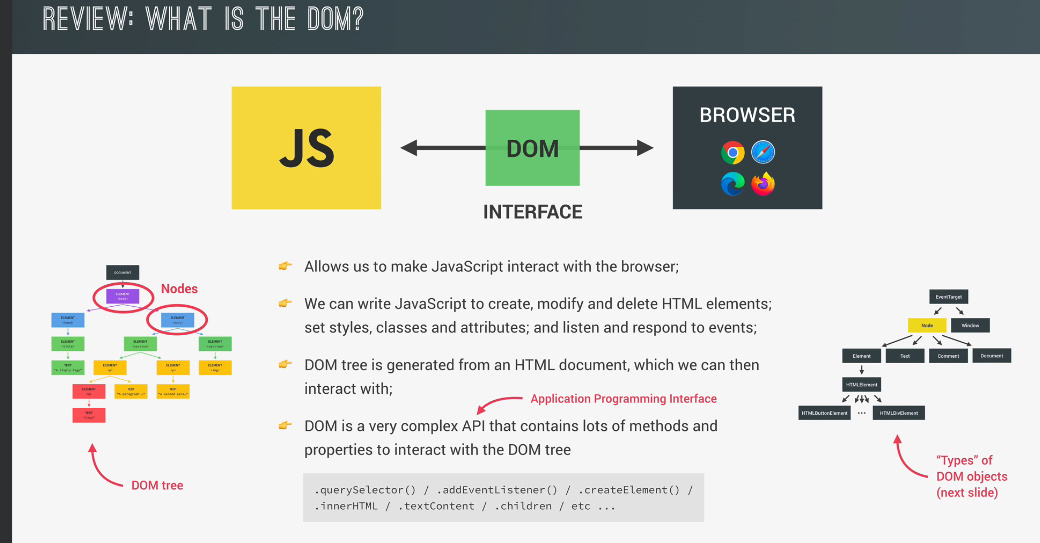
**DOM Tree More Info**

* 

DOM e un API ce permite defapt la JS sa interactioneze cu browserul.

Totodata, DOM permite interactiunea cu DOM Tree

* Fiecare node din tree e de tip Node, de aceea, querySelectAll de ex, returneaza un NodeList
* Asa cum un node este un obiect, are si el metode si proprietati, ca textContent, childNode etc.
* Un Node poate fi de mai multe feluri:

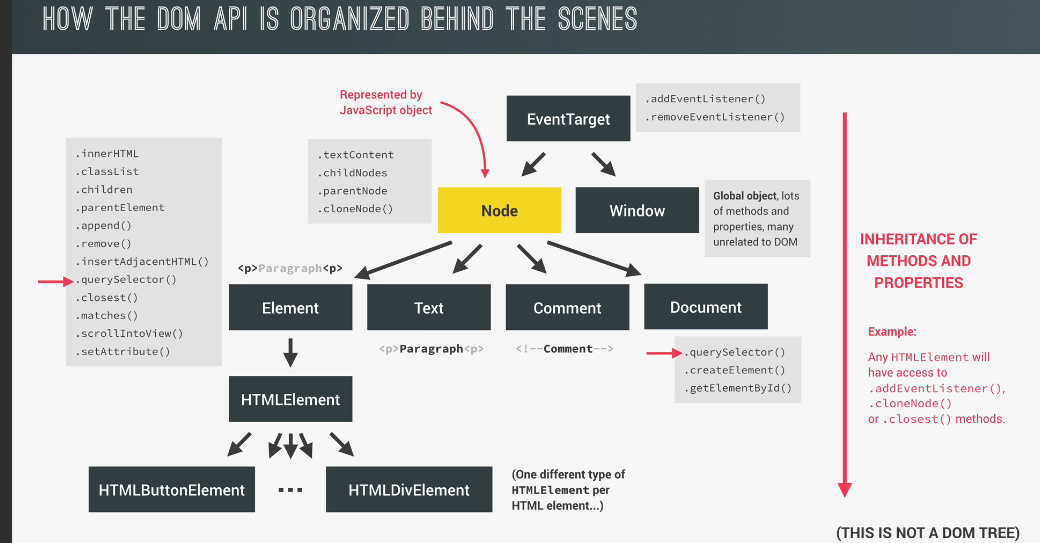
- Element – fiecare element de pe pagina, ca tag, este un node. Fiecare element este de tip Node si Element, dar are implementare specifica, de ex un buton e de tip HTMLButtonElement si tot asa si fiecare are atribute specifice

- Text – este textul elementelor

- Comment

- Document – intreaga pagina html. Folosim mereu ‘document.ceva’

Atentie! Asta nu e DOM TREE, e ierarhia de Node!!!!



Node mosteneste de la EventTarget, la fel si window, si asta ne permite sa folosim event listener methods la orice.

