RAPPORT WEB

Anne Passelègue
Fanny Ruiz

Table des matières

Concept du site

Notre site web est le site d'un musée fictif appelé Musée Bretcho. Il est possible pour l'utilisateur de consulter la liste des œuvres que possède le musée (ces œuvres sont renvoyées grâce à l'API de l'institut d'art de Chicago). L'utilisateur peut aussi consulter les détails techniques d'une œuvre ainsi que des informations sur l'accessibilité à notre musée.

HTML : contenu du site

Page d'accueil

Cette première page permet d'accueillir l'utilisateur sur le site.
En haut, il y a un header qui permet de se déplacer sur les différentes pages du site (accueil, collection et contact). Pour réaliser ce menu, nous avons utilisé une balise <nav> car elle a une valeur sémantique, elle est adaptée pour un menu de navigation. Pour lister les éléments du menu, nous avons utilisé une liste non ordonnée : balise <ul>. Nous avons fait ce choix car nous n'avons pas besoin d'afficher de chiffres à côté des éléments de notre liste, mais aussi parce qu'il n'y a pas vraiment d'importance dans l'ordre des éléments de notre menu.
Nous avons intégré le logo du musée (que nous avons réalisé) dans cette liste pour pouvoir le voir dans chaque page, mais aussi car ce logo correspond un à lien permettant de rejoindre la page d'accueil.

```
<nav>
    <ul>
        <li><a href="accueil.html"> <img id="logoBretcho" src="Ressources/logo.png" alt="Logo du Musée Bretcho"> </a> </li>
        <li class="menu"><a href="accueil.html">Accueil</a></li>
        <li class="menu"><a href="HTML/collection.html">Collection</a></li>
        <li class="menu"><a href="HTML/contact.html">Contact</a></li>
    </ul>
</nav>
```

Au niveau de la vidéo, nous avons utilisé la balise <video> et avons mis deux sources disponibles au cas où le navigateur n'arrive pas à lire le premier lien. Nous avons utilisé les attributs suivants : autoplay qui permet une lecture de la vidéo automatique lors de l'ouverture de la page d'accueil, muted qui permet d'avoir une vidéo muette, playsinline qui permet de lire la vidéo dans la zone de lecture de l'élément et enfin loop qui permet de lire la vidéo en continu.

```
<video autoplay muted playsinline loop>
    <source src="Ressources/video_en-tete.mp4" type="video/mp4" />
    <source src="Ressources/video_en-tete.webm" type="video/webm" />
</video>
```

Notre footer est divisé en 2 grandes parties.
La première partie est une flexbox car nous souhaitons une mise en page automatique et responsive sur 1 axe qui est divisée en 3 sections. Nous avons utilisé la balise <section> afin d'avoir une valeur sémantique qu'une ne nous apporte pas. La première section permet l'accès à la page d'accueil et à la page contact par des liens. La deuxième section permettrait, dans le cadre d'un site complet, d'accéder à toutes les pages du site. Et enfin, la dernière section contient des icones de réseaux sociaux qui permettent d'emmener l'utilisateur directement dans les différents comptes du musée (dans notre cas, comme nous n'avons pas des vrais comptes, les liens emmènent seulement sur la page d'accueil des réseaux sociaux).
La deuxième partie de notre footer est également une flexbox pour les mêmes raisons. Elle contient une section contenant également des liens.

Si nous entrons dans les détails, nous avons utilisé deux @médias screen afin d'avoir un visuel plus agréable pour des écrans avec des plus petites dimensions tels qu'un téléphone portable.

Un favicon a aussi été ajouté au site.

```
<link rel="icon" href="/Ressources/favicon.ico" />
```

Page de collection

Cette page permet de voir une liste de toutes les œuvres présentes au Musée Bretcho.
L'API pouvant mettre un certain temps à renvoyer les données nécessaires à l'affichage des œuvres, on affiche un loader pour faire patienter l'utilisateur. Ce loader est une à laquelle on va attribuer des propriétés CSS, telles qu'un border-top noir épais et arrondis, ainsi qu'une animation qui la fait tourner à 360° à l'infini.

```
<div id="conteneurLoader">
    <div id="loader"></div>
</div>
```

```
#loader {
    border: 16px solid var(--fond);
    border-top: 16px solid black;
    border-radius: 50%;
    width: 120px;
    height: 120px;
    animation: spin 2s linear infinite;
    display: block;
}
```

```css
@keyframes spin {
    0% {
        transform: rotate(0deg);
    }
    100% {
        transform: rotate(360deg);
    }
}
```

Une fois les données de l'API récupérées, on fait disparaitre le loader en JavaScript en lui donnant la propriété CSS « display none ».

```javascript
document.getElementById("conteneurLoader").style.display = "none"
```

La page HTML contient également le code du template d'une qui sera ensuite clonée puis remplie de données en JavaScript avec les images et les informations concernant les œuvres récupérées via l'API. Dans cette , on trouve une balise <img> dans laquelle on affichera l'image des œuvres, une balise <h1> pour le titre des œuvres et une balise <h2> pour le nom de leur artiste.

```html
<div id="apercu-oeuvre-template" class="apercu-oeuvre">
    <a id="" href="oeuvre.html">
    <img src="" srcset="" alt="Photo d'euvre d'art">
    <h1></h1>
    <h2></h2>
    </a>
</div>
```

Lorsqu'on passe la souris sur une de ces , une ombre apparait autour pour la mettre en valeur, et pour indiquer à l'utilisateur qu'il peut cliquer sur cette . En cliquant dessus, il pourra accéder à une page qui donne des détails sur cette œuvre.

```css
main .apercu-oeuvre:hover {
    box-shadow: 0px 0px 10px 5px rgb(165, 165, 165);
    transition: all 0.5s ease;
}
```

À la fin de notre collection, nous avons créé un bouton permettant de retourner en haut de la page. Pour cela nous avons créé une dans le fichier HTML permettant d'afficher le texte « Retour en haut de la page ». Ensuite, dans le fichier CSS, nous avons mis en forme cette div afin qu'elle agisse comme un bouton. Puis enfin, en JavaScript, nous avons écrit une méthode afin que lorsque l'utilisateur appuie sur ce bouton, la page remonte doucement jusqu'à arriver en haut de la page.

