

Vitor Amadeu Souza

Introdução ao

Mobile-First

com

React

No ambiente Replit ou CodeSandBox

© 2024 by Cerne Tecnologia e Treinamento Ltda.

© 2024 by Vitor Amadeu Souza

Nenhuma parte desta publicação poderá ser reproduzida sem autorização prévia e escrita de **Cerne Tecnologia e Treinamento Ltda.** Este livro publica nomes comerciais e marcas registradas de produtos pertencentes a diversas companhias. O editor utiliza as marcas somente para fins editoriais e em benefício dos proprietários das marcas, sem nenhuma intenção de atingir seus direitos.

Outubro de 2024

Direitos reservados por:

Cerne Tecnologia e Treinamento Ltda

Produção: Cerne Tecnologia e Treinamento

E-mail da Empresa: cerne@cerne-tec.com.br

Home Page: www.cerne-tec.com.br.com.br

Atendimento ao Consumidor: sac@cerne-tec.com.br

Contato com o Autor: vitor@cerne-tec.com.br



FEITO NO BRASIL

Dedicatória

Aos alunos(as) das graduações de EDC e ADS na disciplina de Mobile-First UI com React 2024.3, que contribuíram significativamente para a elaboração deste material.

“Acredite que você pode, assim você já está no meio do caminho.”

Theodore Roosevelt

Cerne Tecnologia

A Cerne Tecnologia tem uma equipe preparada para desenvolvimento de projetos eletrônicos em diversas áreas: Médica, Entretenimento, Industrial, Robótica, Científica, Automobilística, Aeronáutica, etc. Trabalhamos com tecnologia microcontrolada usando o PIC, ARM, AVR, 8051, dsPIC, PIC24, PIC32 além do Arduino, Raspberry, Beaglebone etc. Desenvolvemos o projeto desde sua concepção até a entrega do produto final, passando pelas etapas de esquema elétrico, protótipo e desenvolvimento de circuito impresso.

Desenvolvemos aplicativos para smartphones/tablets Android, iOS, Blackberry, Windows Phone e no desenvolvimento de softwares a nível PC para plataforma Windows, usando ferramentas como o Visual Basic, C# e C++.

Atuamos na parte de montagem de placas, onde podemos fornecer ambos os serviços de desenvolvimento de projetos e produção ou apenas um destes.

Desenvolvemos esquemas elétricos e layout de PCI, tanto em tecnologia convencional como SMD.

Temos a flexibilidade de customizar um de nossos produtos, de modo a atender a uma necessidade específica do cliente, tornando o custo de desenvolvimento menor se comparado a construção de um projeto desde a sua fase inicial.

Desenvolvemos e fornecemos kits didáticos para diversos microcontroladores além de apostilas, livros e e-books.

Na hora de desenvolver um projeto ou equipar seu laboratório não hesite em nos contatar. Entre em contato conosco através do endereço cerne-tec.com.br para obter mais informações.



Sumário

Capítulo I – Metodologia de desenvolvimento.....	11
Capítulo II – Programação em React e CSS.....	13
1. React.....	13
2. CSS.....	14
3. Criando o primeiro projeto.....	16
4. Modificando o projeto inicial.....	20
5. Comentários.....	21
6. Olá mundo!.....	21
7. Renderizando em duas divs.....	22
8. Alterando a cor da div.....	27
9. Alterando a cor de fundo.....	31
10. Adicionando uma div.....	33
11. Exemplo usando o flexbox.....	37
12. Adicionando um botão.....	41
13. Botão com link.....	46
14. flex-direction.....	50
15. Disposição de elementos.....	51
16. flex-grow.....	55
17. Testando o flex-grow.....	56
18. flex-shrink.....	59
19. Testando o flex-shrink.....	60
20. flex-basis.....	63
21. Exemplo com flex-basis.....	64

22. flex-start, flex-end, center, stretch	70
23. Exemplo aplicando o flex-start, flex-end, center, stretch	71
24. Container aninhado com container.....	75
25. Elementos centralizados em um container.....	80
26. fr.....	84
27. Grid layout.....	85
28. Trabalhando com useState.....	89
29. Saudação personalizada.....	93
30. Calculadora com React.....	98
31. Calculadora com React II.....	105
32. Centralizar um botão.....	111
33. Layout de Navbar Horizontal.....	114
34. Layout de Navbar Horizontal com logo.....	117
35. Layout de Navbar Horizontal com logo e texto.....	120
36. Distribuição de Espaço Igual.....	126
37. Alinhamento de Itens no Centro.....	129
38. Disposição em Coluna.....	132
39. Alinhamento de Itens no Final.....	136
40. Alinhamento ao Centro com Espaço entre Itens.....	139
41. Itens com Largura Fixa e Flexível.....	142
42. Itens com Flex Wrap.....	145
43. Gerando um artefato.....	149

44. Interface com Header, Sidebar, Content e Footer.....	150
45. Testando flex-wrap através de box.....	154
46. Carregando imagens em boxes.....	158
47. em	161
48. rem	162
49. Carrossel de imagens.....	163
50. Tela responsiva I.....	168
51. Exemplo de imagem e texto responsivo.....	172
52. Layout responsivo com flexbox.....	176
53. Tipografia fluida.....	178
54. Imagens responsivas.....	182
55. Tela responsiva II.....	184
56. Tela responsiva com vídeo.....	193
57. Tela responsiva em uma div.....	202
58. Jogo do clique no quadrado.....	206
59. Layout com Header, Main, Aside e Footer.....	211
60. Layout flutuante.....	215
61. Dropdown.....	219
62. Navbar responsivo.....	223
63. Menu hambúrguer.....	228
64. Dark mode.....	233
65. Header, boxes e footer	237

66. Texto centralizado e responsivo.....	241
67. Header, boxes e footer II.....	244
68. Carregando imagem em um container.....	248
69. Imagem responsiva.....	252
70. CSS inline.....	256
71. CSS externo.....	259
72. Módulo CSS.....	263
73. CSS in JS.....	267
74. Componente Card.....	270
75. Componente Button.....	275
76. Componente ToggleSwitch.....	281
77. Componente DynamicBox.....	286
78. Componente HoverButton.....	290
79. Componente ProgressBar I.....	294
80. Componente ProgressBar II.....	298
81. Componente UserCard.....	303
82. Componente TabNavigator.....	311
83. Componente ColorPalette.....	317
84. Componente PricingCard.....	323
85. Componente Accordion.....	331
86. Componente ThemeToggle.....	340
87. Componente Form.....	347

88. Componente ToolTip.....	358
89. Plotando uma função linear com o componente Line.....	365
90. Plotando uma parábola com o componente Line.....	370
91. Plotando uma função constante com o componente Line.....	375
92. Plotando uma função cúbica com o componente Line.....	379
93. Plotando uma Gaussiana com o componente Line.....	384
94. Plotando uma função sigmoide com o componente Line.....	389
95. Plotando uma função módulo com o componente Line.....	394
96. Plotando uma função raiz com o componente Line.....	399
97. Plotando uma função seno com o componente Line.....	403
98. Plotando uma função racional com o componente Line.....	409
99. Plotando uma função degrau com o componente Line.....	413
100. Plotando uma função logaritmo com o componente Line.....	417
101. Plotando uma função exponencial com o componente Line.....	422
102. VS Code.....	428
103. Biblioteca Material-UI.....	429
104. Biblioteca Chakra UI.....	434
105. Biblioteca Blueprint.....	438
106. Biblioteca visx.....	442
107. Biblioteca Fluent.....	448
108. Biblioteca Semantic UI React.....	454
109. Biblioteca React Joyride.....	459

110. Tela responsiva III.....	464
111. Tela de Artistas Clássicos Brasileiros.....	472
112. Media queries.....	482
113. Mobile simulator.....	484
114. Exemplo media query I.....	485
115. Exemplo media query II.....	489
116. Exemplo media query III.....	491
117. Pontos de ruptura.....	494

Capítulo I

Metodologia de desenvolvimento

1. Introdução

A revolução digital trouxe consigo a necessidade de uma abordagem mais eficiente e centrada no usuário para o desenvolvimento web. À medida que os dispositivos móveis se tornaram a principal forma de acesso à internet, a metodologia Mobile-First emergiu como uma estratégia essencial para criar experiências otimizadas e responsivas. Este livro, Introdução ao Mobile-First com React, tem como objetivo ser um guia através dos princípios fundamentais dessa abordagem, utilizando a poderosa biblioteca React.

