Les interfaces de programmation applicative (API)

# Définition :

Quand vous utiliser un ascenseur, vous appuyer sur un bouton et vous savez que l’ascenseur va vous amener à l’étage sélectionner. En revanche vous avez aucune idée du fonctionnement interne de l’ascenseur (quelle puissance électrique doit recevoir chaque moteurs …)

Pour utiliser un ascenseur vous passez donc par **une interface utilisateur (une interface entre l’utilisateur et l’ascenseur)** qui simplifie l’utilisation de l’ascenseur.

Une API est une interface mais n’est pas une interface entre l’application et l’utilisateur mais interface entre 2 applications.

Vous avez enfaité déjà utiliser une API :

<https://www.php.net/manual/fr/mysqlinfo.api.choosing.php>

L’extension PDO est enfaitée une API. Vous savez comment créer et utiliser un objet PDO comment utiliser la méthode prepare() pour préparer une requête SQL sous la forme d’un objet de la classe PDOStatement et ensuite comment utiliser les méthodes excute() et feth() …

Mais vous ne savez comment chacune de ses méthodes a est coder.

PDO as pour but de permettre à votre application PHP d’interagir avec une BDD (développer des composants d’accès aux données). Mais il existe une très grande variété d’API.

Il y a 2 grandes sous catégories d’API :

* Les APIs intégrées : qui sont des APIs que l’on peut utiliser sans passer pas une application tierce. Exemple : BOM ([Lien MDN](https://developer.mozilla.org/fr/docs/Web/API/Window)), DOM ([Lien MDN](https://developer.mozilla.org/fr/docs/Web/API/Document)), [Liste des APIs JS](https://developer.mozilla.org/fr/docs/Web/API/Document)
* Les APIs externes : qui sont des APIs que l’on peut utiliser en passant par des applications tierces. Exemple : Suite API Google ( GoogleMap …), API Paypal ([Lien](https://developer.paypal.com/docs/api-basics/)), TinyMCE ([Lien](https://www.tiny.cloud/docs/demo/)).

Il est possible de créer vos propres APIs notamment sous la forme de service web (REST ou SOAP).

Exemple :

* <https://restcountries.com/>
* <http://www.omdbapi.com/>
* <https://public-apis.io/>
* …

Dans ce cours on va principalement se concentrer sur les APIs intégrées.

# BOM (Browser Object Model)

Le BOM peut être qualifier de Super API c’est une API qui contient elle-même d’autres APIs qui contiennent parfois elles même d’autres APIs…

