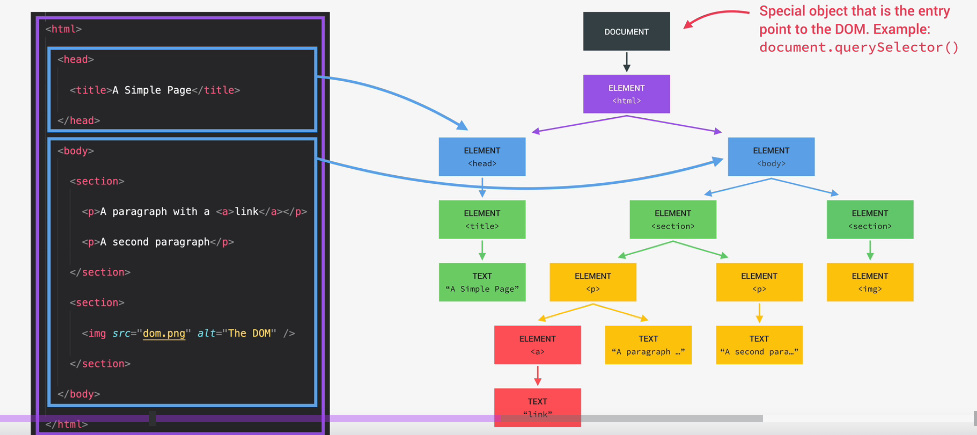
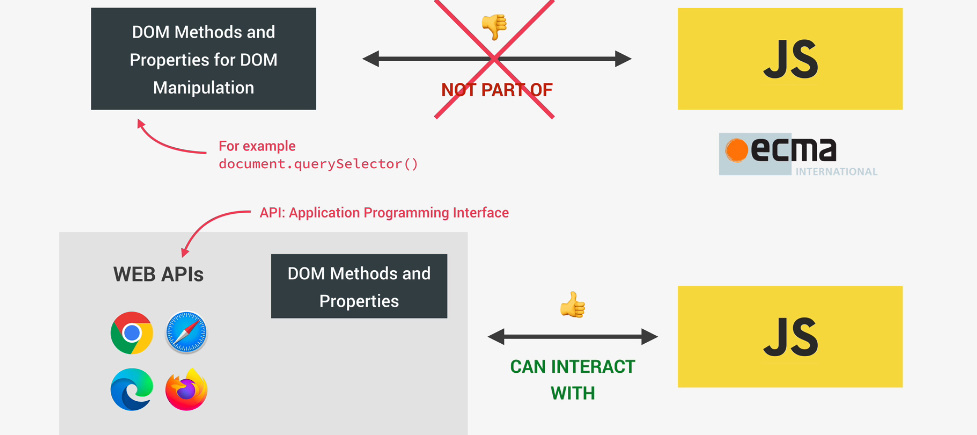
**DOM**

* **DOM** – Document Object Model – este o reprezentare structurata a unui document HTML. Permite accesarea elementelor HTML si styles pentru a le manipula
* **DOM** este un mod de a reprezenta pagina web intr-un mod ierarhic structurat pentru a fi mai usor de accesat si manipulat elementele paginii.
* Cu DOM putem obtine anumite elemente si styles de pe pagina web si sa le modificam cum vrem
* DOM e ca un fel de conexiune intre JS si pagina web
* DOM este creat automat de browser dindata ce pagina HTML este incarcata
* DOM este reprezentat printr-un tree:



Document este obiectul root, de la care incepem accesarea elementlor paginii, deci Document este obiectul paginii

* Cand browserul incarca pagina, JS nu intelege codul HTML, deci browser creaza o reprezentare a paginii, DOM, pentru a reprezenta pagina web **prin obiecte** in asa fel incat JS sa inteleaga cum ea e formata. De ex, JS nu intelege <h1> </h1>, dar intelege obiectul lui h1
* Document, adica DOM, nu este creat in JS! El este parte din WEB API, cu care JS poate interactiona



**Accesarea elementelor**

* **document.querySelector()** – selectam un item exact ca in CSS. Daca sunt mai multe, selecteaza doar primul. Deci, “.clasa” nu inseamna ca se va selecta clasa, ci itemul ce are o asa clasa. Daca de ex un item are clasele clasa1 si clasa2, selectand querySelector(“.clasa1”), itemul dat se va selecta, caci corespunde la query
* **.metoda.textContent** – returneaza textul din tag

**.metoda.textContent = “”** – setam textul

**.metoda.value –** returnam textul dintr-un **input** field

* **.getElementById()**
* **querySelectorAll()** – returneaza o lista cu toate elementele gasite ce corespund la query
* **getElementsByName()** – returneaza elementele dupa name dat

<h1 name=’test’> aa </h1>

* **getElementsByTagName(“p”)** – returneaza elementele dupa tag, gen p, a, h1 etc.
* **getElementsByClassName()**