Page d'une œuvre

Cette page permet de voir le détail d'une œuvre. On y trouve une balise <img> qui contiendra l'image de l'œuvre, puis une qui contiendra le nom de l'œuvre (<h1>), son artiste (<h2>), sa date de création (<h4>) et des détails techniques (<p>). Ces éléments HTML seront remplis en JavaScript avec les données de l'API.

```
<img id="imageOeuvre" src="" srcset="" alt="Photo d'oeuvre d'art">
<div id="informations">
    <h1></h1>
    <h2></h2>
    <h3></h3>
    <p></p>
</div>
```

En attendant l'apparition des données, l'utilisateur peut aussi patienter avec un loader.

Page de contact

La page contact contient tous les emails et numéros permettant à l'utilisateur du site de contacter le musée. Chaque partie contient un titre (<h1>) puis un paragraphe avec les informations importantes en mise en gras à l'aide de la balise <b> ou une liste <ul> car les éléments ne sont pas ordonnés.

CSS : Style du site

Chaque page contient deux fichiers CSS : un fichier qui est propre à la page, et un fichier général présent dans toutes les pages. Ce dernier permet de styliser les éléments HTML basiques que l'on retrouvera partout (liens, boutons, loader…) ainsi que d'agencer le header et le footer.

JavaScript : remplissage et animation du site

Affichage de toutes les œuvres

L'API que nous utilisons renvoie énormément d'œuvres, donc il y a une pagination (12 œuvres par page). Cependant, beaucoup d'œuvres ne contiennent pas d'images et peu d'information, donc nous ne les afficherons pas à l'écran. C'est pourquoi nous devons récupérer les œuvres d'un certain nombre de pages pour avoir un minimum d'œuvres traitables, ce qui peut prendre un certain temps à charger.
Pour récupérer les œuvres de chaque page, on fait une boucle qui fait un appel à l'API page par page, pour un nombre de pages défini (ici 10). Les données récupérées seront ensuite ajoutées à un tableau (*listeOeuvres*). Ce tableau contiendra à la fin 10 listes de 12 œuvres.

```
// Nombre de pages de données que l'on veut récupérer dans l'API
// L'API contenant beaucoup de données, elle est divisée en plusisurs pages
const nbPages = 10

// Récupération des données de l'API pour le nombre de pages souhaité
for (let page = 0; page <= nbPages; page++) {
    // Changement de page dans l'URL de notre API
    let url = 'https://api.artic.edu/api/v1/artworks?page=' + page

    // Fetch de l'URL
    fetch(url)
    .then(function(response) {···
        })
        .then (function(data) {
            //Traitement des données
            listeOeuvres.push(data.data)

            // Test si toutes les données ont été récupérées
            if (page == nbPages) {
                affichageOeuvresRecuperees()
            }
        })
        .catch(function (err) {···
    });
}
```

Une fois les 10 pages d'œuvres récupérées, on appelle une fonction *affichageOeuvresRécupérées()* qui se chargera de cloner la template de pour la remplir des bonnes informations pour chaque œuvre. On ajoute aussi l'id de l'œuvre à l'attribut « id » de la .

```
// Clonage du template de div à remplir avec les bonnes données
let div = template.cloneNode(true)
div.id = listeOeuvres[i][j].id
```

L'API ne renvoie pas directement le lien des images des œuvres, mais l'id de leur image. Cet id sera à intégrer dans une URL qu'il faudra placer dans l'attribut « src » des balises <img>. Il faudra aussi ajouter cette URL accompagnée d'un peu plus de paramètres dans un attribut « srcset » des balises <img>.

```
// Affichage de l'image
div.querySelector("img").src = "https://www.artic.edu/iiif/2/" + listeOeuvres[i][j].image_id
div.querySelector("img").srcset = "https://www.artic.edu/iiif/2/" + listeOeuvres[i][j].image_id + "/full/200,/0/default.jpg"
```

Pour chaque , on ajoute aussi un évènement qui fait que quand on clique dessus, l'attribut « id » de la va s'enregistrer dans le localStorage du navigateur.

```
// Changement de page quand on clique sur l'oeuvre
div.onclick = function() {
    localStorage.clear()
    localStorage.setItem("idOeuvre", this.getAttribute('id'));
}
```

Affichage des détails d'une œuvre

Lorsqu'on arrive sur la page de l'œuvre sur laquelle on a cliqué, on récupère l'id de l'œuvre dans le localStorage du navigateur, puis on fait un appelle à l'API en utilisant cet id pour pouvoir récupérer les données de l'œuvre voulue.

```
// Récupération de l'id de l'oeuvre à afficher
const idOeuvre = localStorage.getItem("idOeuvre")

// URL de notre oeuvre
const url = 'https://api.artic.edu/api/v1/artworks/' + idOeuvre
```

Une fois les données récupérées via l'API, on les affiche dans les balises du DOM.