Neste material, exploraremos a importância de projetar para dispositivos móveis em primeiro lugar, garantindo que suas aplicações sejam não apenas funcionais, mas também intuitivas e atraentes em qualquer tela. Ao longo dos capítulos, abordaremos desde os conceitos básicos do React, suas funcionalidades e componentes, até técnicas avançadas de estilização com CSS, Flexbox e Grid Layout, que são fundamentais para a criação de interfaces responsivas.

Através de exemplos práticos e exercícios, o leitor(a) aprenderá a implementar layouts responsivos em ambientes de desenvolvimento como Replit e CodeSandbox, permitindo testar e refinar suas aplicações em tempo real. Cada seção foi cuidadosamente elaborada para construir uma compreensão sólida das melhores práticas de desenvolvimento, levando em consideração não apenas a estética, mas também a usabilidade e a performance.

Prepare-se para mergulhar no mundo do desenvolvimento Mobile-First com React, onde cada linha de código se transforma em uma experiência interativa e envolvente para o usuário. Seja você um desenvolvedor iniciante ou um profissional em busca de aprimoramento, este livro será um recurso valioso em sua jornada de aprendizado. Vamos juntos transformar a forma como interagimos com o mundo digital!

Capítulo II

Programação em React

1. React

React é uma biblioteca JavaScript de código aberto para a construção de interfaces de usuário, desenvolvida e mantida pelo Facebook. Criada para resolver a complexidade da construção de interfaces interativas, React permite que os desenvolvedores criem componentes reutilizáveis e gerenciem o estado da aplicação de forma eficiente.

A principal característica do React é sua abordagem baseada em componentes, onde a interface do usuário é dividida em pequenos blocos independentes chamados de "componentes". Cada componente pode manter seu próprio estado e lógica, além de receber dados através de propriedades (props). Isso promove uma maior modularidade e facilita a manutenção e a escalabilidade do código.

React utiliza um conceito chamado de "Virtual DOM" (Document Object Model virtual) para otimizar a atualização da

interface do usuário. Em vez de atualizar diretamente o DOM real, React faz alterações no Virtual DOM e, em seguida, calcula a diferença entre o Virtual DOM e o DOM real. Só as partes que realmente mudaram são atualizadas, o que melhora a performance da aplicação.

Além disso, React é frequentemente utilizado com outras bibliotecas e ferramentas, como React Router para roteamento e Redux para gerenciamento de estado global, o que permite construir aplicações web complexas e altamente interativas. Com uma grande comunidade de desenvolvedores e um ecossistema rico, React se tornou uma escolha popular para o desenvolvimento de aplicações modernas e responsivas.

2. CSS

CSS, ou Folhas de Estilo em Cascata, é uma linguagem usada para definir a apresentação visual de documentos escritos em HTML ou XML. Sua principal função é separar a estrutura do conteúdo da sua aparência, facilitando a criação e a manutenção de estilos consistentes e atraentes para páginas web.

Com CSS, os desenvolvedores podem aplicar estilos a elementos HTML utilizando seletores e propriedades. Por exemplo, é possível definir a cor do texto, o tamanho da fonte, e o espaçamento entre os elementos. Além disso, CSS oferece técnicas avançadas para layout e posicionamento, como o modelo de caixas, flexbox e grid, que ajudam a organizar e alinhar elementos de maneira eficiente.

Uma das características mais importantes do CSS é a capacidade de criar layouts responsivos usando media queries, que ajustam o design de acordo com o tamanho da tela e o dispositivo usado. Isso é essencial para garantir que os sites sejam visualmente atraentes e funcionais em qualquer dispositivo, desde smartphones até desktops.

CSS também permite a aplicação de estilos avançados, como gradientes, sombras, e animações, proporcionando uma experiência visual rica e interativa. A linguagem suporta a herança de estilos e a cascata, o que significa que estilos podem ser aplicados em níveis hierárquicos e regras mais específicas podem sobrescrever regras mais gerais.

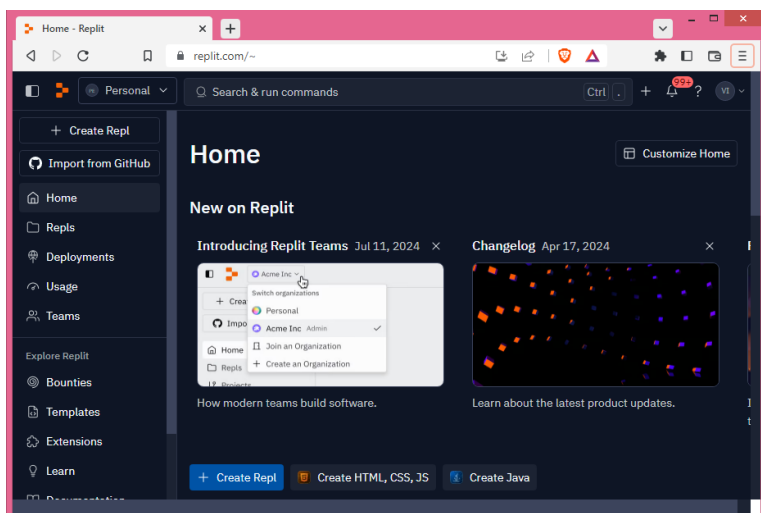
Em resumo, CSS é fundamental para o desenvolvimento web, oferecendo ferramentas para criar designs visuais bonitos e funcionais, enquanto facilita a manutenção e a organização do código.

3. Criando o primeiro projeto

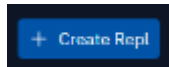
Para o desenvolvimento de React com CSS, foi utilizado o ambiente Replit, que pode ser acessado através do link abaixo.

<https://replit.com/>

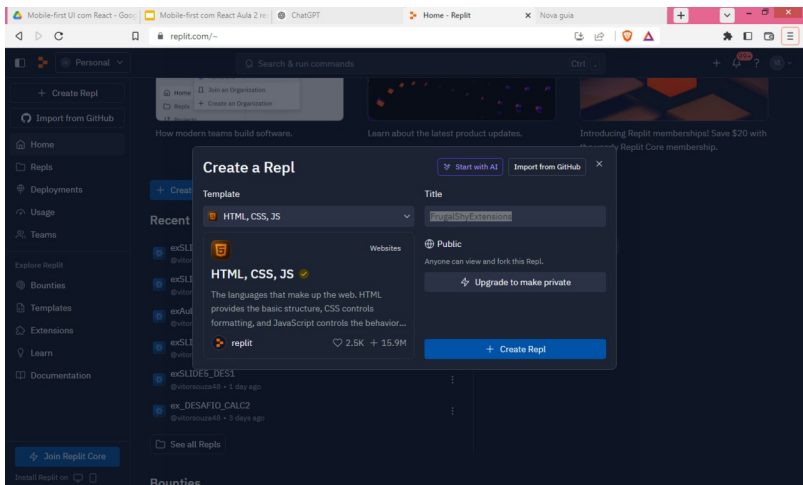
Feito isso, faça o cadastro e login na plataforma. Após este passo, a seguinte tela será apresentada.