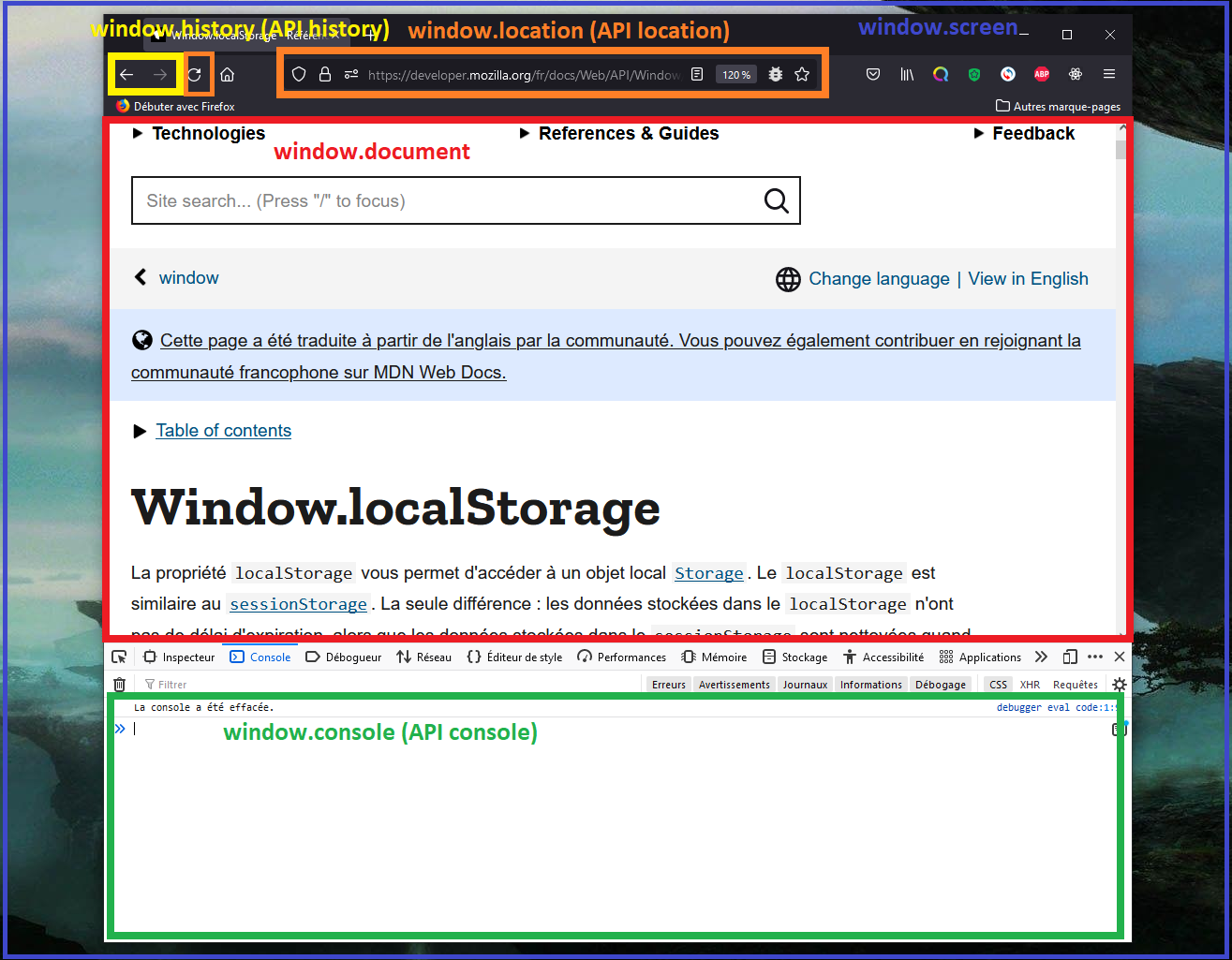
Le BOM est représenté par un objet window qui un objet de la classe Window.

<https://developer.mozilla.org/fr/docs/Web/API/Window>

<https://www.w3schools.com/jsref/obj_window.asp>

<https://www.pierre-giraud.com/javascript-apprendre-coder-cours/browser-object-model-window/>

De manière un peu simplifier l’objet window représente un onglet de la fenêtre de votre navigateur.



**La référence à l’objet window est implicite c’est-à-dire :**

window.alert() == alert()

window.navigator == navigator

L’objet window a plusieurs attributs notamment :

* **window.navigator** qui correspond à un objet de la classe Navigator (une autre API) window.navigator contient des informations propres au navigateur et à l’OS. <https://developer.mozilla.org/fr/docs/Web/API/Navigator>

<https://www.w3schools.com/jsref/obj_navigator.asp>

* Les dimensions externe et interne de la fenêtre de navigation **window.innerHeight, window.innerWidth, window.outerHeight, window.outerWidth**.
* Les dimensions de votre écran (est non pas de la fenêtre) sont accessible par **window.screen** (aussi utiliser pas les medias querry). Note : le mode responsive modifie les valeurs. <https://www.w3schools.com/jsref/obj_screen.asp>
* L’historique de la session de navigation (API History) est accessible par **window.history** <https://developer.mozilla.org/fr/docs/Web/API/Window/history>.
* window.console permet d’utiliser l’API Console <https://developer.mozilla.org/fr/docs/Web/API/Window/console>
* **window.location** (API Location) permet de consulter ou modifier l’adresse web (URI/URL) <https://developer.mozilla.org/fr/docs/Web/API/Window/location>
* **window.sessionStorage** permet au navigateur (client) de mémoriser des informations qui seront effacer quand l’onglet sera fermer <https://developer.mozilla.org/fr/docs/Web/API/Window/sessionStorage>
* **window.localStorage** permet au navigateur de mémoriser des informations qui ne seront pas effacer même si on ferme le navigateur. <https://developer.mozilla.org/fr/docs/Web/API/Window/localStorage>
* **window.document** (API DOM) un objet qui représente le contenue de la page

En plus des attributs il est possible d’utiliser des méthodes à partir de l’objet window.

