PRIMEIRO MÓDULO

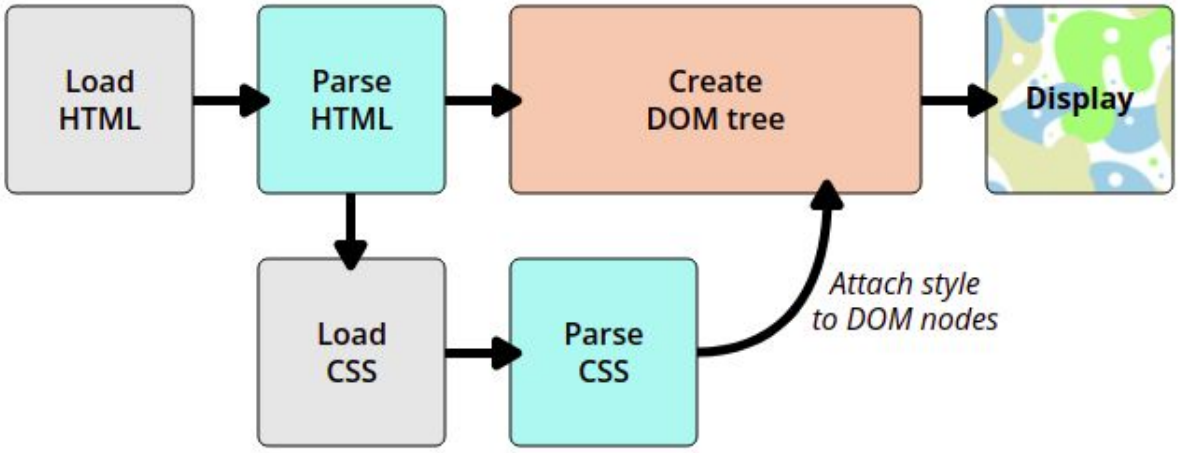
Visão geral sobre interação cliente e servidor

Visão geral sobre HTML e CSS

Hora do Código

Bom exemplo para estruturação do desafio

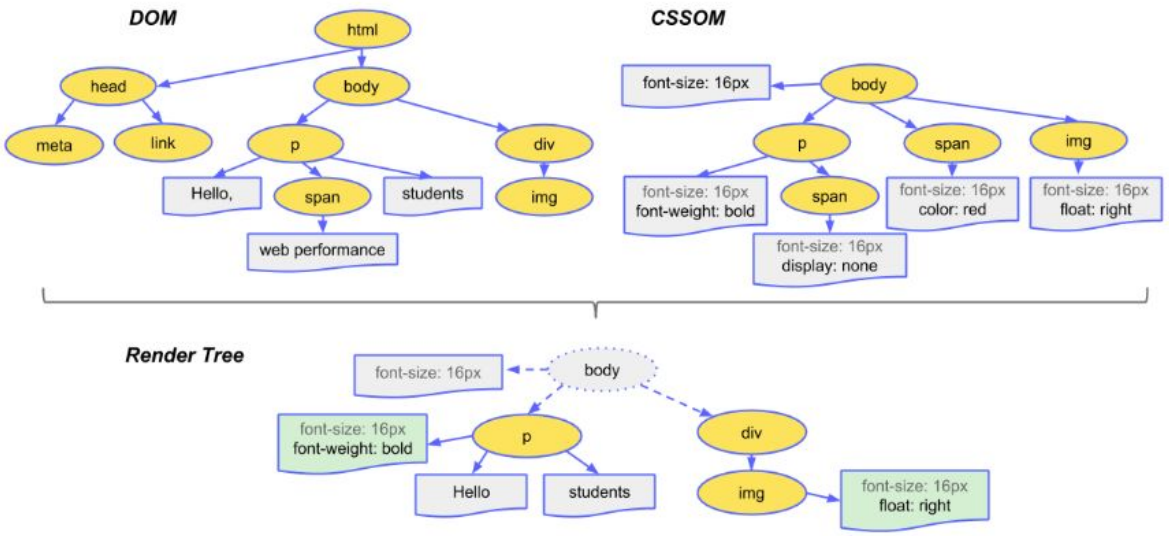
How CSS Works



Quando uma regra CSS é inválida, nada acontece. Aquele estilo simplesmente é ignorado
Por isso, faz sentido aplicar fallbacks em alguns casos, para features novas que nem todos os navegadores suportam. Ex:

```
.box {
  width: 500px;
  width: calc(100% - 50px);
}
```