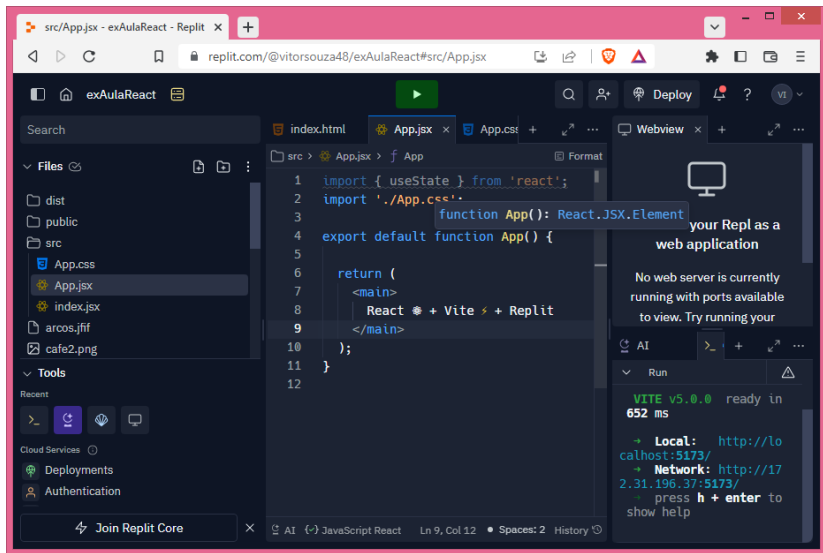
Clique no botão abaixo para criar um novo projeto.



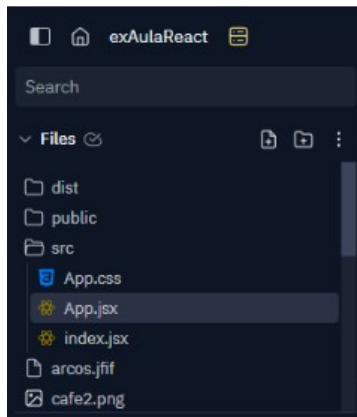
Teremos a próxima tela.



Em template escolha a opção React Javascript. Forneça um nome em title e clique no botão + Create Repl para finalizar a criação do projeto. A tela ficará conforme imagem abaixo.



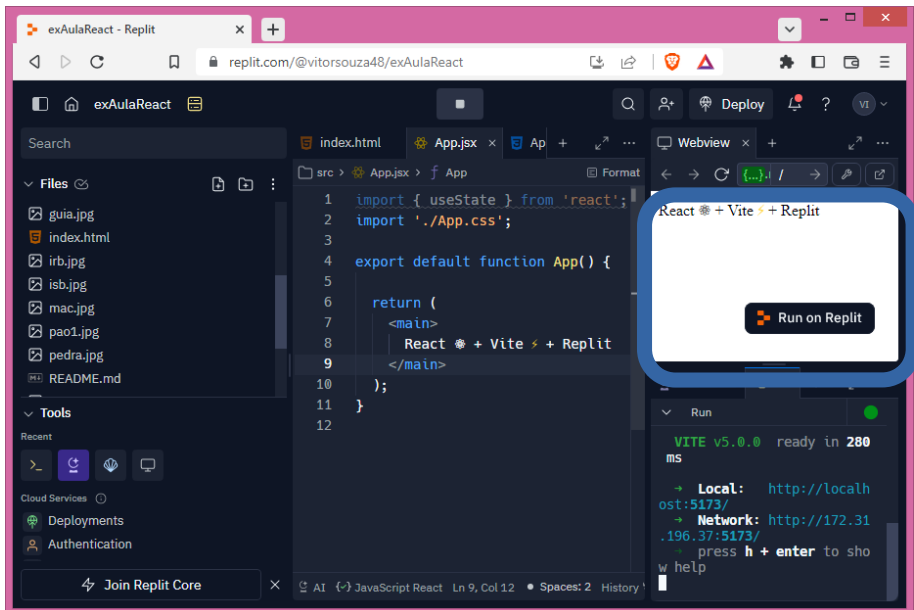
Na aba Files temos os arquivos principais do projeto, em especial o App.jsx onde há a escrita da lógica do programa em React e no arquivo App.css a parte referente a estilização do front-end.



Normalmente iremos trabalhar com estes arquivos. É importante frisar que no arquivo `index.html` há uma div chamada `root`, onde nesta região o React é renderizado. Para iniciar a transpilação e teste do programa padrão, clique no botão Run.



O resultado será:



Para testar o programa em uma tela, clique no botão New tab.



A programação no React sempre é feita no arquivo com a extensão .jsx.

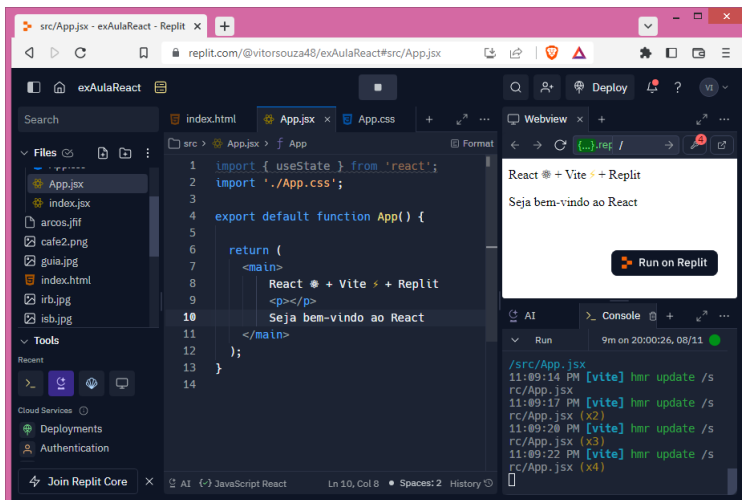
Outra possibilidade é usar a plataforma CodeSandBox, que pode ser acessada pelo link:

<https://codesandbox.io/>

No geral o processo de criação de projeto é parecido.

4. Modificando o projeto inicial

Inicialmente, vamos adicionar o texto “Seja bem-vindo ao React” no código apresentado inicialmente. Para isso, vá até o arquivo App.jsx e faça a mudança apresentada abaixo.



```
1 import { useState } from 'react';
2 import './App.css';
3
4 export default function App() {
5
6   return (
7     <main>
8       React + Vite + Replit
9       <p></p>
10      Seja bem-vindo ao React
11    </main>
12  );
13 }
14
```

