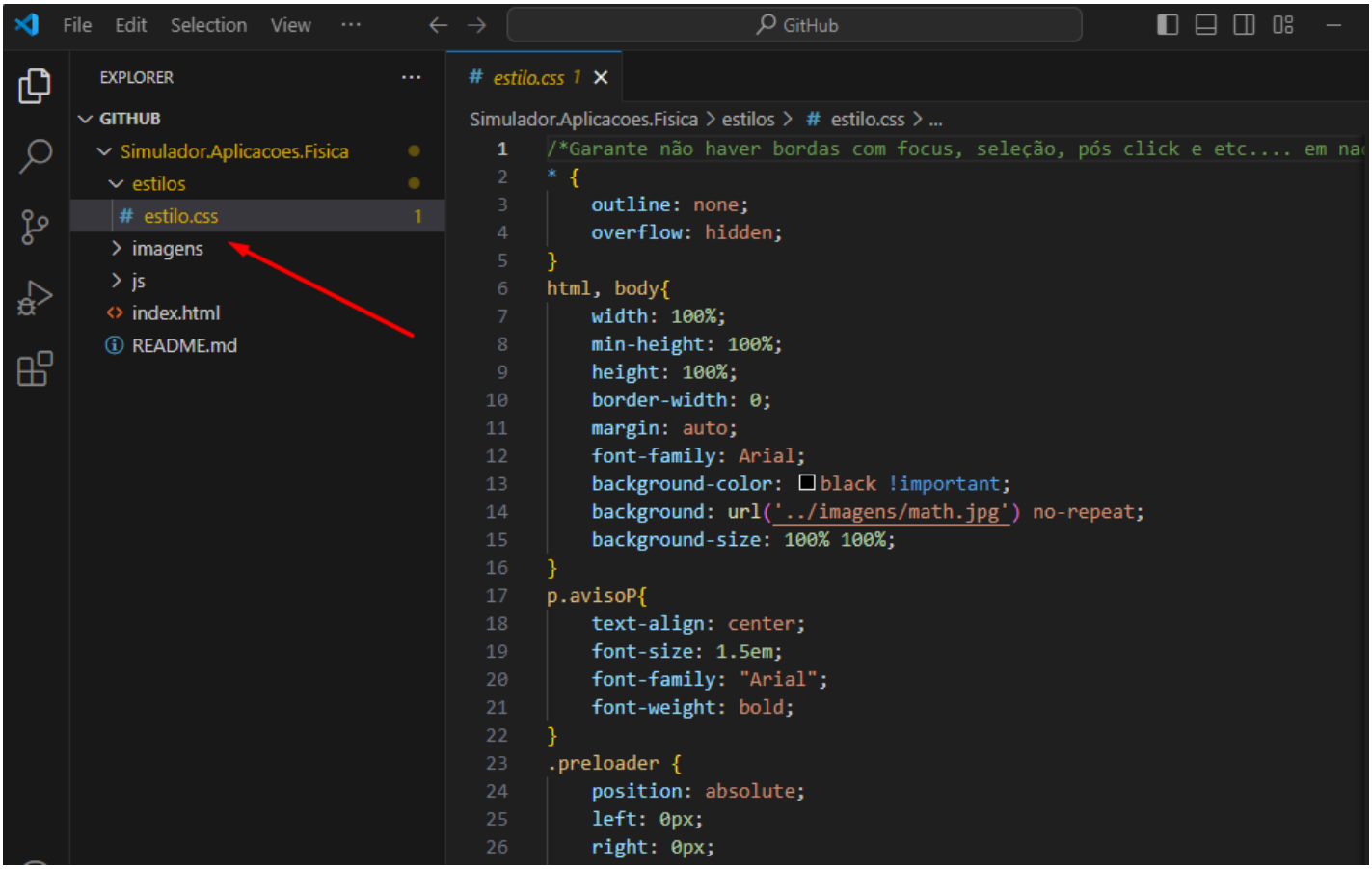

Figura 30 – Códigos *estilo.css* do simulador
Fonte: Autoria própria

7 – JavaScript (JS) Linguagem de Programação. (Não confundir com linguagem JAVA, pois não são relacionadas!)

Por fim, há também nas pastas do simulador, uma série de arquivos com extensão "***.js***", como "***aplicações.js***" e "***ciclo.js***".

JavaScript é uma linguagem de programação executada do lado do cliente (client side), ou seja, no computador do usuário. Essa linguagem é capaz de interagir com os elementos do HTML, processar dados de maneiras variadas, se comunicar com o server side (servidor, em caso de sistemas web mais robustos, diferentes de simples páginas web), entre outros. É uma linguagem muito usual para criação de páginas / sistemas web, para o processamento client side.