**Click Events**

* **document.querySelector().addEventListener(event, listener)**

- **event** – string ce descrie eventul. Nu punem si on! De ex, ‘click’, nu ‘onclick’

- **listener** – functie ce raspunde la event

*document*.querySelector(".check").addEventListener('click', getInput);

function getInput(){  
 const input = *document*.querySelector(".guess").value;  
  
 checkInput(input);  
}

* **Math.random()** – returneaza numere intre 0 si 1,1 exclusiv, deci daca vrem sa avem intre 0 si 20, folosim \*20+1

Math.random()\*20+1, dar mai avem virgula

* Math.trunc(10.251) – scoate toate numerele dupa virgula

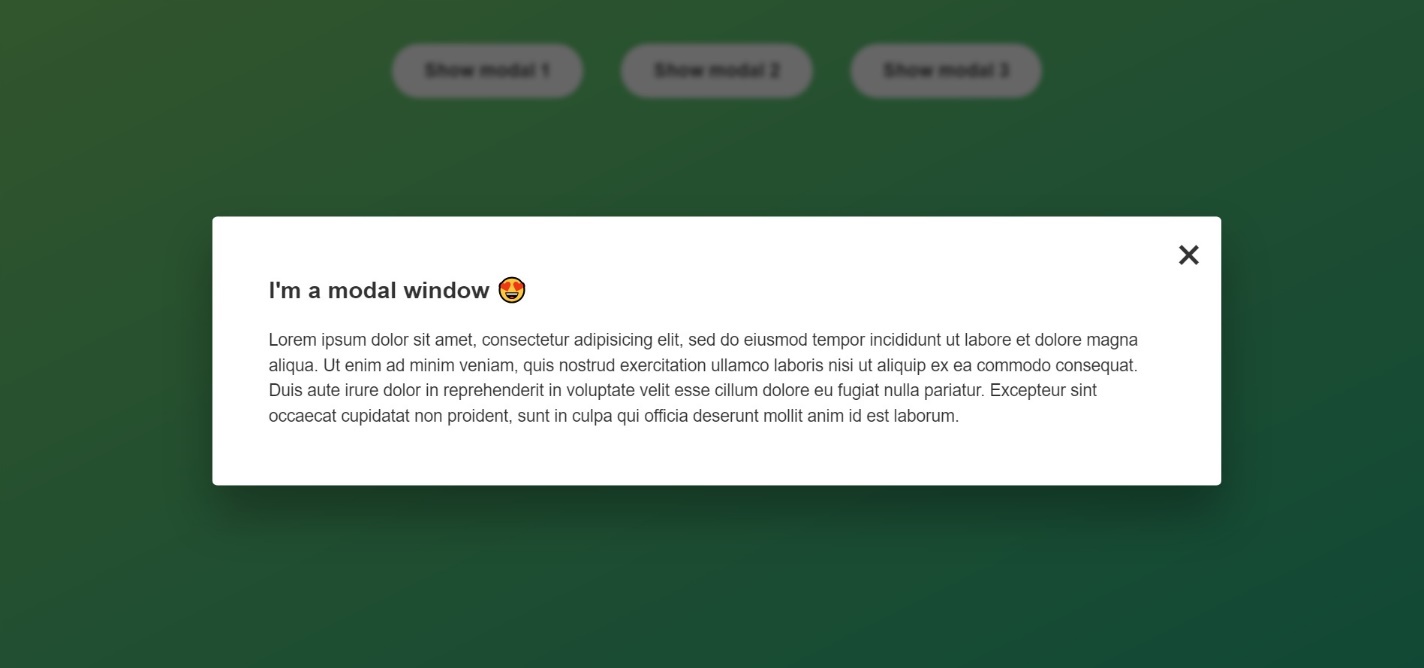
**Manipulating css styles**

* **document.metodaDeSelect().style.property = “”;**

**Mereu punem ca valoare un string! Doar string!**

Asa, proprietatile adauagate vor fi puse inline, nu in CSS file chiar.

**Cum facem o pagina ce are in centru un div si in spatele lui e ceata**



Cream un div ce va fi tot in ceata, adica cu asa style:

.overlay {  
 position: absolute;  
 top: 0;  
 left: 0;  
 width: 100%;  
 height: 100%;  
 background-color: rgba(0, 0, 0, 0.6);  
 backdrop-filter: blur(3px);  
 z-index: 5;  
}

backdrop-filter – adauga blur in spatele la elementele din div

Acum, peste acest div cream cel ce nu va fi in ceata:

.modal {  
 position: absolute;  
 top: 50%;  
 left: 50%;  
 transform: translate(-50%, -50%);  
 width: 70%;  
  
 background-color: white;  
 padding: 6rem;  
 border-radius: 5px;  
 box-shadow: 0 3rem 5rem rgba(0, 0, 0, 0.3);  
 z-index: 10;  
}

z-index conteaza, restul nu chiar.

Asa va fi un proiect complet, in care avem 3 butoane, si indiferent pe care nu apasam, se deschide un div si tot ce e in spatele lui e in ceata.