Run on Replit

Console

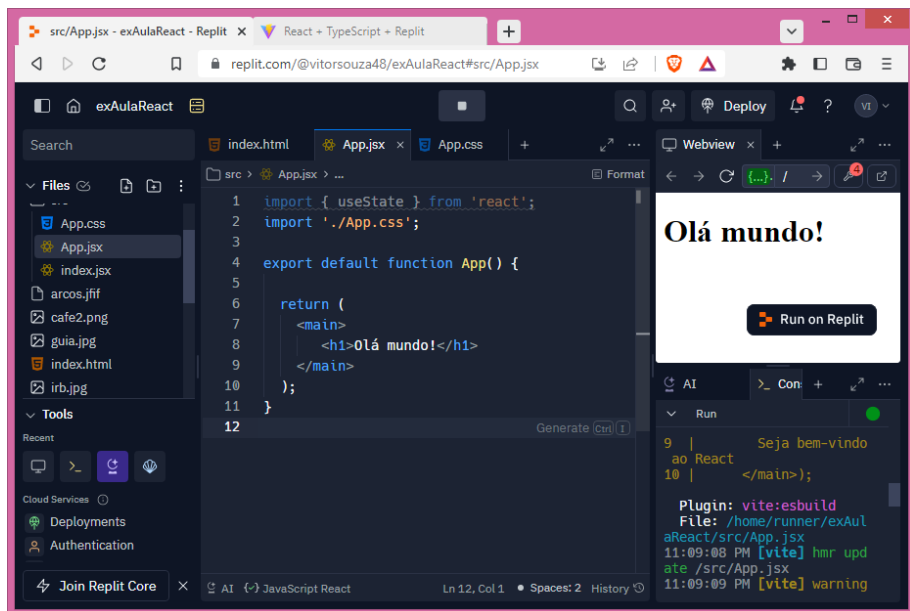
```
/src/App.jsx
11:09:14 PM [vite] hmr update /s
rc/App.jsx
11:09:17 PM [vite] hmr update /s
rc/App.jsx (x2)
11:09:20 PM [vite] hmr update /s
rc/App.jsx (x3)
11:09:22 PM [vite] hmr update /s
rc/App.jsx (x4)
```

5. Comentários

No React, os comentários são feitos de maneira semelhante ao JavaScript, mas com algumas nuances devido à combinação de HTML e JavaScript no JSX. Dentro do JSX, os comentários são inseridos usando `{/* ... */}`, pois o JSX não reconhece diretamente os comentários de HTML (`<!-- ... -->`) ou de JavaScript (`//` ou `/* ... */`). Por exemplo, dentro de um componente React, você pode comentar uma linha dentro do JSX usando `{/* Este é um comentário dentro de JSX */}`. Fora do JSX, ou seja, nas partes que são puro JavaScript, você pode utilizar os comentários padrão de JavaScript, como `//` para comentários de uma linha ou `/* ... */` para comentários de várias linhas.

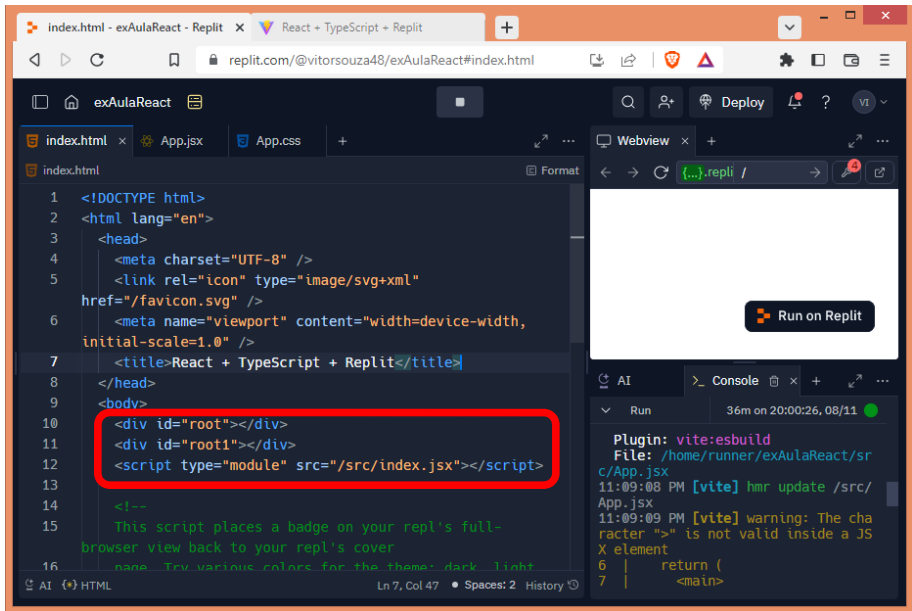
6. Olá mundo!

A proposta deste exemplo é apresentar o texto “Olá mundo!” fazendo uso do cabeçalho `<h1>`. Acompanhe o código `App.jsx`.

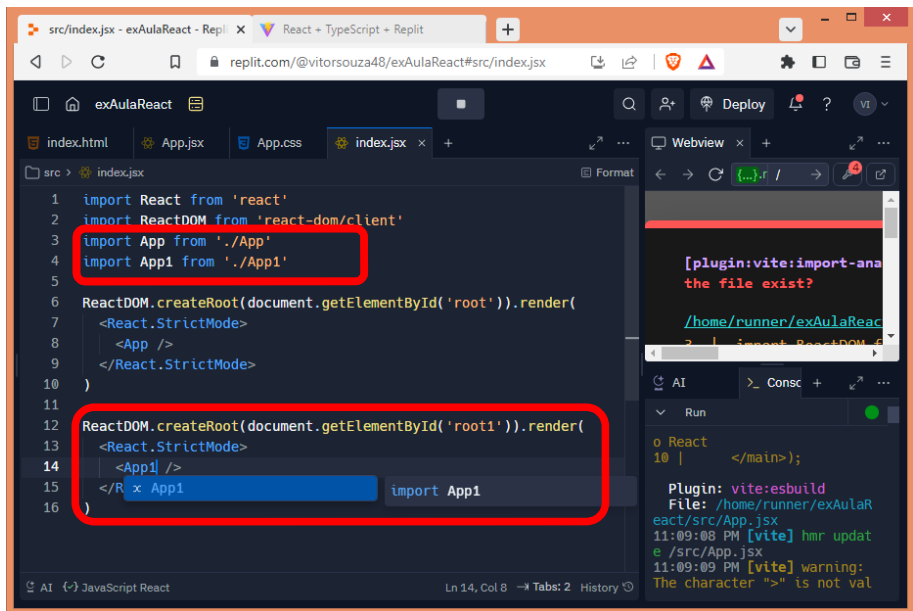


7. Renderizando em duas divs

Neste projeto vamos criar duas divs no index.html, chamadas de root e root1 e em seguida apresentar em root a mensagem “Olá mundo!” e em root1 “Bem-vindo ao React!”. O primeiro passo, consiste em adicionar a div para renderização em index.html.



O segundo passo é ir até o arquivo index.jsx e adicionar mais um elemento ReactDOM fazendo a chamada ao componente App1. Não esqueça de importar o arquivo App1.jsx para o projeto.

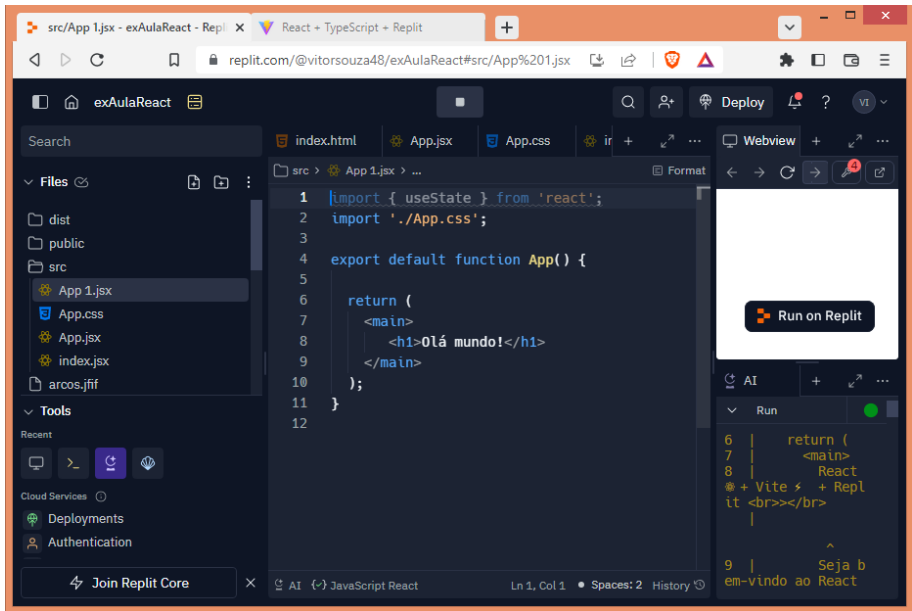


```
1 import React from 'react'
2 import ReactDOM from 'react-dom/client'
3 import App from './App'
4 import App1 from './App1'
5
6 ReactDOM.createRoot(document.getElementById('root')).render(
7   <React.StrictMode>
8     <App />
9   </React.StrictMode>
10 )
11
12 ReactDOM.createRoot(document.getElementById('root1')).render(
13   <React.StrictMode>
14     <App1 />
15   </React.StrictMode>
16 )
```