O **JS** pode ser escrito, também, dentro da página **HTML**, de maneira similar ao **CSS**, mas entre as ***tags SCRIPT***, também dentro do ***HEAD***, dentro do ***BODY*** ou ao final, depois do fechamento da ***tag BODY***. Tudo depende de como precisa controlar o carregamento desse código em relação a renderização da página (HTML em si) (Figura 31).

```
1  <html>
2    <head>
3      <title>Primeiro JS - Hello World!</title>
4      <script language="javascript" type="text/javascript">
5          alert("Olá Mundo!");
6      </script>
7    </head>
8    <body>
9    </body>
10 </html>
```

Figura 31 – Exemplo genérico de programação em JS
Fonte: Autoria própria

No exemplo abaixo, um script em **JS** para execução de um elemento ***alert*** com a mensagem ***Olá Mundo!*** O resultado, na tela, é o seguinte (Figura 32).

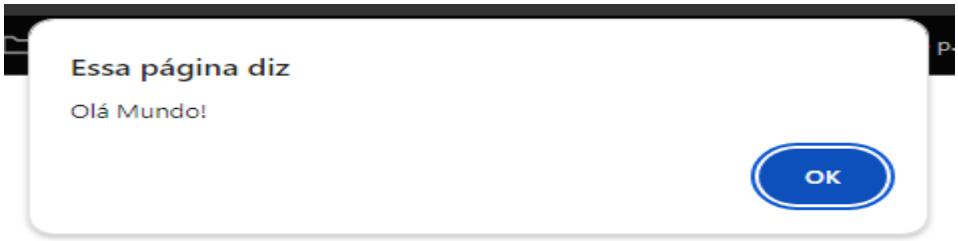

Figura 32 – Exemplo de um script de mensagem
Fonte: Autoria própria

E também de forma similar ao **CSS**, podemos escrever o **JS** em um arquivo separado com extensão **.js** e importar dentro do **HTML**, as mesmas vantagens se aplicam para esse formato, melhor organização, códigos “similares” agrupados, código mais limpo, melhor manutenibilidade, etc (Figura 33).

```
<html>
  <head>
    <script type="text/javascript" src="aqui entra o link do seu arquivo externo"></script>
  </head>
  <body>
  </body>
</html>
```

Figura 33 – Exemplo genérico de programação em JS
Fonte: Autoria própria

Uma biblioteca JavaScript muito utilizada é o **JQuery**. Você pode acessar através do endereço https://jquery.com/. É livre e contém funções JavaScript às quais disponibiliza com uma maneira mais simplificada de escrita e uso. Para utilizar o **JQuery**, é necessário importar ele na sua página também, da mesma forma que importa um arquivo ***js*** ou indicando um link da web onde ele esteja disponível, ou baixando uma versão e incluindo no seu projeto, indicando o link direto do arquivo interno (Figura 34).

```
<script type="text/javascript" src="js/jquery/jquery-3.6.4.min.js"></script>
```

Figura 34 – Exemplo genérico de link direto utilizando o JQuery
Fonte: Autoria própria

No caso do Simulador, podemos utilizar o ***js*** / ***jquery*** para interagir com a ***tag canvas***, além de outros elementos do **HTML** como formulários e botões, para criar as imagens, animações e possibilitar a interação com o simulador. Considere, por exemplo, a ***tag canvas*** definido abaixo (Figura 35).

```
<canvas id="canvasPrincipal"></canvas>
```

Figura 35 – Exemplo de *tag canvas*
Fonte: Autoria própria

Adicionamos um identificador único neste elemento, definido por id = ”nome do identificador único”, para facilitar o acesso ao elemento a partir do ***js***. No código ***js***, podemos obter a referência do elemento com **JQuery** assim (Figura 36):

```
3    var elementoCanvasPrincipal = $("#canvasPrincipal");
```

Figura 36 – Exemplo de identificador único de elemento
Fonte: Autoria própria

Ou, em JavaScript puro, dessa forma (Figura 37):

```
var elementoCanvasPrincipal = document.getElementById("canvasPrincipal");
```

Figura 37 – Exemplo de programação em JS puro
Fonte: Autoria própria

Ambos os códigos têm o mesmo efeito. Perceba como a versão com **JQuey** é mais simples para se escrever.

Nestes dois códigos, estamos basicamente atribuindo à variável **elementoCanvasPrincipal** a referência ao elemento **Canvas** do **HTML** identificado pelo ***id*** **canvasPrincipal**. Dessa forma, podemos interagir com o elemento **Canvas** do **HTML** a partir da variável **elementoCanvasPrincipal**.