* **window.alert()**
* **window.confirm()**
* window.find()
* window.close()
* **window.prompt()**
* **window.setInterval()** une méthode qui de programmer l’exécution de code à intervalle régulier [lien](https://developer.mozilla.org/en-US/docs/Web/API/setInterval) .
* **window.setTimeout()** une méthode qui permet de retarder l’exécution de code d’un certain temps [lien](https://developer.mozilla.org/en-US/docs/Web/API/setTimeout).
* …

Note si vous avez définit des fonctions dans un fichier .js elles deviennent automatiquement des méthodes de l’objet window.

# Le DOM (Document Object Model)

<https://developer.mozilla.org/fr/docs/Glossary/DOM>

<https://developer.mozilla.org/fr/docs/Web/API/Document_Object_Model>

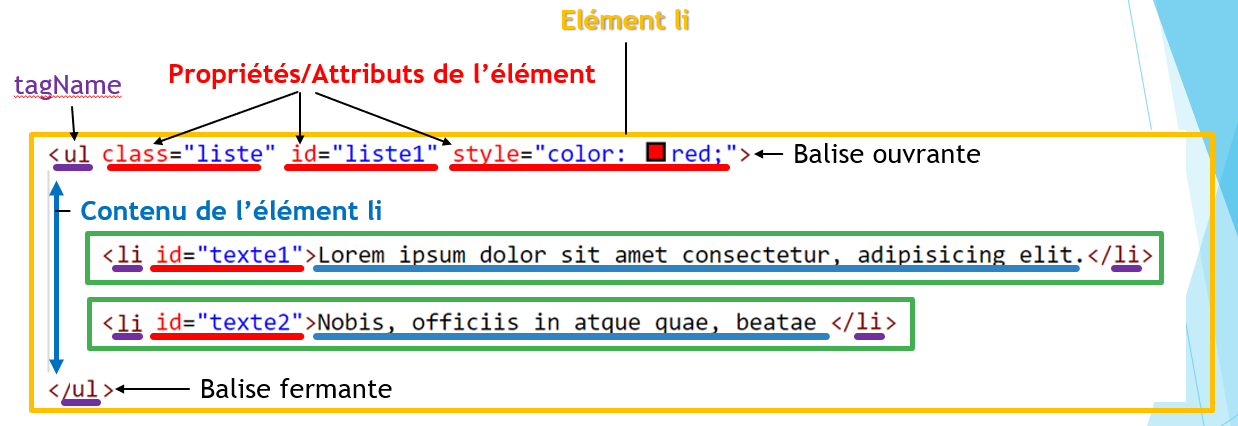
<https://www.pierre-giraud.com/javascript-apprendre-coder-cours/presentation-dom/>

<https://www.w3schools.com/jsref/dom_obj_document.asp>

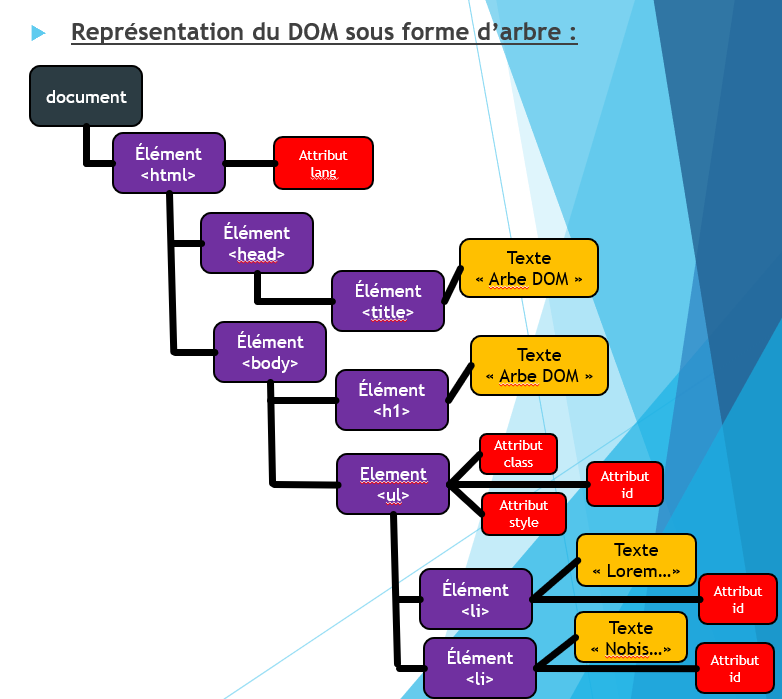
C’est une API qui a pour but de représenter le code HTML/CSS de votre page est de l’organise sous la forme d’un objet document (de la classe Document).