<body>  
 <button class="show-modal">Show modal 1</button>  
 <button class="show-modal">Show modal 2</button>  
 <button class="show-modal">Show modal 3</button>  
  
 <div class="modal hidden">  
 <button class="close-modal">&times;</button>  
 <h1>I'm a modal window 😍</h1>  
 <p>  
 Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod  
 tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim  
 veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea  
 commodo consequat. Duis aute irure dolor in reprehenderit in voluptate  
 velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint  
 occaecat cupidatat non proident, sunt in culpa qui officia deserunt  
 mollit anim id est laborum.  
 </p>  
 </div>  
 <div class="overlay hidden"></div>  
  
 <script src="script.js"></script>  
</body>

\* {  
 margin: 0;  
 padding: 0;  
 box-sizing: inherit;  
}  
  
html {  
 font-size: 62.5%;  
 box-sizing: border-box;  
}  
  
body {  
 font-family: sans-serif;  
 color: #333;  
 line-height: 1.5;  
 height: 100vh;  
 position: relative;  
 display: flex;  
 align-items: flex-start;  
 justify-content: center;  
 background: linear-gradient(to top left, #28b487, #7dd56f);  
}  
  
.show-modal {  
 font-size: 2rem;  
 font-weight: 600;  
 padding: 1.75rem 3.5rem;  
 margin: 5rem 2rem;  
 border: none;  
 background-color: #fff;  
 color: #444;  
 border-radius: 10rem;  
 cursor: pointer;  
}  
  
.close-modal {  
 position: absolute;  
 top: 1.2rem;  
 right: 2rem;  
 font-size: 5rem;  
 color: #333;  
 cursor: pointer;  
 border: none;  
 background: none;  
}  
  
h1 {  
 font-size: 2.5rem;  
 margin-bottom: 2rem;  
}  
  
p {  
 font-size: 1.8rem;  
}  
  
/\* -------------------------- \*/  
/\* CLASSES TO MAKE MODAL WORK \*/  
.hidden {  
 display: none;  
}  
  
.modal {  
 position: absolute;  
 top: 50%;  
 left: 50%;  
 transform: translate(-50%, -50%);  
 width: 70%;  
  
 background-color: white;  
 padding: 6rem;  
 border-radius: 5px;  
 box-shadow: 0 3rem 5rem rgba(0, 0, 0, 0.3);  
 z-index: 10;  
}  
  
.overlay {  
 position: absolute;  
 top: 0;  
 left: 0;  
 width: 100%;  
 height: 100%;  
 background-color: rgba(0, 0, 0, 0.6);  
 backdrop-filter: blur(3px);  
 z-index: 5;  
}

const *hiddenPage* = *document*.querySelector(".modal");  
const *blurPage* = *document*.querySelector(".overlay");  
const *buttons* = *document*.querySelectorAll(".show-modal");  
const *closeButton* = *document*.querySelector(".close-modal");  
  
*closeButton*.addEventListener('click',hidePage);  
*blurPage*.addEventListener('click', hidePage);  
  
for(let button of *buttons*)  
 button.addEventListener('click',showPage);  
  
function showPage(){  
 *blurPage*.classList.remove("hidden");  
 *hiddenPage*.classList.remove("hidden");  
}  
  
function hidePage(){  
 *blurPage*.classList.add("hidden");  
 *hiddenPage*.classList.add("hidden");  
}

**classList ()**

* **obj.classList.remove(“class”)** – sterge o clasa a unui element. Atentie sa nu punem punct in “”, ca deja e clar ca e clasa!
* **obj.classList.add(“class”)** – adauga o clasa
* **obj.classList.contains(“class”)**
* **obj.classList.toggle(“class”) –** daca asa clasa deja exista, o va sterge din acel element, daca nu, o va adauga

**Acces a property from tag**

* S-ar putea sa avem nevoie, de ex, sa accesam src din <img src=””/>
* Asta o facem asa:

**document.getElementById().src = ””**

Deci, accesam direct field cu asa proprietate si gata

**Global Event**

* In unele cazuri, am putea vrea ca pagina deasupra pusa sa fie inchisa apasand pe ESC, de ex.
* Nu e o idee prea buna sa punem un event pentru press button la un element concret, ci mai bine la intregul document
* **keyDown** – cand se apasa butonul

**keyUp** – cand butonul e lasat

**keyPress** – cand butonul e tinut apasat

* Problema e ca aceste events functioneaza pentru orice butoane, dar noi am vrea, de ex, doar pentru ESC
* Informatia despre care tasta a fost folosita o contine eventul
* Pentru asta, pur si simplu adaugam in functia eventului un parametru si cand va fi executat eventul, functia mereu va primi parametrul care este insusi eventul
* Fiecare event are diferite fields, ce contin informatii. Fieldul “key” contine butonul apasa, si anume pe el si il vom folosi

*document*.addEventListener("keydown",function(event) {  
 if(event.key==='Escape')  
 hidePage();  
})