Construção, layout e gravação da árvore de renderização

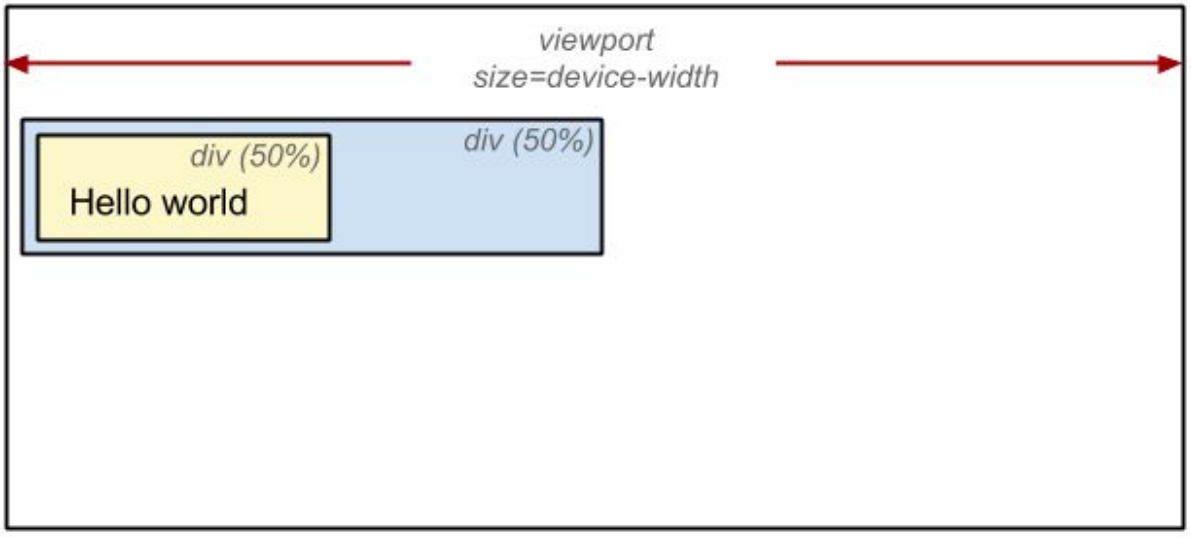


Para construir a árvore de renderização, em termos gerais, o navegador executa as seguintes atividades:

- A partir da raiz da árvore DOM, percorre cada nó visível.
  - Alguns nós não são visíveis (por exemplo, tags script, tags meta e assim por diante) e são omitidos, pois não são refletidos no resultado da renderização.
  - Alguns nós foram ocultados via CSS e também são omitidos da árvore de renderização, como por exemplo, o nó "span"---do exemplo acima---não está presente na árvore de renderização porque temos uma regra explícita que define a propriedade "display: none" nela.
- Para cada nó visível, encontre as regras do CSSOM correspondentes adequadas e aplique-as.
- Para cada nó visível, encontre as regras do CSSOM correspondentes adequadas e aplique-as.

Até agora, calculamos que nós devem ser visíveis e seus estilos processados. Mas ainda não calculamos a posição e o tamanho exatos na janela de visualização do dispositivo---essa é a fase do layout, também conhecida como **"reflow".**

Para determinar o tamanho e a posição exatos de cada objeto, o navegador começa na raiz da árvore de renderização e passa por toda ela



O resultado do processo de layout é um "modelo de caixa" que captura a posição e o tamanho exatos de cada elemento dentro da janela de visualização. Todas as medições relativas são convertidas em pixels absolutos na tela.

Agora podemos finalmente passar essas informações para a última fase, que converte cada nó da árvore de renderização em pixels reais na tela. Essa etapa é frequentemente chamada de **"gravação" ou "rasterização"**.

Vamos recapitular as etapas do navegador:

1. Processar a marcação HTML e criar a árvore do DOM.
2. Processar a marcação CSS e criar a árvore do CSSOM.
3. Combinar o DOM e o CSSOM em uma árvore de renderização.
4. Executar o layout na árvore de renderização para calcular a geometria de cada nó.
5. Pintar os nós individuais na tela.

A otimização do caminho crítico de renderização é o processo de minimizar o total de tempo gasto nas etapas 1 a 5 da sequência acima. Isso permite renderizar conteúdo na tela o mais cedo possível, além de reduzir o tempo entre as atualizações da tela após a renderização inicial, ou seja, atingir uma taxa de atualização mais alta para conteúdo interativo.