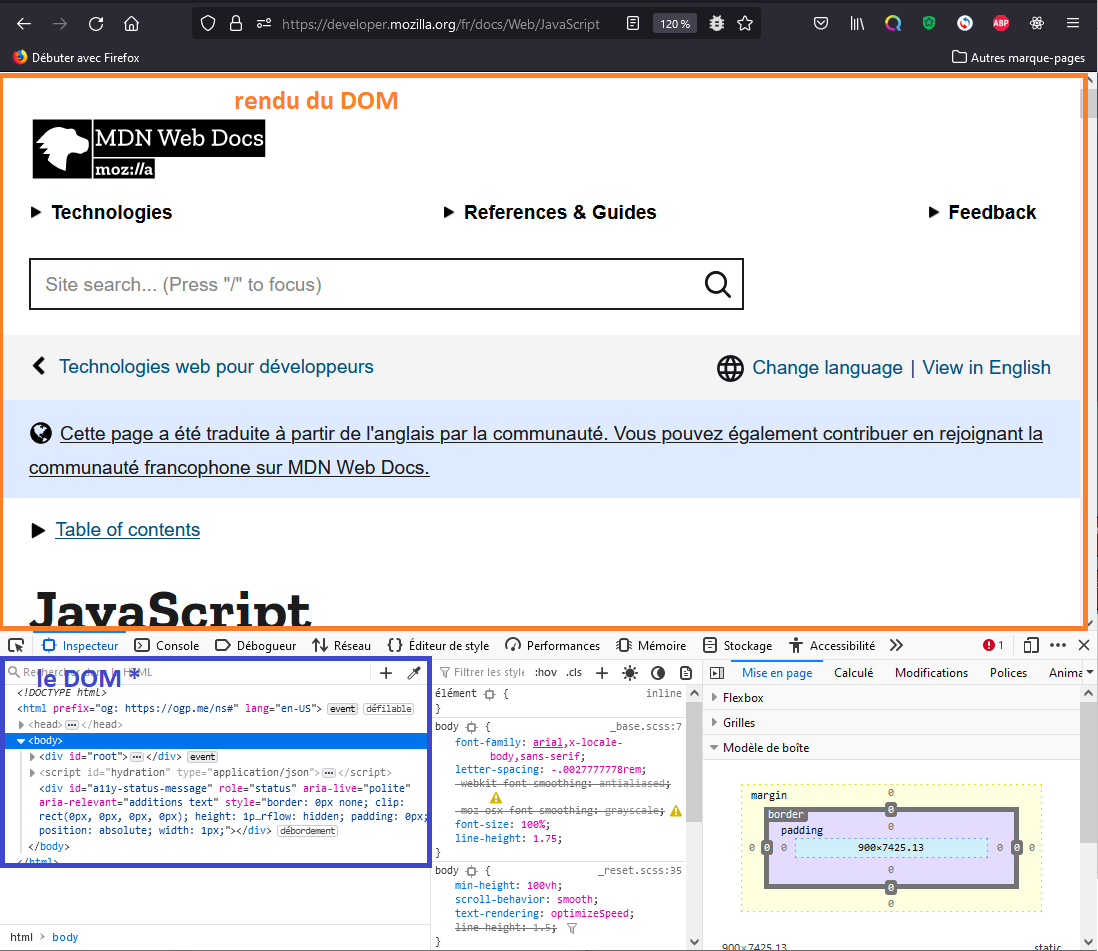
<https://developer.mozilla.org/fr/docs/Web/API/Document>

Rappel :



**On peut représenter le DOM sous la forme d’arborescence :** 





Un peut comme dans un arbre généalogique il existe des nœuds.

Les nœuds permettent de représenter des morceaux du document :

* + Un élément html est un type de nœud
  + Un texte est un type de nœud
  + Une propriété d’un élément est un type de nœud
  + Un commentaire est un type de nœud
  + L’objet document est lui aussi un type de nœud

**Les APIs du DOM :**

L’API Document permet de représenter le document via l’objet document.