Para podermos manipular esse **Canvas** com o ***js***, precisamos seguir algumas regras específicas para esses elementos, com os recursos que o JavaScript disponibiliza para isso. Após obter a referência do **Canvas**, precisamos do contexto gráfico dela, o qual objetos da seguinte forma (Figura 38):

```
var ctx = elementoCanvasPrincipal.getContext("2d");
```

Figura 38 – Exemplo de elementos Canvas no JS
Fonte: Autoria própria

Pronto, agora podemos trabalhar no Canvas.

Desenhando uma linha, como exemplo e os códigos do simulador (Figura 39) e (Figura 40).

```
ctx.moveTo(20,50);//move para o ponto inicial (X,Y)
ctx.lineTo(150,120);//define o desenho de linha até o próximo ponto (X1, Y1)
ctx.stroke();//desenha a linha
```

Figura 39 – Exemplo de criação de uma linha de código de elemento Canvas
Fonte: Autoria própria

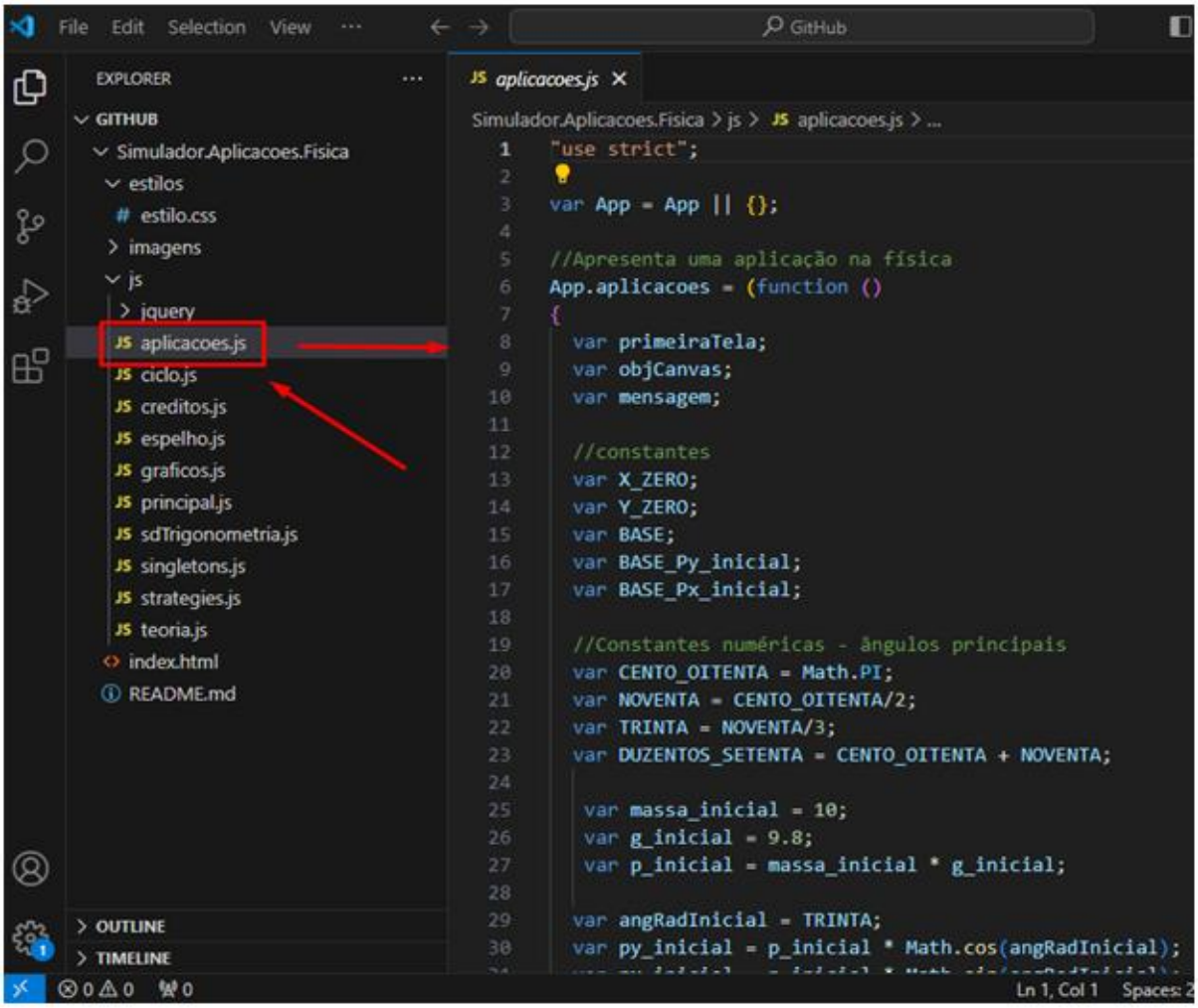

Figura 40 – Códigos *js* do simulador
Fonte: Autoria própria

Vale ressaltar ainda que, há uma pasta relativa às imagens que irão aparecer no simulador quando utilizamos o navegador de internet (Figura 41):