Browser Engine: Motores e Debugging

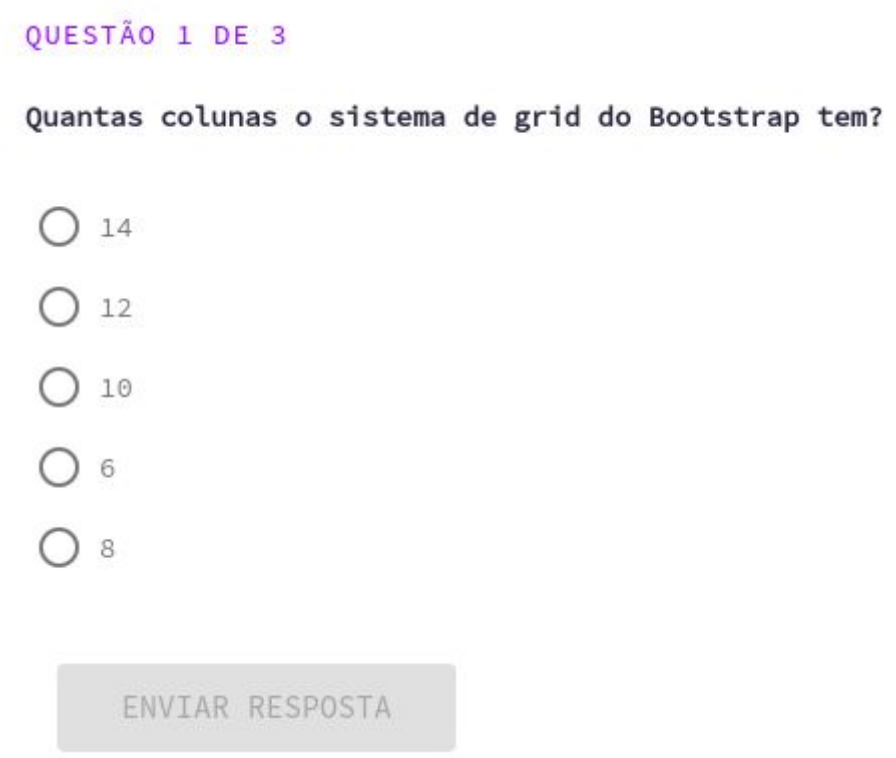
O servidor é basicamente dividido em algumas partes:

- Web Server (Servidor Web) que é o programa responsável por receber e responder
as requisições HTTP vindas do cliente
- Web Application (Aplicação Web) que é o programa que tem a lógica de negócio da aplicação
- Database (Banco de Dados) que é onde as informações são armazenadas

Tags descritivas:

main: utilizamos apenas uma vez na página, ela deve englobar o conteúdo principal da página
header: engloba conteúdos de cabeçalho de algum texto
footer: também geralmente utilizamos uma vez na página, sendo a tag que engloba o conteúdo do final da página
nav: tag que engloba o principal menu de navegação da página
video: tag que engloba conteúdo de vídeo
article: tag que engloba um conteúdo auto contido
section: tag que engloba vários conteúdos que se relacionam

QUIZZ



R: 12

QUESTÃO 2 DE 3

Quais são componentes do Bulma?

- ◯ Breadcrumb, Message e Tabs
- ◯ Tabs, Panel e Popover
- ◯ Popover Pagination, Navbar
- ◯ Carousel Dropdown e Breadcrumb
- ◯ Tabs, Carousel e Nav

ENVIAR RESPOSTA

R: Breadcrumb, Message e Tabs

QUESTÃO 3 DE 3

No componente Card do Bootstrap, qual classe é usada para adicionar padding em uma seção?