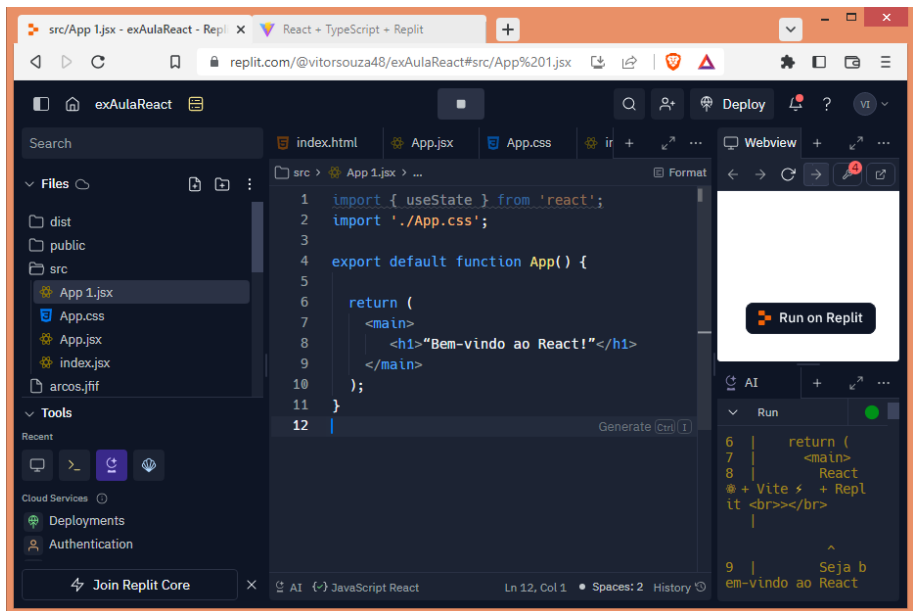
Terminal output:

```
[plugin:vite:import-ana
the file exist?
/home/runner/exAulaReact
2 1 import ReactDOM
o React
10 | </main>);
Plugin: vite:esbuild
File: /home/runner/exAulaR
eact/src/App.jsx
11:09:08 PM [vite] hmr updat
e /src/App.jsx
11:09:09 PM [vite] warning:
The character ">" is not val
```

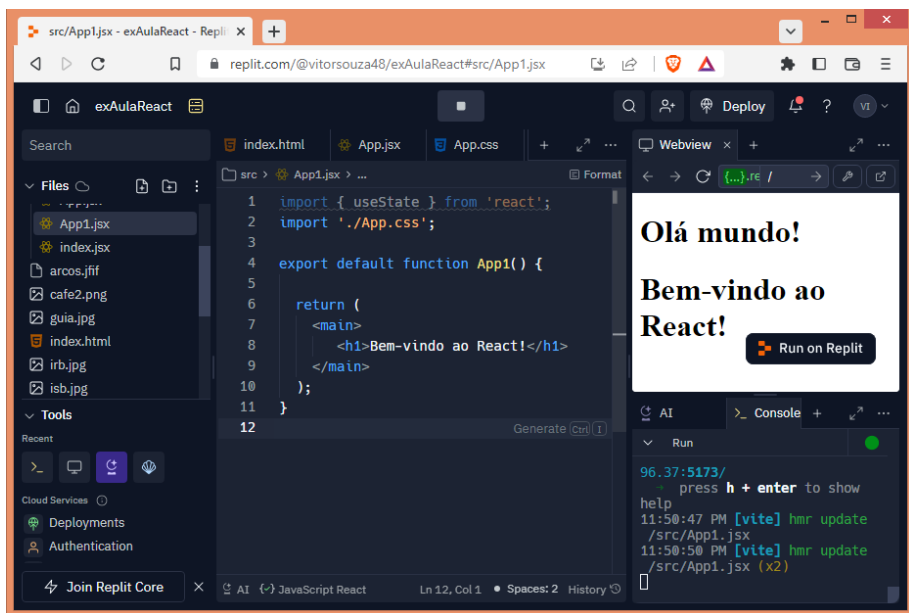
Agora copie o arquivo App.jsx e cole na mesma pasta, chamando-o de App1.jsx. Edite o arquivo App.jsx e deixe-o com o texto “Olá mundo!”, como sugere a próxima figura.



Edite o arquivo App1.jsx e deixe-o com o texto “Bem-vindo ao React!”.



Feito isso, rode o programa e observe o resultado.



8. Alterando a cor da div

Baseado no último exemplo, o objetivo deste exercício é deixar a primeira div na cor vermelha e a segunda na cor azul e os textos em negrito. Para este exemplo, precisaremos modificar o arquivo App.css para que atenda a esta demanda, usando o className para citar a classe com as definições deita neste arquivo. Observe como fica o arquivo App.css, App.jsx e App1.jsx para deixar o exercício conforme solicitado.

App.css

```
.red
{
  background-color:red;
}
.azul
{
  background-color:blue;
}
h1
{
  font-weight: bold;
}
```

App.jsx

```
import './App.css'

export default function App() {
  return (
    <h1 className="red">Olá Mundo!</h1>
  );
}
```

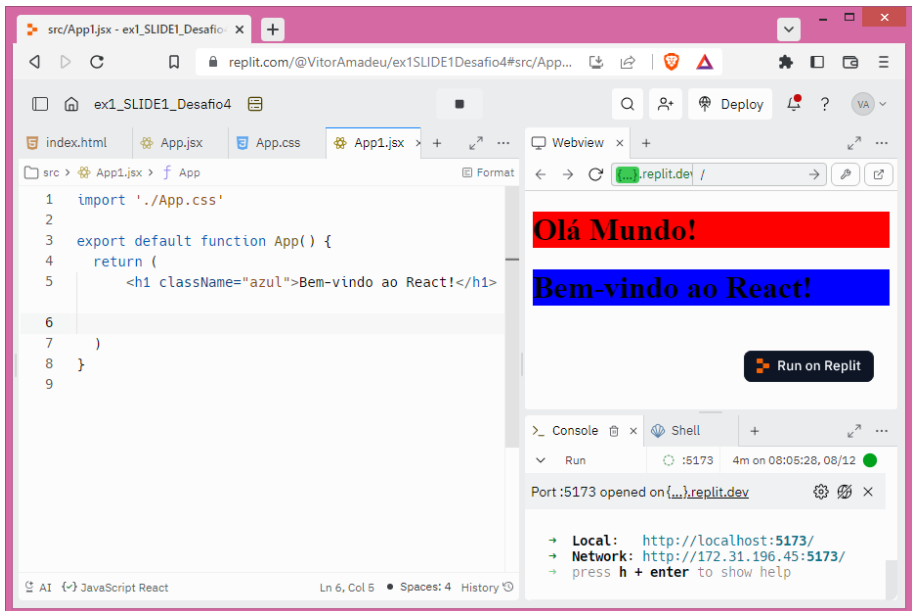
App1.jsx

```
import './App.css'

export default function App() {
  return (
```

```
<h1 className="azul">Bem-vindo ao React!</h1>
```

```
)  
}
```



Os códigos apresentados são dois trechos de uma aplicação React que utiliza CSS para estilizar o conteúdo. No primeiro trecho, uma classe CSS chamada `.red` é definida para aplicar um fundo vermelho (`background-color:red;`) a qualquer elemento HTML que utilize essa classe. Outra classe `.azul` é criada para aplicar um fundo azul (`background-color:blue;`). Além disso, uma regra CSS é

estabelecida para que qualquer elemento `<h1>` tenha o texto em negrito, utilizando `font-weight: bold;`.