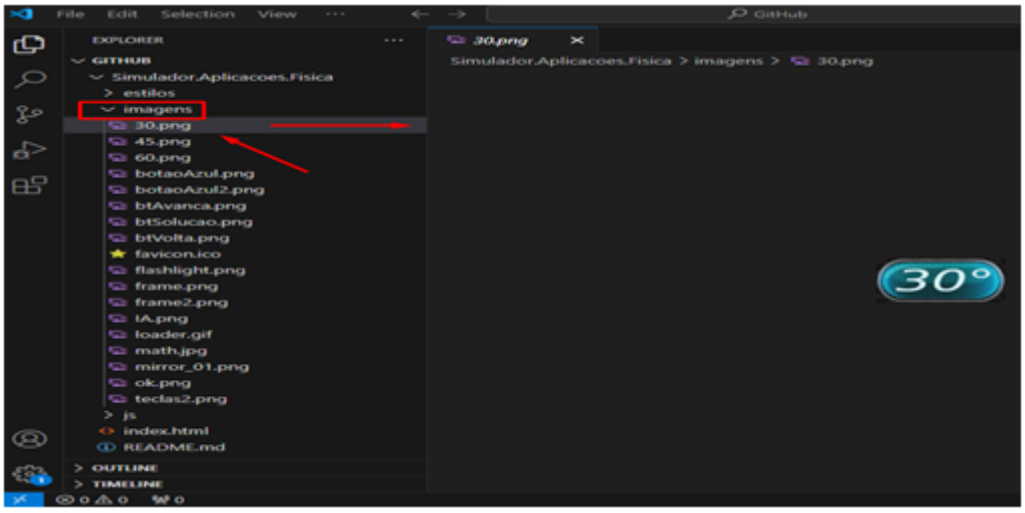

Figura 41 – Arquivos de imagens do simulador
Fonte: Autoria própria

Diante de todo o exposto, caro usuário/leitor, ficam mais perceptíveis os processos de encadeamento lógico das bases de codificação do simulador. Perceba, como mostrado a seguir, no arquivo ***aplicações.js***, estão relacionadas as constantes numéricas e os ângulos principais. "***Var***" significa "***Variável***", que neste caso, do ângulo de ***Cento e Oitenta*** graus, e demais ângulos, que são variáveis angulares. Observe que foi criada uma relação de multiplicidade, ângulos múltiplos de três, para facilitar a construção dos códigos e posterior clonagem destes, para servir de base, nos demais a serem construídos (Figura 42).