L’API Node est utilisée dans plusieurs objets du DOM (comme par exemple : document, les objets de type Element …)

<https://developer.mozilla.org/fr/docs/Web/API/Node>

L’API Element est une classe qui permet de représenter un élément html (une balise)

<https://developer.mozilla.org/fr/docs/Web/API/Element>

<https://www.w3schools.com/jsref/dom_obj_all.asp>

L’API Event permet de gérer les évènements du DOM

<https://www.w3schools.com/jsref/dom_obj_event.asp>

L’API EventTarget que des objets du DOM vont implémenter quand ils peuvent recevoir des évènements.

…

En JS le DOM va nous permettre de modifier n’importe quel élément de notre page. On peut ajouter un contenu a un élément, modifier les classes ou les styles qui s’applique sur un élément. Mais on peut également gérer les éléments qui s’applique sur chaque élément.

Pratique :

Pour récupérer **un élément HTM**L sous la forme d’un objet de la classe Element :

On peut utiliser **document.querySelector()** qui prend **un selecteur css** **sous forme de string** :

<https://developer.mozilla.org/fr/docs/Web/API/Document/querySelector>

On peut également utiliser **document.getElementById()** qui prend **un id sous forme de string** :

<https://developer.mozilla.org/fr/docs/Web/API/Document/getElementById>

Dans les deux cas on nous retourne **un** objet de la classe Element (si on trouve l’élément) ou alors null.

Pour récupérer **plusieurs éléments**:

On peut utiliser **document.querySelectorAll()** qui prend **un selecteur css sous forme de string** en argument et retourne un objet NodeList (un objet similaire a un array de nœud)

<https://developer.mozilla.org/fr/docs/Web/API/Document/querySelectorAll>

On peut utiliser **document.getElementsByClassName()** qui prend en argument **le nom d’une classe(HTML) sous forme de string** et retourne un objet HTMLCollection similaire a un array d’Element.

<https://developer.mozilla.org/fr/docs/Web/API/Document/getElementsByClassName>

Dans les deux cas on peut considérer que on obtient une sorte d’array. Mais attention c’est une simplification.

Voir aussi :

<https://developer.mozilla.org/fr/docs/Web/API/Document/getElementsByName>

<https://developer.mozilla.org/fr/docs/Web/API/Document/getElementsByTagName>

Manipuler un objet de la classe Element :

Les Propriétés :

* + La propriété innerHTML d’un élément permet d’accéder au contenu d’un élément
  + La propriété outerHTML d’un élément permet d’accéder au code HTML de l’élément et de son contenu
  + La propriété innerText permet d’accéder au texte d’un élément ou nœud
  + La propriété style d’un élément permet d’accéder à son style CSS sous la forme d’un objet; on peut accéder aux propriétés CSS comme color, backgroundColor …
  + La propriété classList permet d’accéder aux classes d’un élément retourner dans un objet (pratique pour la mise en forme et en combinaison avec Bootstrap) **voir le lien suivant** <https://developer.mozilla.org/fr/docs/Web/API/Element/classList>
  + La propriété id permet d’accéder à la propriété id d’un élément

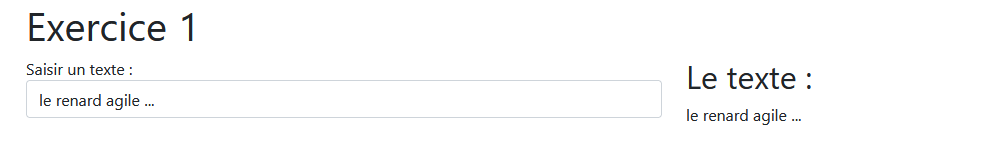
Les méthodes :

* + La méthode getAttributeNames() permet de connaitre les clés des différents attributs de l’élément
  + La méthode getAttribute(« nomAttribut ») permet de connaitre la valeur d’un attribut
  + La méthode setAttribute(« clé »,  «valeur ») permet d’ajouter un attribut
  + La méthode addEventListener qui peut appeler soit depuis un objet Element ou Document permet d’ajouter un événement le premier paramètre et le type d’évènement en string le deuxième paramètre est une fonction (callback). Voir <https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener>

# **Exercice 1 : Afficher un texte**

L’utilisateur tape un texte dans un input, et le texte s’affiche dans une div.

Exemple :



# **Exercice 2 : Modifier la couleur d’une div**

L’utilisateur sélectionne une couleur la div à côté prend la couleur sélectionnée.