- ◯ .card-body
- ◯ .card-content
- ◯ .card-box
- ◯ .card-padding
- ◯ .card-item

ENVIAR RESPOSTA

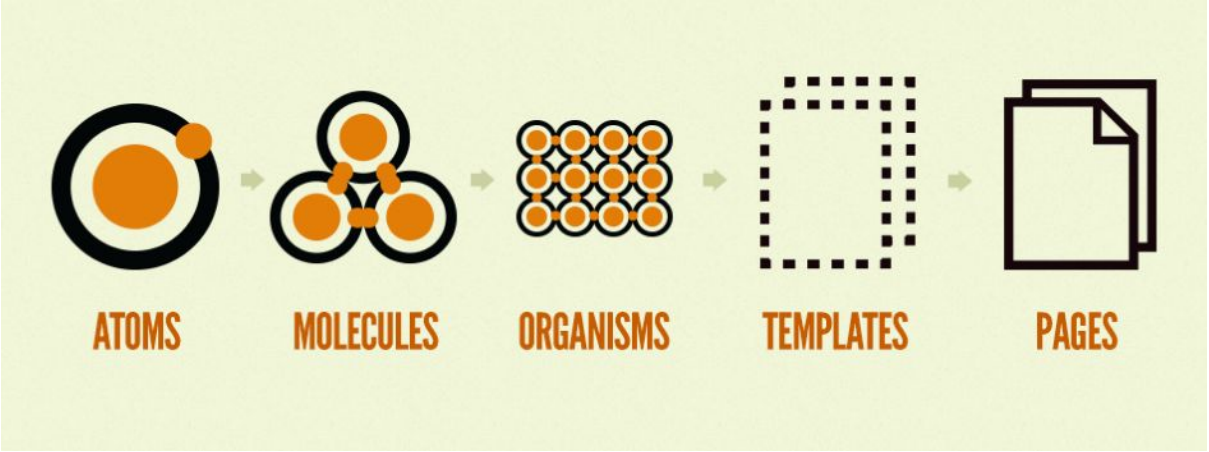
R: .card-body

SEGUNDO MÓDULO

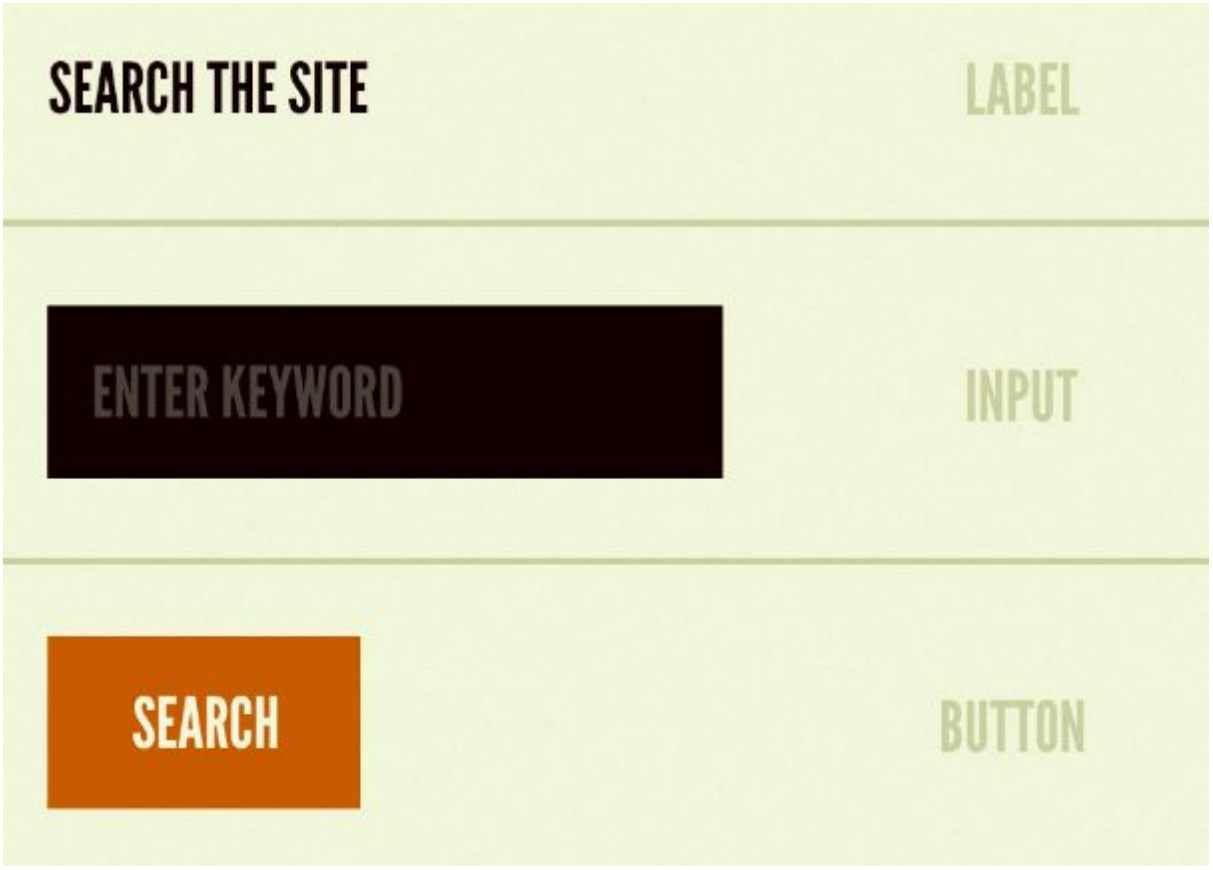
BEM (Block, Element, Modifier)