No código JavaScript que segue, há uma função chamada `App`, que é um componente React. Neste componente, o texto "Olá Mundo!" é envolvido por um elemento `<h1>` com a classe `red`. Quando este componente for renderizado, o texto "Olá Mundo!" aparecerá em negrito e com o fundo vermelho, devido ao uso da classe `red` e da regra CSS para o `<h1>`.

No segundo trecho, o código é similar, mas o componente `App` agora renderiza um texto diferente: "Bem-vindo ao React!". Desta vez, o elemento `<h1>` utiliza a classe `azul`, o que faz com que o fundo do texto seja azul. Assim, ao ser renderizado, o texto "Bem-vindo ao React!" aparecerá em negrito e com o fundo azul, devido à aplicação da classe `azul` e da regra CSS para o `<h1>`.

Ambos os trechos fazem uso do arquivo `App.css` para a estilização dos componentes.

9. Alterando a cor de fundo

Baseado no último exemplo, o objetivo é deixar a tela na cor verde, mantendo o que foi feito no exemplo passado. Acompanhe os arquivos atualizados.

App.css

```
.red
{
  background-color:red;
}
.azul
{
  background-color:blue;
}
h1
{
  font-weight: bold;
}
body
{
  background-color:green;
}
```

App.jsx

```
import './App.css'

export default function App() {
  return (
    <h1 className="red">Olá Mundo!</h1>
```

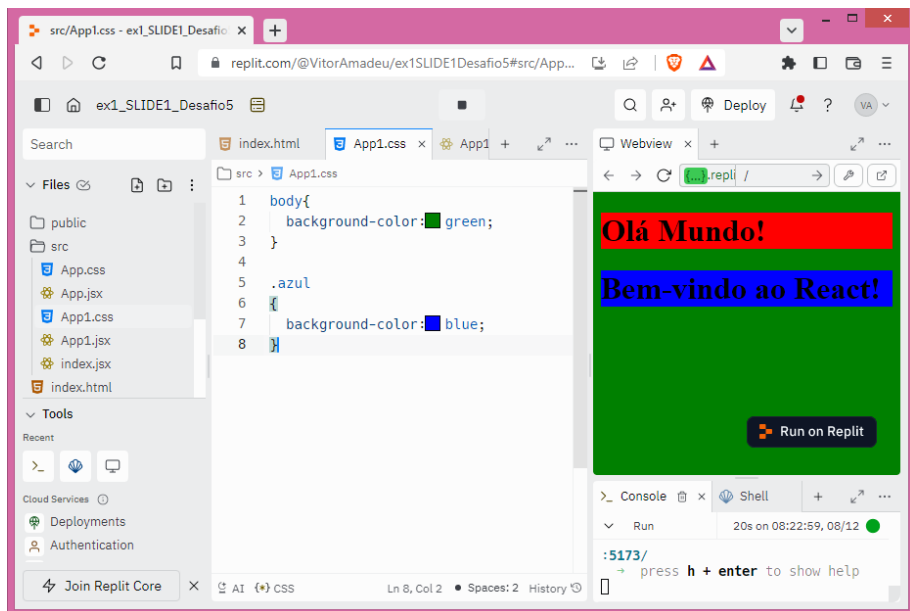
```
);
}
```

App1.jsx

```
import './App.css'

export default function App() {
  return (
    <h1 className="azul">Bem-vindo ao React!</h1>

  )
}
```



Os códigos apresentados consistem em um arquivo de estilo CSS (App.css) e dois componentes React (App.jsx e App1.jsx), que são usados para criar diferentes estilos visuais na aplicação. No App.css, foram definidas classes e regras de estilo. A classe .red aplica um fundo vermelho a qualquer elemento que a utilize, enquanto a classe .azul aplica um fundo azul. Além disso, a regra aplicada ao h1 deixa o texto dentro desse elemento em negrito, e a regra para o body define a cor de fundo da página como verde.

No componente App.jsx, o arquivo de estilos é importado, e um elemento <h1> é retornado com o texto "Olá Mundo!" e a classe red, resultando em um texto com fundo vermelho e em negrito, enquanto o fundo da página é verde. Já no componente App1.jsx, o processo é semelhante, mas o texto "Bem-vindo ao React!" é renderizado com a classe azul, o que faz com que o fundo do texto seja azul e o texto em negrito, mantendo o fundo da página verde. Assim, os dois componentes exibem diferentes textos com diferentes cores de fundo, conforme definido no CSS.

10. Adicionando uma div

Baseado no último exemplo, adicione mais uma div a tela com e escreva o texto “Programação responsiva”. Deixe o fundo desta div

na cor branca e cor da letra verde. Acompanhe os arquivos atualizados e o resultado.

O primeiro passo para execução deste exercício é adicionar uma div no arquivo index.html, neste caso a div root2.

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>React + TypeScript + Replit</title>
  </head>
  <body>
    <div id="root"></div>
    <div id="root1"></div>
    <div id="root2"></div>
    <script type="module" src="/src/index.jsx"></script>

    <script src="https://replit.com/public/js/replit-badge-v2.js" theme="dark"
      position="bottom-right"></script>
  </body>
</html>
```

Será necessário definir mais um componente App2.jsx para atualização na div que acabou de ser criada. Acompanhe os arquivos atualizados e adicionados assim como o resultado.

App.css

```
.red
{
  background-color:red;
}
.azul
{
  background-color:blue;
}
h1
{
  font-weight: bold;
}
body
{
  background-color:green;
}
.moda
{
  background-color:white;
  color:green;
}
```

App.jsx

```
import './App.css'

export default function App() {
  return (
    <h1 className="red">Olá Mundo!</h1>
  );
}
```

App1.jsx

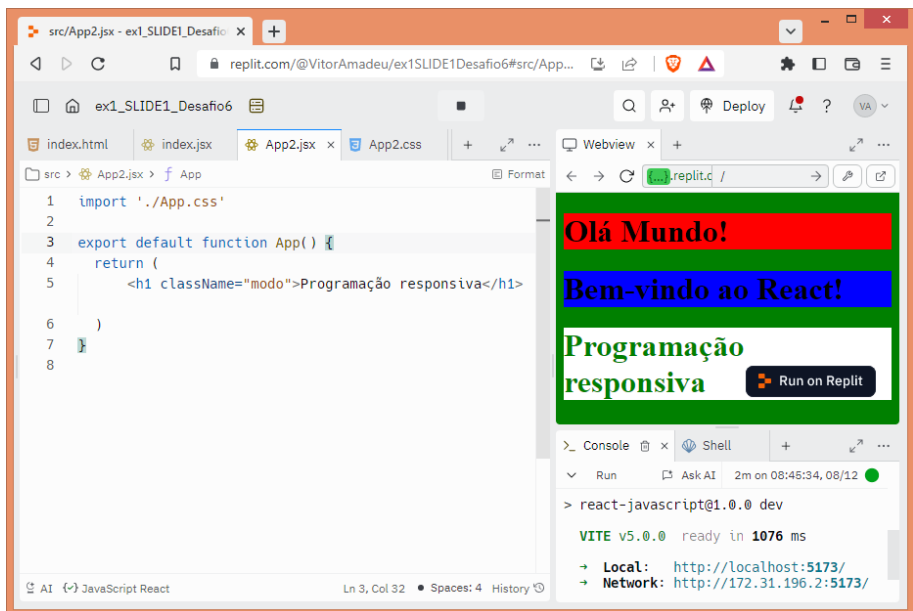
```
import './App.css'

export default function App() {
  return (
    <h1 className="azul">Bem-vindo ao React!</h1>
  );
}
```

App2.jsx