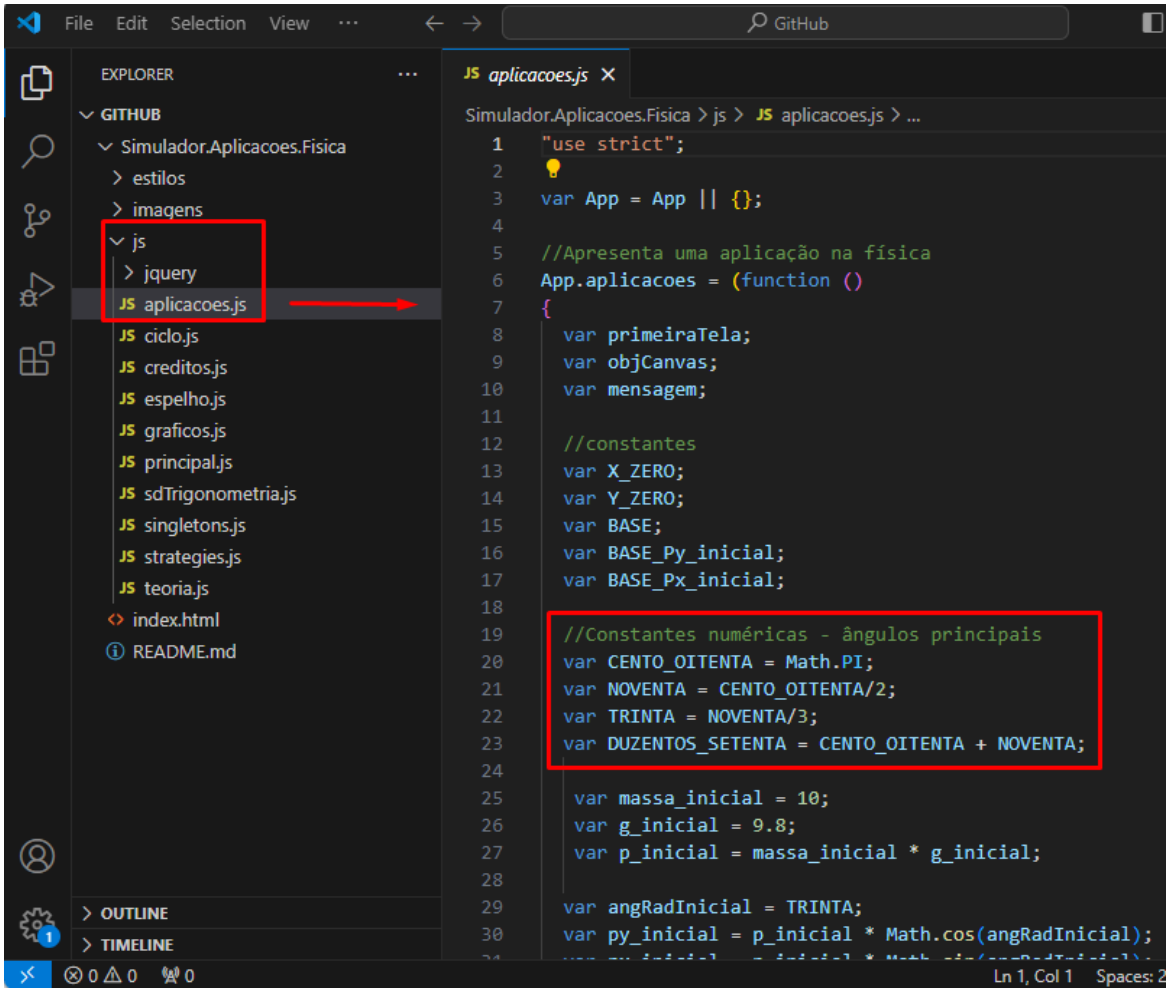

Figura 42 – Códigos das simulações em Matemática
Fonte: Autoria própria

Já nos códigos das simulações de Física, também na pasta de arquivos ***aplicações.js***, coordenadas dos pontos vértices do quadrilátero, podemos observar que as linhas dos códigos são mais extensas. Isso se deve ao fato de cada linha do código programável conter mais de duas operações matemáticas realizáveis. Como mostrado a seguir, "***angRadInicial***" significa "**Ângulo Radial Inicial**", cuja funcionabilidade permite alterar o ângulo da rampa do bloco, através das setas do teclado (Figura 43).

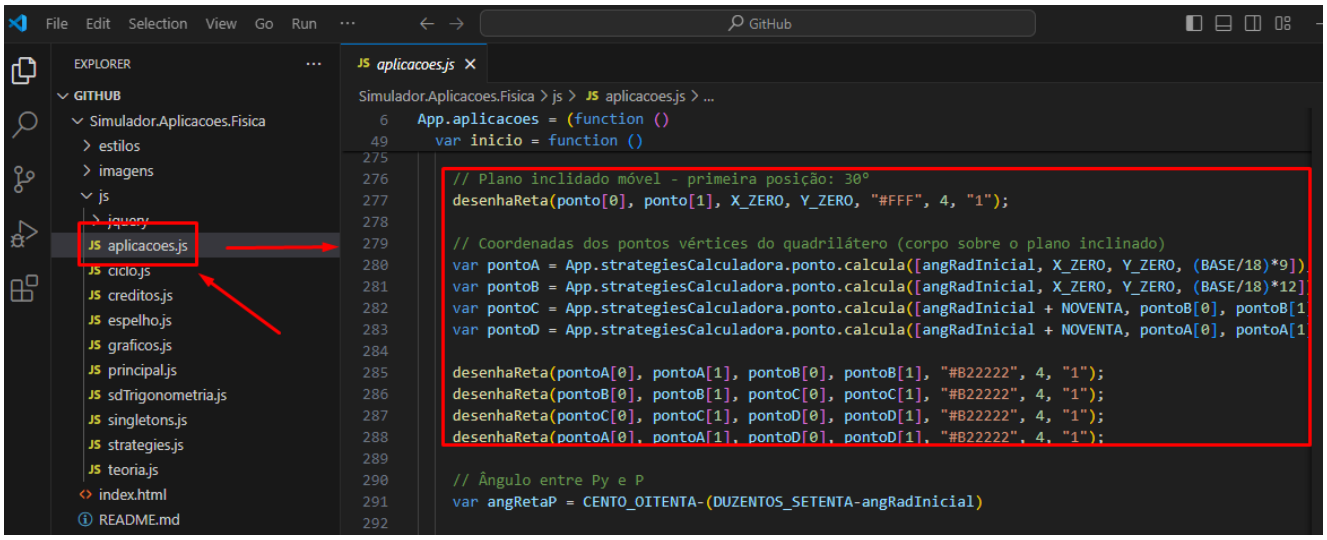

Figura 43 – Códigos das simulações em Física

Fonte: Autoria própria

Com isso, caro leitor/usuário, a ideia demonstrada até o momento é que, partimos de uma linha de programação "base" e esta será o suporte-base de clonagem para a continuidade da codificação programável.