Pour modifier le style css via le JS :

<https://developer.mozilla.org/fr/docs/Web/API/HTMLStyleElement>

# **Exercice 3 : Thème jour/nuit**

Il y a deux boutons sur le page et plusieurs textes.

Quand l’utilisateur clique sur le bouton nuit, la couleur du fond des textes deviennent sombre et la couleur de la police claire.

Inversement quand l’utilisateur clique sur le bouton jour, la couleur de fond devient claire et la police des textes deviennent sombre.

**Attention il y a plusieurs textes**

(Il est possible d’affecter le style CSS via le JS ou d’affecter les classes Bootstrap qui s’applique sur les éléments)

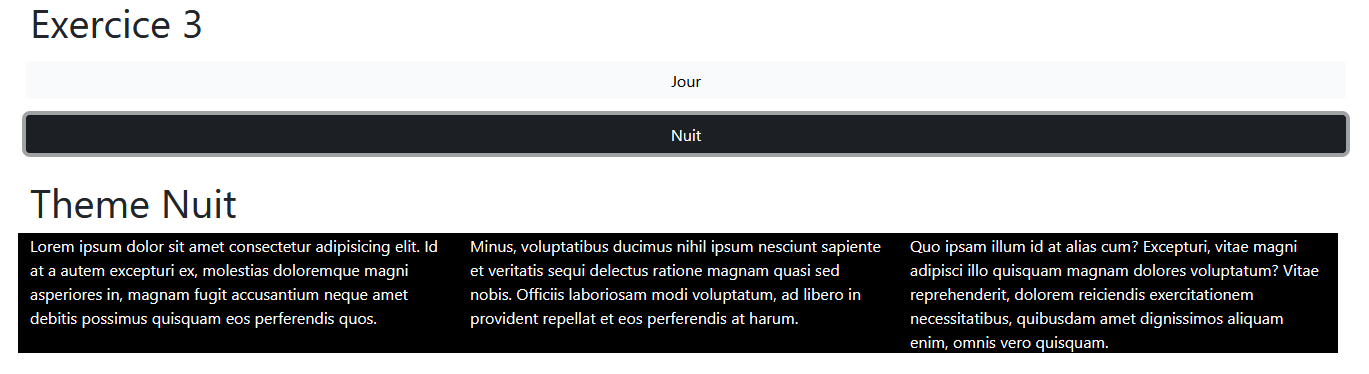


Figure 1 mode nuit

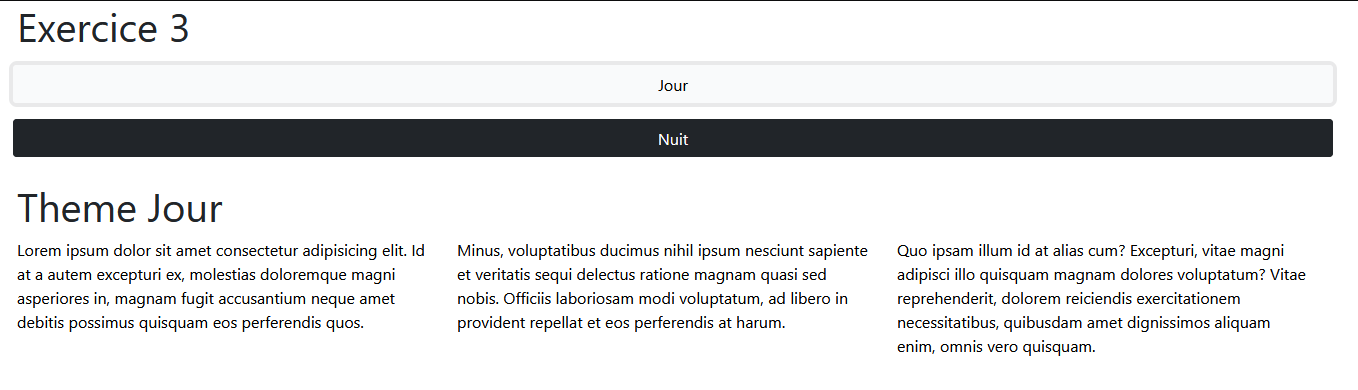


Figure 2 mode jour

**Améliorations** :

* On peut aussi affecter les couleurs du body
* On peut afficher le thème (Jour ou Nuit) dans le h1