```
import './App.css'

export default function App() {
  return (
    <h1 className="modo">Programação responsiva</h1>
  )
}
```



11. Exemplo usando o flexbox

O Flexbox, ou Flexible Box Layout, é um modelo de layout no CSS que facilita a criação de layouts responsivos e bem alinhados, especialmente na distribuição de espaço entre itens em um contêiner e no controle de seu alinhamento. Ele é ideal para organizar elementos de forma eficiente e previsível, mesmo quando o tamanho

dos itens é desconhecido ou dinâmico. No Flexbox, o contêiner principal, chamado de Flex Container, é o elemento pai ao qual é aplicada a propriedade `display: flex;`. Todos os elementos filhos diretos desse contêiner se tornam Flex Items, ou seja, os itens flexíveis dentro do contêiner.

O Flexbox funciona com dois eixos principais: o Main Axis (eixo principal) e o Cross Axis (eixo transversal). O eixo principal, por padrão, é horizontal, e é ao longo dele que os itens são distribuídos. O eixo transversal é perpendicular ao eixo principal e controla o alinhamento dos itens na direção oposta. O Flexbox oferece várias propriedades para controlar como os itens são dispostos, alinhados e dimensionados, tanto ao longo do eixo principal quanto do eixo transversal, proporcionando grande flexibilidade na construção de layouts.

Baseado no flexbox, a proposta deste exemplo é elaborar uma caixa (box) de 64px e renderizá-la dentro de outra div (cor amarela), com três caixas cada uma com cor diferente (verde, azul e vermelho). Para isso, devemos deixar todas caixas centralizadas vertical e horizontalmente, sendo que a div maior tem altura de 256 px com espaçamento entre cada box de 10px. Crie um projeto novo no

Replit e deixe os arquivos App.jsx e App.css como apresentado a seguir.

App.css

```
.principal {
  background-color: yellow;
  height: 256px;
  display: flex;
  justify-content: center; /* Centraliza horizontalmente */
  align-items: center;     /* Centraliza verticalmente */
  flex-wrap: wrap;         /* Os boxes podem quebrar a linha*/
}

.box {
  height: 64px;
  width: 64px;
  margin: 10px;
}

.verde{
  background-color:green;
}

.azul{
  background-color:blue;
}

.vermelho{
  background-color:red;
}
```

App.jsx

```
import './App.css'

export default function App() {
  return (
    <main>
      <div class="principal">
        <div class="box verde"></div>
```

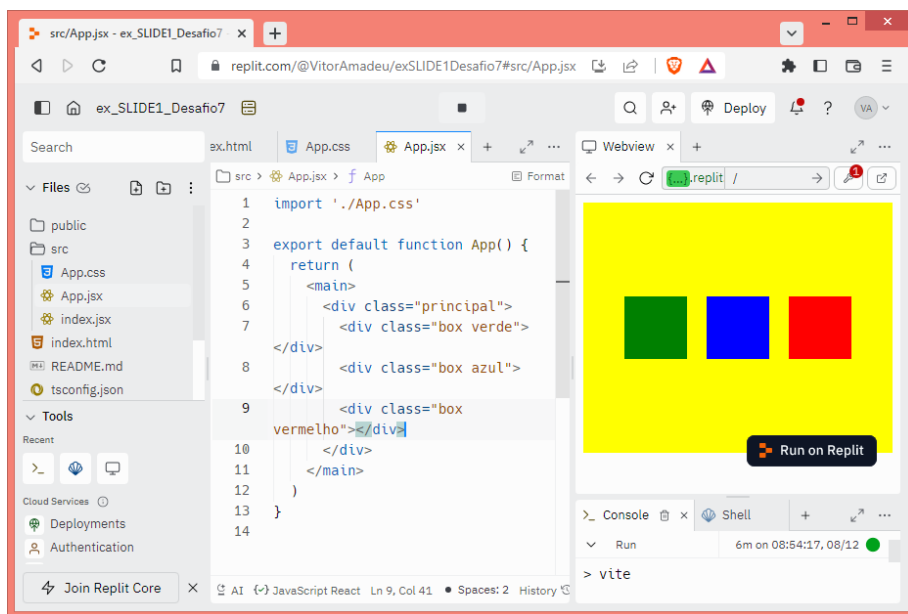
```
        <div class="box azul"></div>
        <div class="box vermelho"></div>
    </div>
</main>
)
}
```

Os códigos apresentados incluem dois arquivos: App.jsx, um componente React, e App.css, um arquivo de estilos CSS, que juntos criam e estilizam uma interface com três caixas coloridas dispostas em um contêiner flexível. No arquivo App.jsx, o componente App retorna uma estrutura HTML que consiste em um elemento <main>, dentro do qual há um <div> com a classe principal. Este <div> contém três outros <div>, cada um com a classe box e uma classe adicional (verde, azul, vermelho) que define a cor da caixa.

O arquivo App.css define os estilos para esses elementos. A classe principal aplica um fundo amarelo ao contêiner, define sua altura como 256 pixels e o transforma em um contêiner flexível com display: flex. As propriedades justify-content: center; e align-items: center; centralizam os itens horizontal e verticalmente dentro do contêiner, enquanto flex-wrap: wrap; permite que os itens quebrem a linha se necessário. A classe box define o tamanho das caixas (64x64 pixels) e aplica uma margem de 10 pixels ao redor de cada uma. As classes verde, azul e vermelho definem as cores de fundo

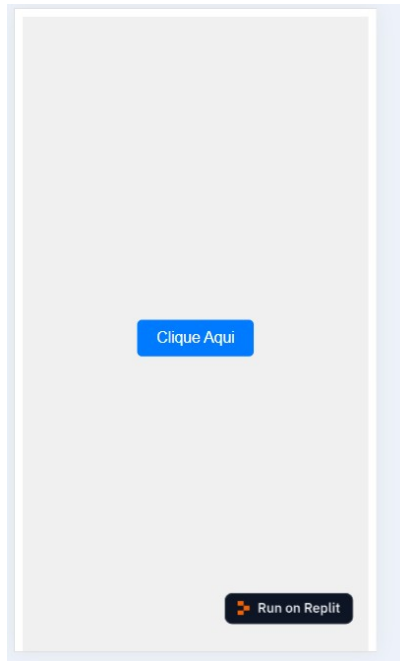
das caixas como verde, azul e vermelho, respectivamente. Quando o componente é renderizado, ele exibe três caixas centralizadas e coloridas dentro de um contêiner com fundo amarelo.

Acompanhe o resultado na tela abaixo.

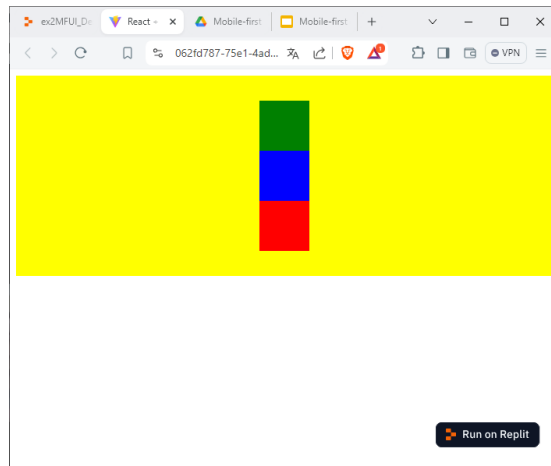
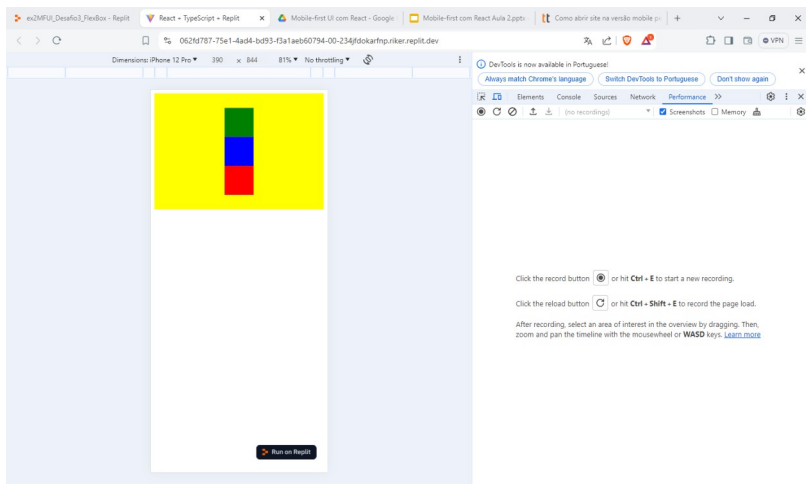


12. Adicionando um botão

O objetivo do próximo exemplo é definir um botão centralizado, como o apresentado na próxima figura.



Para observar a tela em diversos formatos de aparelhos, pressione F12 na tela de apresentação do browser. Veja que ao fazer isso, temos disponível diversas opções de dispositivos no qual pode ser verificado se o design de fato está responsivo.



Acompanhe como ficam os arquivos App.css e App.jsx para este exercício.

App.css