8 – Criando uma Página de Visualização na Web via GitHub (GitHub Pages)

Os passos anteriores viabilizaram o processo de clonagem dos códigos de programação do simulador. Feito isso, daqui em diante, vamos construir uma página de exibição na web, para que o simulador, de fato, execute/exiba as simulações. Esta ação pode ser realizada dentro do próprio **GitHub**, sem a necessidade de adquirir um provedor de hospedagem.

8.1 – Acesse https://github.com/. Clique em "***Entrar***", conforme passo 1.1 (Figura 44).

Figura 44 – Página de acesso ao GitHub
Fonte: Disponível em: github.com Acesso em: 27 de out. 2024

8.2 – Após efetuar seu login, você estará na tela do painel do GitHub. **Clique no repositório dos códigos do simulador que você criou anteriormente** (Figura 45).

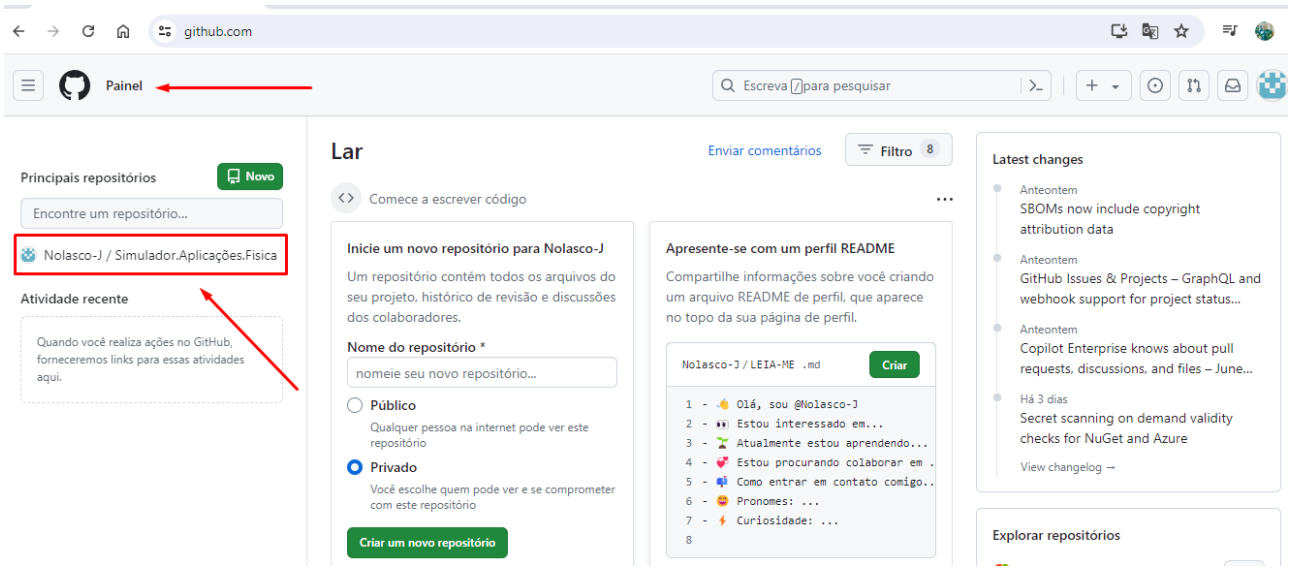

Figura 45 – Painel do repositório do simulador
Fonte: Autoria própria

8.3 – **Clique no repositório criado**. Depois, **clique em** “**Configurações**” (Figura 46).