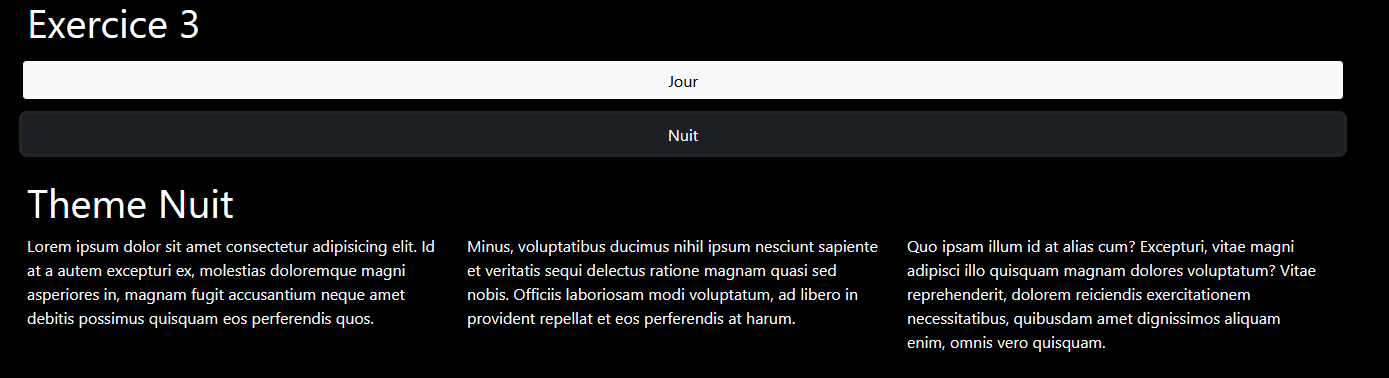


Figure 3 mode nuit avec les améliorations

**Super-amélioration :**

* On peut garder ne mémoire le dernier choix de thème de l’utilisateur après qu’il quitte la page utiliser l’API localStorage [lien](https://developer.mozilla.org/fr/docs/Web/API/Window/localStorage).
* Au chargement de la page on vérifie s’il y a un thème enregistrer dans le localStorage.

# **Exercice 4**,5,6

L’utilisateur clique sur un bouton alors la source de l’image doit changer pour faire apparaitre l’image de la bonne couleur.

Les images sont toutes dans le dossier img.

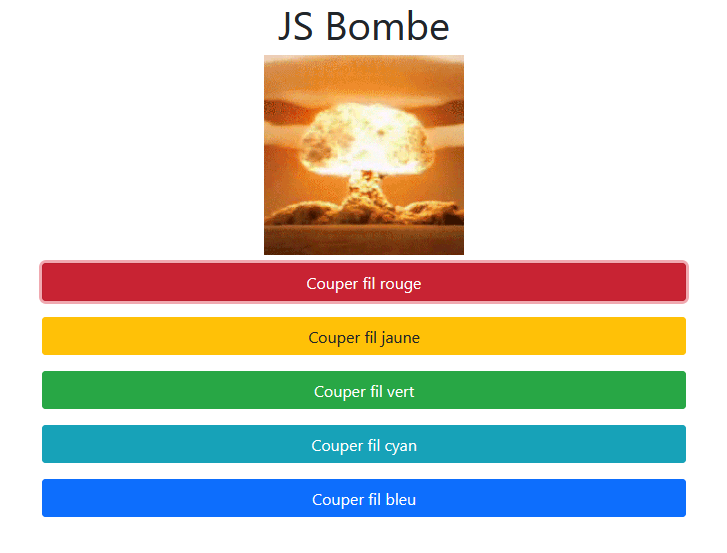
Indice : voir le nom des images et la valeur (attribut value) des boutons.

# Exercice 7 : JS Bombe (difficile + recherche)

L’utilisateur peut couper des fils en fonction des fils qu’il coupe est de sa chance le bombe va réagir différemment.

Suivre les commentaires et intervenir sur les annotations Todo

Fil rouge : la bombe explose



Fil jaune : une chance sur deux le timer gagne ou perd 1 minute

Fil vert : la bombe est désamorcer

Fil cyan : le temps passe deux fois plus vite (interval)

Fil bleu : l’ecran tombe en panne (mais la bombe explose quand même si elle ateint 0)