E evident ca fiecare tip de Node mosteneste metodele si proprietatile de la parent

**Selecting, creating and deleting elements**

* JS ofera document object
* document este root node din documentul HTML
* totusi, document nu propriu zis obiectul document de tip node din Tree, ci mai contine si alte chestii, deci pentru a accesa propriu zis document object al paginii, din tree, folosim

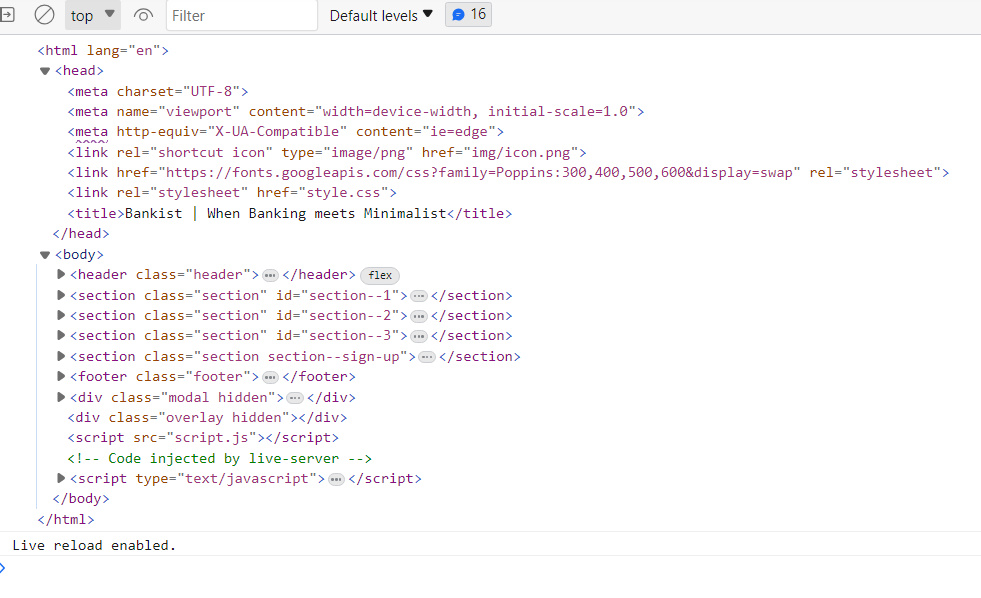
**- document.documentElement**

Asta e document:



contine mult mai multe decat document object din tree

Dar, asta deja e propriu zis document element, cu **document.documentElement**



Asta e pagina propriu zisa, deci anume asa accesam propriu zis document al paginii, intregul HTML

- **document.head** – returneaza <head> </head>

- **document.body** – returneaza <body> </body>

- **document.getElementById()**

- **document.getElementsByTagName()** – returneaza un HTMLCollection. Deosebirea fata de NodeList e ca daca stergem vren tag, de ex un <button>, el dispare deodata din HTMLCollection, deci o lista live, dar asta nu se intampla si in NodeList

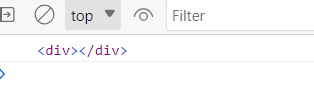
- **document.getElementsByClassName()** – tot returneaza o HTMLCollection

- **document.insertAdjacentHTML(‘where’, html)**

- **document.createElement(‘tag’)** – creaza un dom element si il returneaza, dar nu e pus nicaieri pe pagina. Urmeaza inca sa o facem. Doar s-a creat un dom element si gata

const *message* = *document*.createElement('div');  
*console*.log(*message*)

si asta el contine



Este exact acelasi tip de obiect ca cel returnat, de ex, cu querySelector(), deci putem aplica exact aceleasi metoda asupra lui, sau properties, ca textContent, insertAdjaentHTML() etc.

Odata creat, il inseram undeva, gen cu insertAdjacentHTML

const *message* = *document*.createElement('div');  
*message*.innerHTML = '<b>Hello!!!</b>';  
  
*document*.getElementById('test').append(*message*)

am putea si direct pune in append ‘<div> <b>Hello!!!</b< </div>’

**- append()** – pune elementul la urma, ca ultimul mereu. E o simplificare pentru insertAdjacentHTML(‘beforend’)

- **prepend()** – pune elementul ca primul. E simplificare pentru insertAdjacentHTML(‘afterbegin’)

- **before()** – pune elementul inaintea elementului curent. E simplificare pentru insertAdjacentHTML(‘beforebegin’)

- **after()** – pune elementul dupa elementul curent. E simplificare pentru insertAdjacentHTML(‘afterend’)

