Centre de Réadaptation de Mulhouse

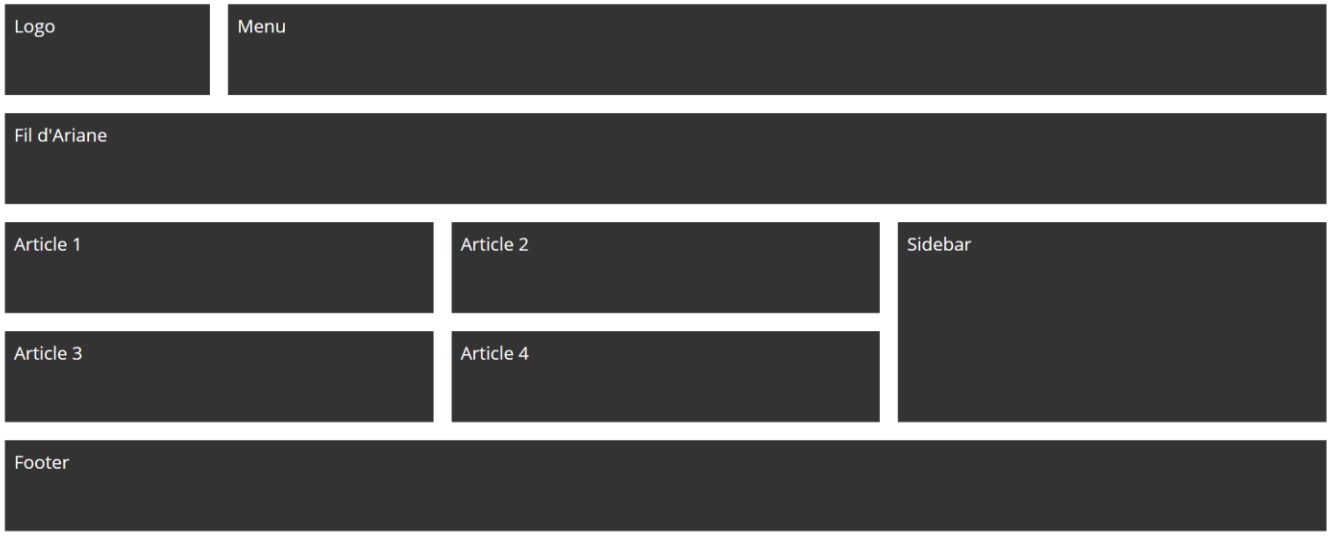
CREATION D'UNE MISE EN PAGE RESPONSIVE SIMPLE AVEC LES GRID CSS
MANIPULATION DES GRILLE CSS ET DES MEDIA-QUERIES EN CSS

# CRÉATION D'UNE INTERFACE BASIQUE EN HTML 5 A L'AIDE DES GRID CSS

## 1) CREATION DE LA GRILLE.

Création de la grille

Nous souhaitons réaliser la mise en page suivante, basée sur 6 colonnes :




Commençons par créer une structure HTML toute simple avec des classes :

```
<body>
 <header class="logo box">
  Logo
 </header>
```

```
<nav class="menu box">
  Menu
</nav>

<div class="fil box">
  Fil d'Ariane
</div>

<main class="main">
  <article class="box">
    Article 1
  </article>
  <article class="box">
    Article 2
  </article>
  <article class="box">
    Article 3
  </article>
  <article class="box">
    Article 4
  </article>
</main>
```

```
 <aside class="sidebar box">
  Complément information
 </aside>

 <footer class="footer box">
  Pied de page
 </footer>
</body>
```

Et quelques lignes de CSS pour démarrer :

```
body
{margin : 0px
 padding: 0px;

}

.box
{
 background: #333;
 color: #fff;
 padding: 20px;
}
```

Etape 1 ) La première chose à faire est de définir notre Grid Container. Nous devons donc déclarer 6 colonnes de largeur égale et 4 lignes de hauteur libre. De plus nous avons des espacements de 20px entre chaque colonne et chaque ligne :

Les propriétés grid-template-columns et grid-template-rows permettent respectivement de déclarer les colonnes et les lignes, grid-gap permet de définir les espacements. Etant donné que nous souhaitons la même largeur pour toutes les cellules, nous utilisons la fonction repeat pour boucler sur le nombre qui nous intéresse. L'unité fr représente une fraction de l'espace à occuper par la cellule (à la manière des flexbox avec flex: 1). C'est une unité pratique car sa valeur est automatiquement calculée par le navigateur en fonction de la largeur ou de la hauteur disponible dans le Container.

Par défaut, les Grid Items (enfants directs du Grid Container) viennent chacun leur tour remplir les cases de la grille (l'ordre d'apparition dans le code HTML est alors respecté). Dans notre cas, nous avons 6 Grid Items (logo, menu, fil d'Ariane, contenu principal, sidebar et footer) et la totalité de la grille devrait être remplie.