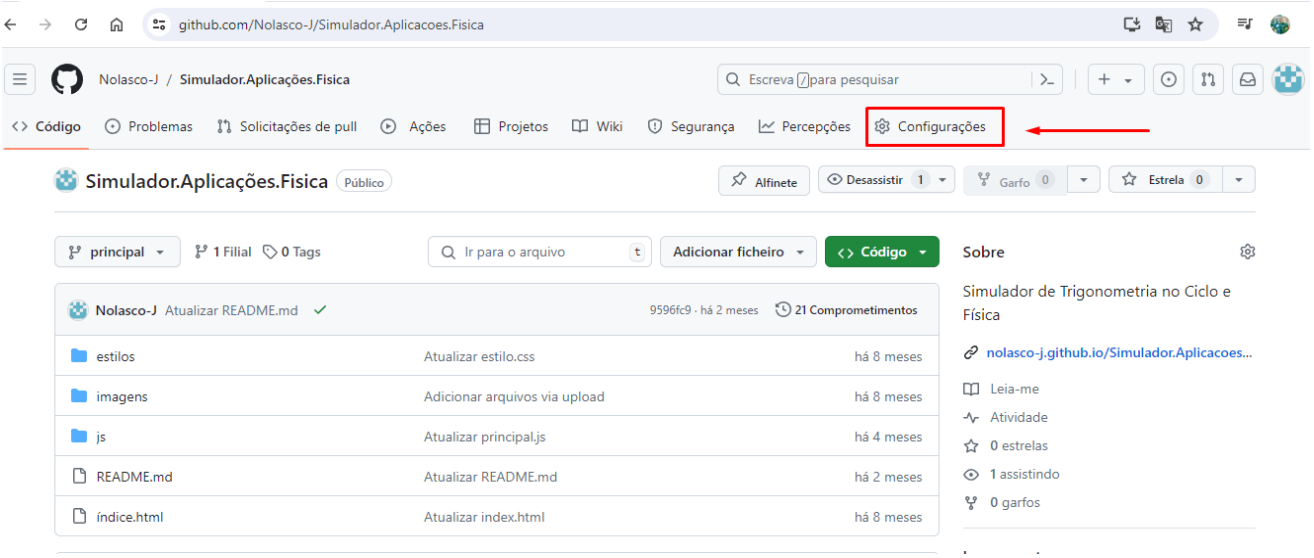

Figura 46 – Painel do repositório e configurações
Fonte: Autoria própria

8.4 – Dentro das **Configurações**, **Clique em** “**Páginas**” (Figura 47).