sintaxe: **.block__element--modifier**
exemplo: **.header__logo--dark, .card__item--active**

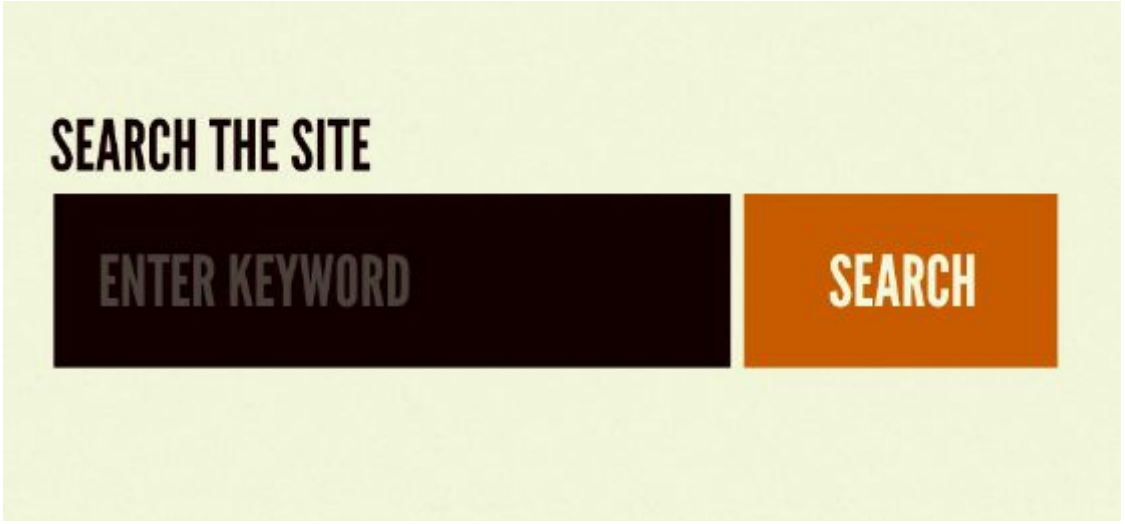
ATOMIC DESIGN



Átomos:



Molécula

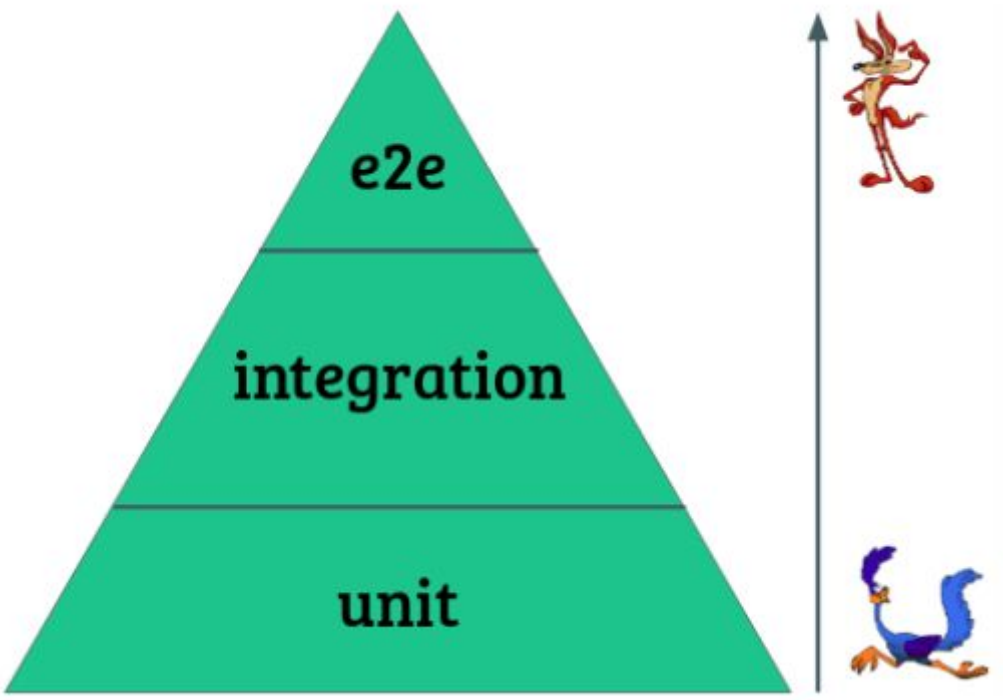


Organismo

QUARTO MÓDULO

Learning JS by doing it: https://jskatas.org/

A pirâmide de testes:



Funções puras -> sem side effects. Mesma entrada gerada mesma saída SEMPRE. Com isso, são mais fáceis de testar

Currying: A técnica de transformar uma função com múltiplos parâmetros em uma sequência de funções que aceitam apenas um parâmetro é chamada de Currying. Grande vantagem: transformar 0 código em pequenos pedaços mais expressivos e com maior reuso.

Exemplo:

```
var add = function(x, y) {
   return x + y;
};

add(1, 2) // 3
```

noCurrying.js hosted with ❤ by GitHub view raw

Nisso:

```
var add = function(x) {
    return function(y) {
        return x + y;
    };
};

add(1)(2); // 3
```

currying.js hosted with ❤ by GitHub view raw

Compose:

```
const compose = (f, g) => x => f(g(x));

const toUpperCase = x => x.toUpperCase();
const exclaim = x => x + '!';

const angry = compose(toUpperCase, exclaim);

angry('ahhh'); // AHHH!
```

composeES6.js hosted with ❤ by GitHub view raw

Closures:

Voltando às closures...

Closure é a forma de fazer com que as variáveis dentro de uma função sejam **privadas** e **persistentes**.

Top highlight

```
function pai(){
   var x = 1;
   function filho(){
     console.log(x);
     x++;
   }
   return filho;
}

var contador = pai();
contador();    // 1
contador();    // 2
contador();    // 3
```

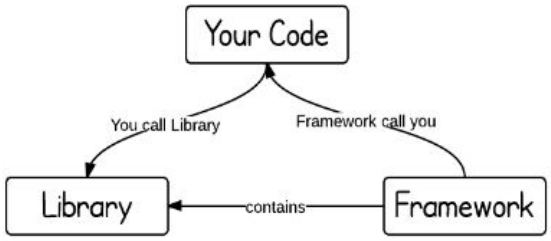
O simples fato de atribuirmos a função filho para uma variável, faz com que a função filho ainda seja necessária, da mesma forma que as variáveis que ela utiliza (o x no escopo da função pai) e isso faz com que a função pai também seja necessária. OU SEJA, esses dados não serão coletados pelo garbage collector da engine, fazendo com que a informação ali contida seja persistente e possa ser utilizada fora de seu escopo estático (!).

A função filho possui uma referência ao escopo da função pai, e a essa referência nós damos o nome de closure.

QUINTO MÓDULO

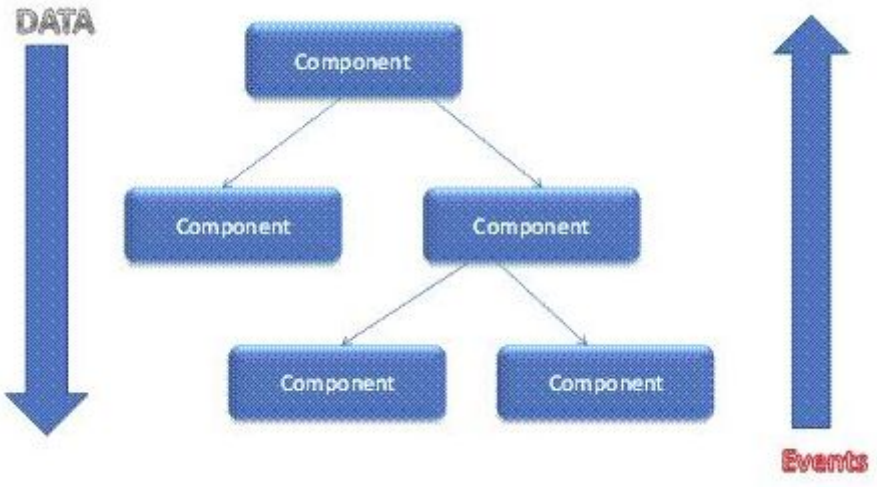
Diferença entre biblioteca e framework:

The key difference between a library and a framework is "Inversion of Control". When you call a method from a library, you are in control. But with a framework, the control is inverted: the framework calls you.