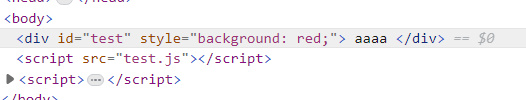
- **cloneNode(true)** – face o copie a elementului. Daca inseram acelasi obiect de mai multe ori, el se va pune doar odata, caci nu poate exista aceeasi referinta in mai multe locuri in tree, dar cloneNode() va returna un nou obiect

- **remove()** – sterge elementul curent din document si pagina deci

*document*.getElementById('test').remove();

**Styles, attributes**

* Orice element are un field style ce ne permite sa setam proprietati. Folosind style.stil le setam ca inline:



Problema e ca, daca vom face asa ceva:

*console*.log(*document*.getElementById('test').style.background);

daca background nu e inline style, el nu va fi afisat. Nu conteaza ca el e definit undeva intr-o foaie CSS, style field nu de paginile css se ocupa. style field ia si pune totul doar ca inline styles

* Pentru a primi styles si din css file, pentru un element, folosim **getComputedStyle(element)**

const *element* = *document*.getElementById('test');  
  
*console*.log(getComputedStyle(*element*))

computed style inseamna style ce e vizibil pe pagina, indiferent ca vine de la inline style sau css file

* Daca vrem sa extragem o anumite proprietate, gen color, folosim:

**getComputedStyle(element).color**

si ca sa o setam

getComputedStyle(element).color = ‘green’

* Putem accesa un atribut, de ex src, usor asa:

*document*.getElementById('test').src

dar mai e si metoda

**.getAttribute(‘atribut’)**

*document*.getElementById('test').getAttribute('src')

* Putem seta un atribut cu

**.setAttribute(‘atribut’, ‘value’)**

Dar putem si asa

*document*.getElementById('test').src = 'index.html'

**Variables in css**

* CSS tot ofera posibilitatea de a declara variabile
* Ele se declara in :root si mereu incep cu --

:root{  
 --test: 25px;  
}

* Pentru a le folosi, utilizam var(‘--variabile’)

:root{  
 --test: 25px;  
 }  
#test{  
 color: red;  
 font-size: var(--test);  
}

* Daca vrem sa modificam o variabila cu JS:

*document*.documentElement.style.setProperty('--test', '100px')

**data- attribute**

* data- attribute e folosit pentru a adauga anumite date in element, ce vor fi folosite apoi de JS
* El se creaza asa:

- incepe cu data-

- dupa – urmeaza litere fara uppercase!

<div id='test' data-test='test\_data' data-try='try'>  
aaaa  
</div>

In JS o accesam asa

*console*.log(*document*.getElementById('test').dataset.try)

in dataset se pastreaza toate atributele ce incep cu data-,

**getBoundingClientRect()**

* Pentru a afla coordonatele unui obiect,in raport cu bordurile, folosim metoda:

.**getBoundingClientRect()**

Si primim asa ceva:

DOMRect{

bottom: 528.8000011444092,

height: 28.80000114440918,

left: 158,

right: 1528,

top: 500,

width: 1370,

x: 158,

y: 500

Daca elementul e mai jos de cat e bordura, oricum va arata cati pixeli e distanta pana la el, ceea ce ar insemna ca evident,daca nu e dat scroll, distanta va fi mai mare ca bordura la browser si ar necesita sa se dea scroll. Atentie!!! Se ia in raport cu bordura la browser, nu cu punctul de start al paginii. De ex, fie un element jos de tot. Cand deschidem siteul, fara a da scroll, y = 1800. Daca dam putin scroll in jos, y = 1600, de ex. Se ia x si y in raport cu bordura la care se afla browser in pagina, deci daca dam scroll, y si x se modifica! Nu se ia extremitatea paginii, ci locul actul al bordurei!

De ex, astea sunt coordonatele la un obiect ce nu e in bordura, ci necesita scroll

bottom: 2423.8500366210938

height: 1670.25

left: 0

right: 1519.2000732421875

top: 753.6000366210938

width: 1519.2000732421875

x: 0

y: 753.6000366210938

se vede ca sunt mii la coordonate. Daca dam scroll in jos, y va scadea, caci se misca bordura la browser, si nu se ia extremitatea paginii, ci acolo unde e browser in pagina. De ex, daca dam 200px in jos, y = 753 – 200 = 553

* Metodele ce pot fi apelate asupra obiectului returnat:

- x

- y

- bottom

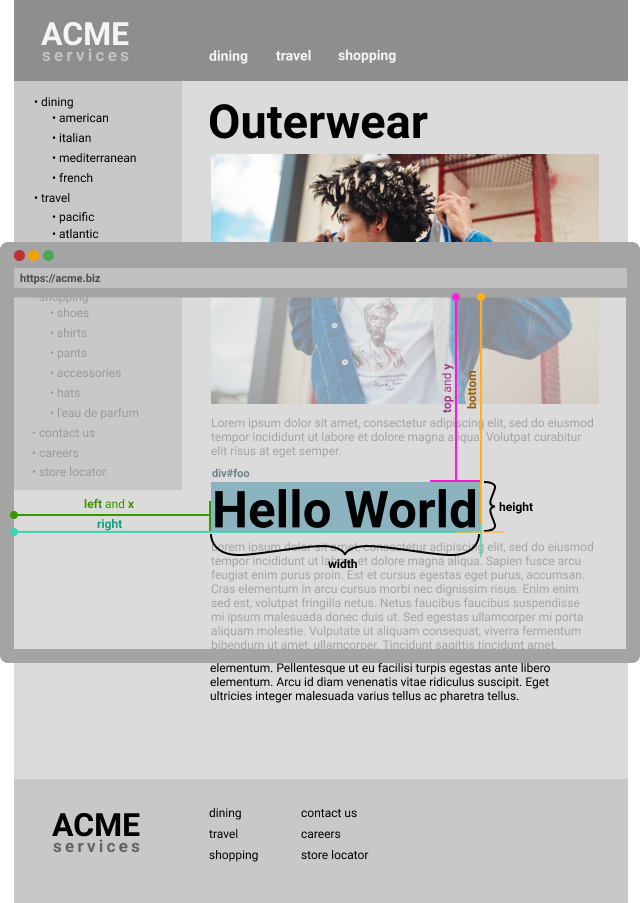
- top

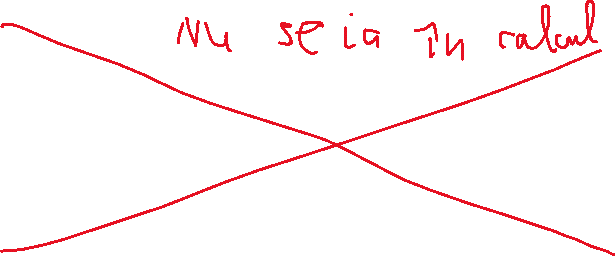
- left

- right

- height

- weight





- **left and x** – ele ne arata distanta de la bordura din stanga pana partea stanga a elementului, adica inceputul lui

- **right** – arat distanta de la bordura din stanga pana la partea dreapta a lementului, adica finalul elementului

- **top and y** – distanta de la bordura de sus a paginii si partea de sus a elementului

- **bottom** – de la bordura de sus pana la partea de jos a elementului

Atentie! Aceste valori nu sunt mereu aceleasi(in afara de with si height)! Daca userul da scroll in jos sau dreapta ori invers si da iar click, coordonatele se schimba! Datele de la metoda asta se iau in raport cu pagina din browser, adica bordurile din browser, nu cu pagina propriu zisa.

**document.documentElement.clientWidth/clientHeight**

* Aceste fields arata care este marimea paginii userului. Daca el face pagina browserului mai mica, evident ca si aceste marimi se fac mai mici

**window.pageYOffSet(scrollX)/pageXOffSet(scrollY)**

* Aceste proprietati arata cu cati px userul a dat scroll in lungime si latime la pagina.
* **pageX/YoffSet sunt depreciate, si se foloseste scrollX/Y!!!**

**window.scrollTo(x, y)**

* Metoda asta duce userul la coordonatele specificate
* x si y nu se refera la bordura, ci la intreg siteul. Daca siteul e scrollabale in jos, si gen in total are 2500px, pai y = 1800 mereu il va duce la y = 1800, indiferent unde e dat scroll si deci indiferent unde e bordura
* metoda asta nu are nimic de a face cu bordura, ia coordonatele in mod static, care nu se schimba niciodata.
* Problema e ca cand obtinem coordonatele unui element cu getBoundingClientReact, coordonatele se modifica in dependenta de scroll, si deci nu putem sa obtinem niciodata asta coordonatele cocnrete in pagina, fara a tine cont de bordura. De asta, luam in calcul si cat de mult a dat scroll userul:
* *window*.scrollTo(coord.x + *window*.scrollX, coord.y + *window*.scrollY);

coord – coordonatele elementului la care sa fie dus userul, care se modifica in dependenta de cum dam scroll, dar adunam si scrollul pentru ca sa fie mereu aceleasi coordonate, cele reale deci.

**window.scrollTo(object)**

* Nu e prea frumos ca mereu suntem teleportati la partea din site
* Facand asa, se va da scroll:
* *window*.scrollTo({  
   left: coord.x + *window*.scrollX,   
   top: coord.y + *window*.scrollY,  
   behavior: 'smooth'  
  });

**scrollIntoView()**

* Asta e o metoda mai noua. Dam doar element.scrollIntoView() si el va face toate aceste calcule cu coordonatele automat:
* Putem sa-i mai dam un parametru de tip object {behaviour: ‘smooth’}
* *infoSection*.scrollIntoView({behavior: 'smooth'})