```
.flex.flex-container {
  display: flex;
  justify-content: center; /* Centraliza horizontalmente */
  align-items: center;    /* Centraliza verticalmente */
  height: 100vh;          /* Altura da viewport */
  background-color: #f0f0f0;
}

.flex-button {
  padding: 10px 20px;
  font-size: 16px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
}

.flex-button:hover {
  background-color: #0056b3;
}
```

App.jsx

```
import './App.css'

export default function App() {
  return (
    <div className="flex-container">
      <button className="flex-button">Clique Aqui</button>
    </div>
  )
}
```

```
)  
}
```

Os códigos apresentados consistem em dois arquivos: App.jsx, que é um componente React, e App.css, que define os estilos CSS para os elementos da aplicação. No arquivo App.jsx, o componente App retorna uma estrutura HTML, composta por um <div> com a classe flex-container. Dentro deste <div>, há um botão (<button>) com a classe flex-button e o texto "Clique Aqui".

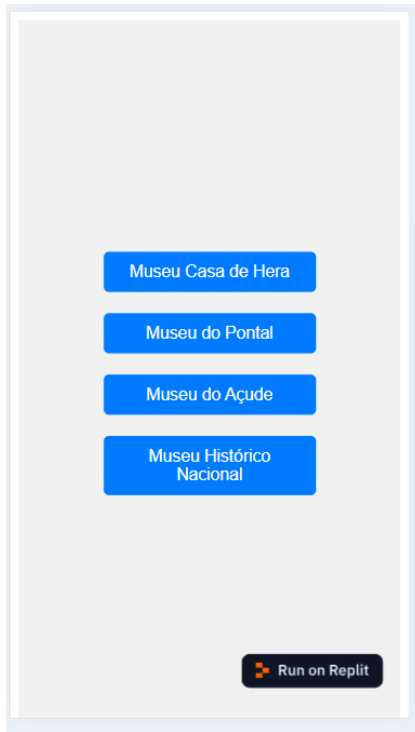
O arquivo App.css define os estilos para esses elementos. A classe flex-container transforma o <div> em um contêiner flexível (display: flex;), centralizando seu conteúdo tanto horizontal quanto verticalmente, utilizando as propriedades justify-content: center; e align-items: center;. A altura do contêiner é definida como 100% da altura da viewport (height: 100vh;), e o fundo do contêiner é um cinza claro (background-color: #f0f0f0;).

A classe flex-button estiliza o botão dentro do contêiner. O botão recebe um preenchimento interno (padding: 10px 20px;), um tamanho de fonte de 16 pixels, um fundo azul (background-color: #007bff;), texto branco, bordas arredondadas com um raio de 5 pixels, e um cursor do tipo "pointer" ao passar o mouse sobre ele. A transição da cor de fundo é suavizada ao longo de 0,3 segundos

(transition: background-color 0.3s;). Além disso, quando o botão é focado ou o mouse passa sobre ele (hover), a cor de fundo muda para um azul mais escuro (#0056b3), criando um efeito visual de interação. Quando o componente App é renderizado, ele exibe um botão centralizado na tela, estilizado de acordo com as regras definidas, e com um efeito de mudança de cor ao passar o mouse sobre ele.

13. Botão com link

O objetivo do próximo exemplo é adicionar quatro botões ao layout responsivo com o redirecionamento para quatro museus disponíveis no estado do RJ, como o apresentado na próxima figura.



Acompanhe como ficam os arquivos App.css e App.jsx para este exercício.

App.css

```
.flex-container {  
  display: flex;  
  justify-content: center; /* Centraliza horizontalmente */  
  align-items: center;    /* Centraliza verticalmente */  
  height: 100vh;         /* Altura da viewport */  
  background-color: #f0f0f0;  
  flex-direction: column;
```

```

}

.flex-button {
  padding: 10px 20px;
  font-size: 16px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
  margin: 10px;
  width: 200px;
}

.flex-button:hover {
  background-color: #0056b3;
}

.flex-button-link {
  text-decoration: none; /* Remove sublinhado do link */
}

```

App.jsx

```

import './App.css'

export default function App() {
  return (
    <div className="flex-container">
      <a href="https://museucasadahera.museus.gov.br/" className="flex-button-link">
        <button className="flex-button">Museu Casa de Hera</button>
      </a>

      <a href="https://museudoportal.org.br/" className="flex-button-link">
        <button className="flex-button">Museu do Portal</button>
      </a>
    </div>
  )
}

```

```

    </a>

    <a href="https://pt.wikipedia.org/wiki/Museu_do_A%C3%A7ude" className="flex-
button-link">

    <button className="flex-button">Museu do Açude</button>
  </a>

  <a href="https://mhn.museus.gov.br/" className="flex-button-link">
    <button className="flex-button">Museu Histórico Nacional</button>
  </a>
</div>
)
}

```

Os códigos apresentados consistem em um componente React (App.jsx) e um arquivo de estilos CSS (App.css). No componente App, a estrutura HTML inclui um contêiner flexível (<div> com a classe flex-container) que organiza verticalmente quatro botões, cada um envolvido por um link (<a>). Esses links direcionam para diferentes sites de museus.

O arquivo CSS define o estilo para o contêiner e os botões. O contêiner é configurado como um contêiner flexível com display: flex;, centralizando seu conteúdo horizontal e verticalmente e ajustando sua altura para ocupar toda a altura da viewport (height: 100vh;). O fundo do contêiner é cinza claro (background-color: #f0f0f0;), e a direção dos itens é definida como coluna (flex-direction: column;), fazendo com que os botões se empilhem verticalmente.