Proprieades do React:

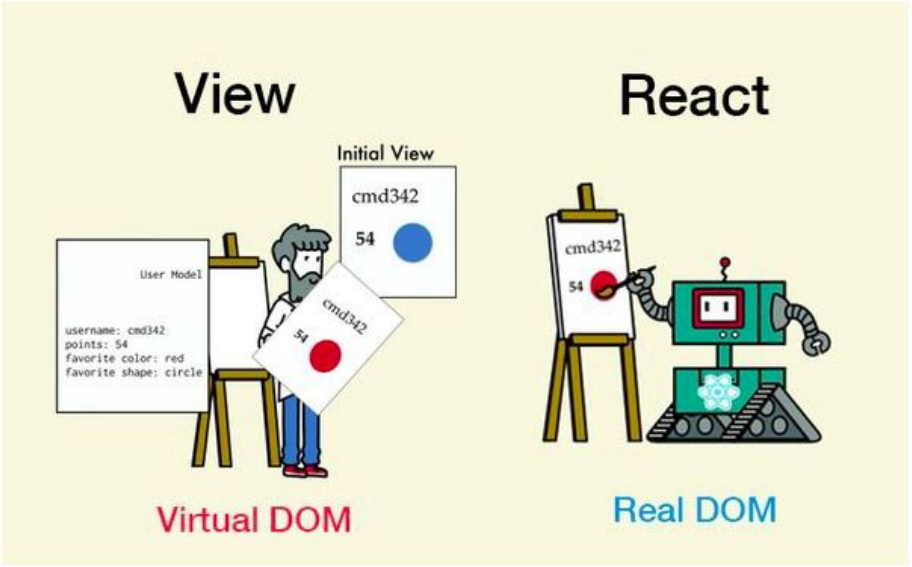
Fluxo unidirecional:



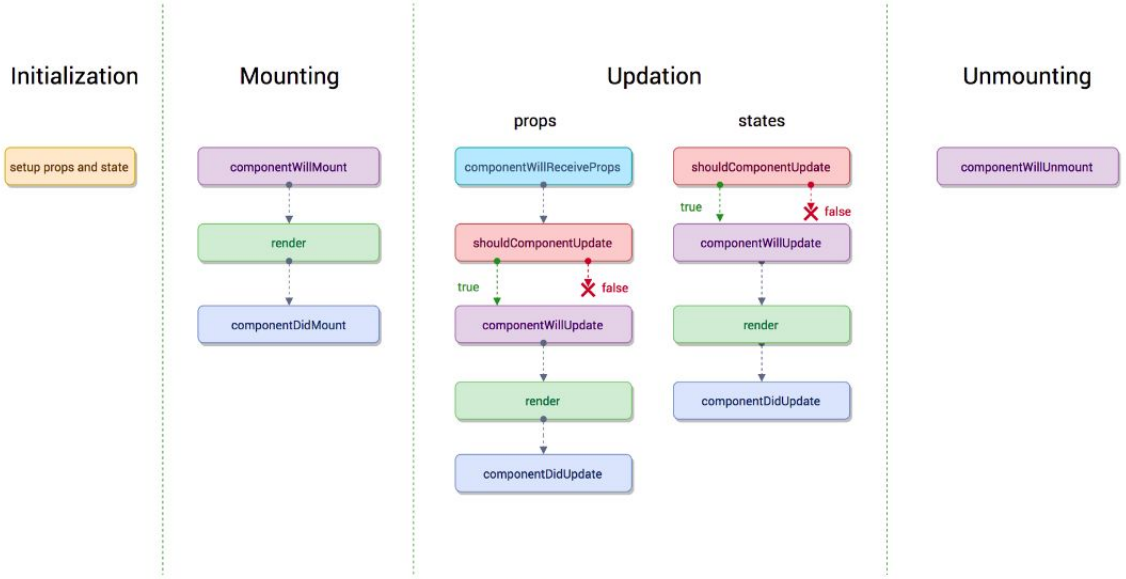
“properties flow down; actions flow up”.