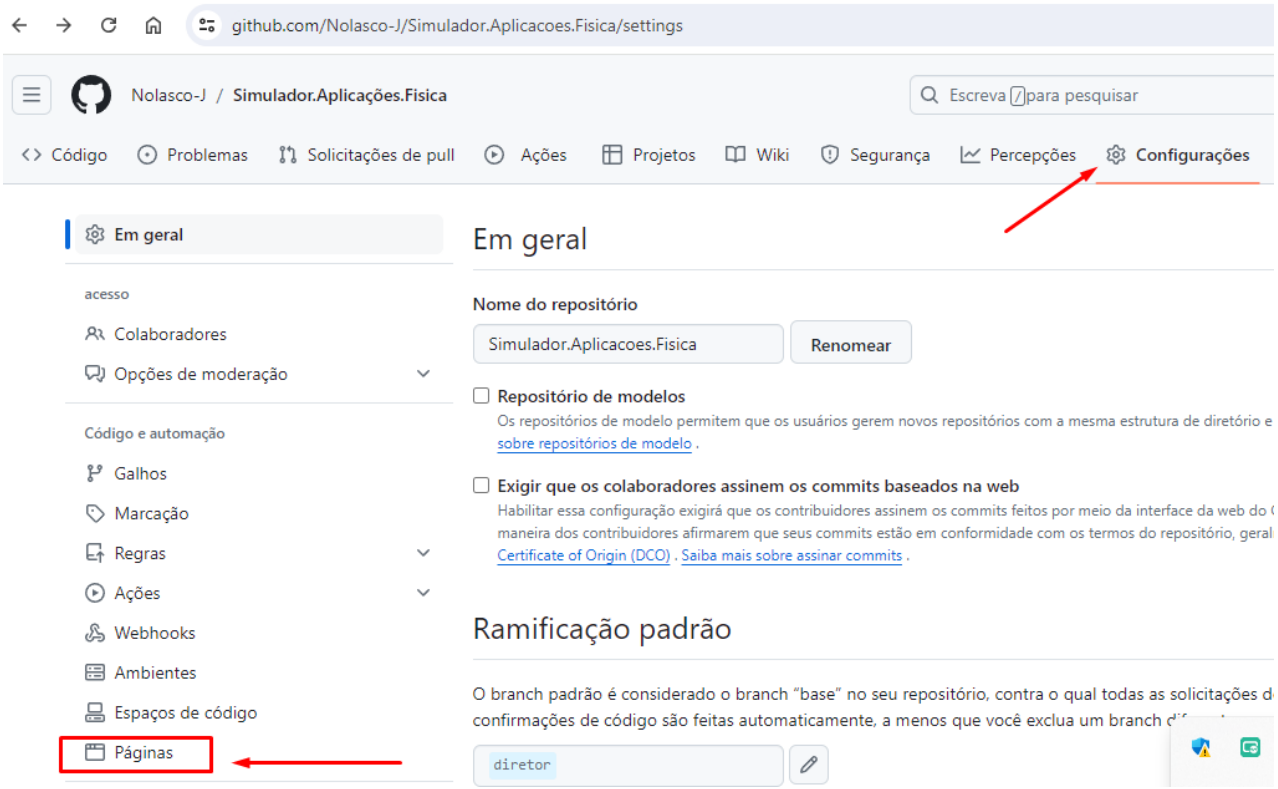

Figura 47 – Painel configurações de páginas
Fonte: Autoria própria

8.5 – **Clique em Páginas**. Certifique-se **PRIMEIRAMENTE** que os campos **Filial** e **Construção e implantação** estão preenchidos conforme a imagem abaixo. (Figura 48)

8.6 - Clique em **Salvar**. Em alguns casos, esta ação não é instantânea! O GitHub Pages pode levar alguns minutos para construir a página de exibição na web. (Figura 48)

8.7 – Após a conclusão de construção da página, **o próprio GitHub** irá utilizar o nome do usuário e a nomeação do repositório para criar o site e disponibilizá-lo na web. O GitHub utiliza a extensão "***.io***" para identificar seu domínio na internet. (Figura 48)

8.8 – Para visualizar seu site criado, **clique no link gerado** ou no botão "**Visite o site**" (Figura 48).

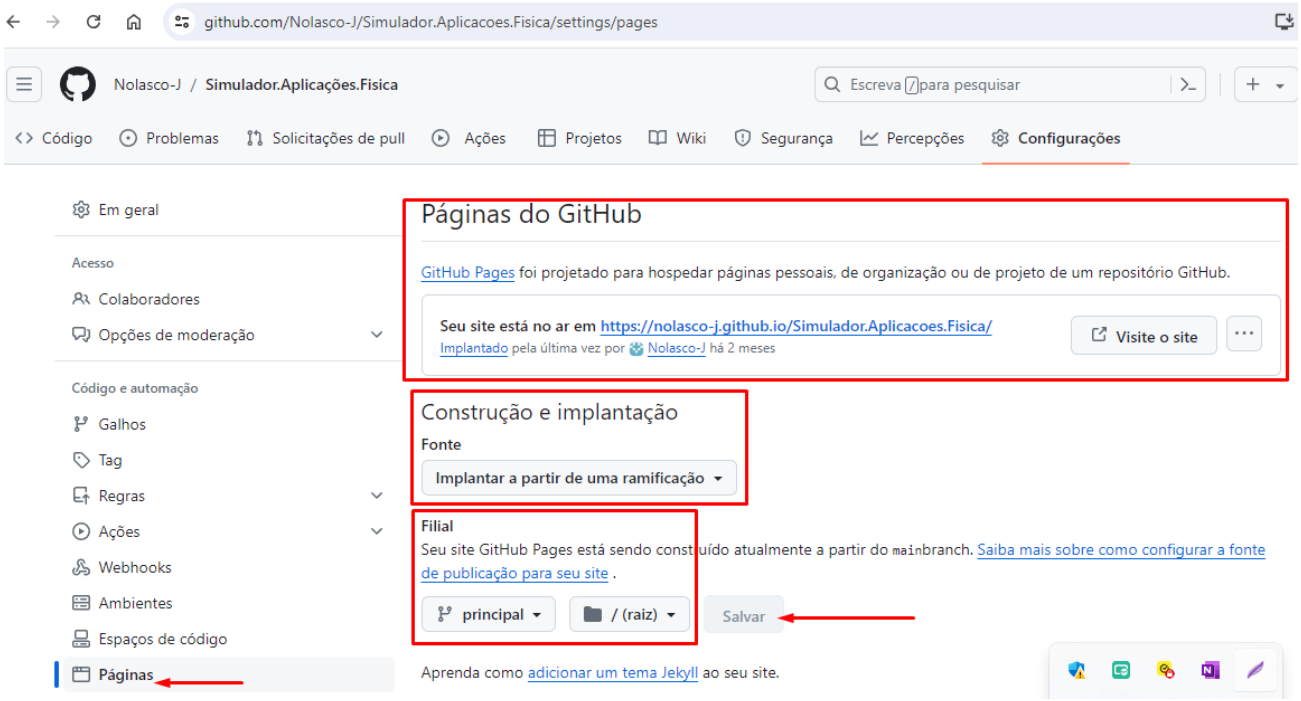

Figura 48 – Criação de páginas no GitHub
Fonte: Autoria própria

Para acessar o ***Simulador Computacional***, desenvolvido pelo **Mestrando NOLASCO**, clique aqui!

Para acessar os ***Códigos de Programação***, desenvolvido pelo **Mestrando NOLASCO**, clique aqui!