Virtual Document Object Model

React creates an in-memory data structure cache which computes the changes made and then updates the browser. This allows a special feature that enables the programmer to code as if the whole page is rendered on each change whereas react library only renders components that actually change.

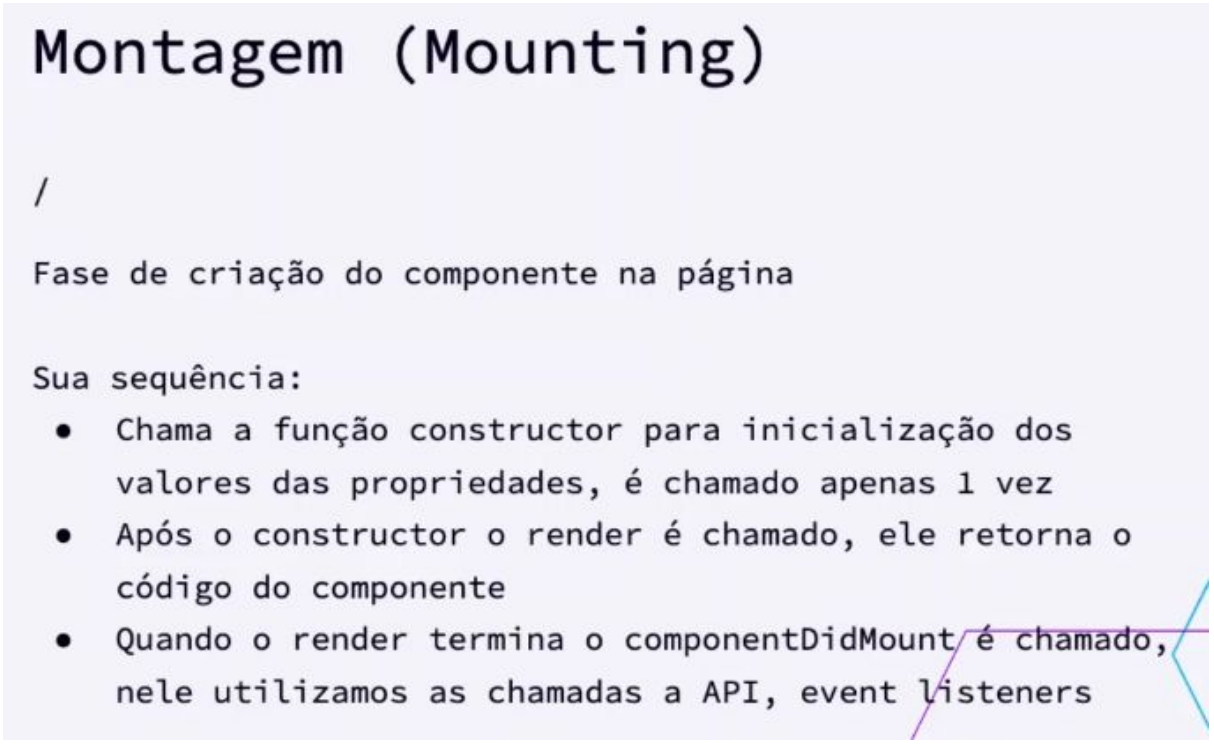


Clico de vida de um componente React:



Fases e métodos do ciclo de vida do ReactJs

Sobre Cicos de Vida (aula 1)



Para ter acesso aos ciclos de vida de um componente, ele deve ser escrito na forma de Class extends React.Component

props são externas, recebidas pelo componente
state é o estado local do componente, inicializado com this.state = {} e alterável via this.setState

React Hooks e React Funcional

“Hooks são funções que permitem a você “ligar-se” aos recursos de state e ciclo de vida do React a partir de componentes funcionais. Hooks não funcionam dentro de classes — eles permitem que você use React sem classes.”

Principais: useState e useEffect

Exemplo:
import React, { useState } from ‘react’;

const [count, setCount] = useState(0); cria uma variável count no state e uma função setCount para alterar este valor. O valor inicial é informado como parâmetro de chamada de useState;

Já o useEffect segue a mesma finalidade do componentDidMount, componentDidUpdate, e componentWillUnmount em classes React, mas unificado em uma mesma API.

Quando não há um segundo parâmetro informado, funciona como os 3 ciclos de vida citados anteriormente. Se há segundo parâmetro, funciona como “update” apenas para o valor informado.

O useEffect ainda pode retornar uma função, que será executada no componenteWillUnmount, geralmente com objetivo de cleanUp