Etape 2) Positionnement des Grid Items

Nous souhaitons maintenant positionner nos Grid Items dans les bonnes cases avec les bonnes dimensions. Voici les nouvelles règles CSS à créer :

Pour le moment, nous n'agissons que sur le positionnement par rapport aux colonnes de notre layout grâce à la propriété grid-column (qui est un raccourci pour grid-column-start et grid-column-end) placée sur chacun de nos Grid Items. Cette propriété permet tout simplement de définir une colonne de départ / une colonne d'arrivée ou bien une plage à occuper par le Grid Item (NB : Les index des colonnes commencent à 1 et non à 0).

Le logo étant déjà correctement positionné (1ère colonne, 1ère ligne), nous ne le modifions pas.

Le menu, doit occuper le reste de la 1ère ligne. Nous lui affectons donc 5 colonnes grâce au mot-clé span. N'ayant plus de place pour se loger sur la 1ère ligne, les Grid Items suivants sont automatiquement renvoyés à la suivante.

Le fil d'Ariane doit occuper une ligne complète, c'est pourquoi nous le faisons démarrer à la 1ère colonne et le faisons terminer à la dernière en utilisant un chiffre négatif qui permet de compter à partir de la fin. Nous aurions pu écrire span 6 mais c'était pour utiliser une autre syntaxe.

Pour le contenu principal et la sidebar, nous partons sur une répartition 2/3 - 1/3 avec respectivement span 4 et span 2.

Le footer reprend les mêmes propriétés que le fil d'Ariane.

## 2) IMBRICATION DE GRIDS.

Le cas que nous avons vu est idéal. Les Grids Items apparaissent dans le code HTML dans l'ordre qui nous convient le mieux et nous n'avons qu'à agir sur les colonnes.
On pimente les choses, voyons de façon plus générique, la manipulation des grids  supposons maintenant que le code HTML est le suivant :

```
<body>
 <footer class="footer box">
  Footer
 </footer>

 <aside class="sidebar box">
  Sidebar
 </aside>

 <nav class="menu box">
  Menu
 </nav>

 <header class="logo box">
  Logo
 </header>

 <main class="main">
  <article class="box">
   Article 1
  </article>
  <article class="box">
```

```
    Article 2
   </article>
   <article class="box">
    Article 3
   </article>
   <article class="box">
    Article 4
   </article>
  </main>

  <div class="breadcrumb box">
   Fil d'Ariane
  </div>
</body>
```

Les éléments sont volontairement dans le désordre au niveau du HTML.
Nous allons uniquement utiliser la propriété grid-row pour rétablir l'ordre naturel des choses. Voici donc le code CSS mis à jour :
Etape 3) Essayer d'obtenir la même interface que précédemment avec la,propriété Grid-row

A l'instar de grid-column, la propriété grid-row (qui est un raccourci pour grid-row-start et grid-row-end) permet de positionner les Grids Items par rapport à des lignes avec le même principe (un début / une fin ou une plage à occuper). Dans notre cas, seule la ligne de début est nécessaire pour remettre les éléments sur la bonne rangée.

Le menu apparaissant avant le logo et la sidebar avant le contenu principal dans le nouveau code HTML, vous noterez les légères retouches apportées aux propriétés grid-column de ces éléments pour notamment spécifier les colonnes de début

Pour positionner les articles à l'intérieur du contenu principal .
Nous avons besoin de repartir sur une nouvelle Grid de 2 colonnes de largeur égale ainsi qu'un nombre de ligne indéterminé (puisqu'il peut potentiellement y avoir une infinités d'articles). Nous allons donc étendre les possibilités du Grid Item concerné en le transformant en Grid Container :

Solution possible :

```
.main
{
  grid-column: 1 / span 4;
  grid-row: 3;
  display: grid;
  grid-template-columns: 1fr 1fr;
  grid-auto-rows: minmax(100px, auto);
  grid-gap: 20px;
}
```

## 3) UTILISATION DES GRID-AREA

Depuis le début, nous déclarons un Grid Container et son template (colonnes / lignes) et positionnons les Grid Items "à la main" en spécifiant des pseudo-coordonnées (grid-column et grid-row).
Il existe une autre manière de procéder qui peut s'avérer être plus élégante à écrire dans certains cas en utilisant les Grid Areas. Nous allons modifier le code CSS de la manière suivante :

```
body
{
  display: grid;
  grid-template-columns: repeat(6, 1fr);
  grid-template-rows: repeat(4, auto);
  grid-gap: 20px;
  grid-template-areas:  "logo menu menu menu menu menu"
                        "breadcrumb breadcrumb breadcrumb breadcrumb breadcrumb breadcrumb"
                        "main main main main sidebar sidebar"
                        "footer footer footer footer footer footer";
}

.logo
{
  grid-area: logo;
}

.menu
{
  grid-area: menu;
}

.breadcrumb
{
  grid-area: breadcrumb;
}

.main
{
  grid-area: main;
  display: grid;
  grid-template-columns: 1fr 1fr;
```

```
  grid-auto-rows: minmax(100px, auto);
  grid-gap: 20px;
}

.sidebar
{
  grid-area: sidebar;
}

.footer
{
  grid-area: footer;
}
```

## 4) LE RESPONSIVE DESIGN ET LES « GRID CSS »

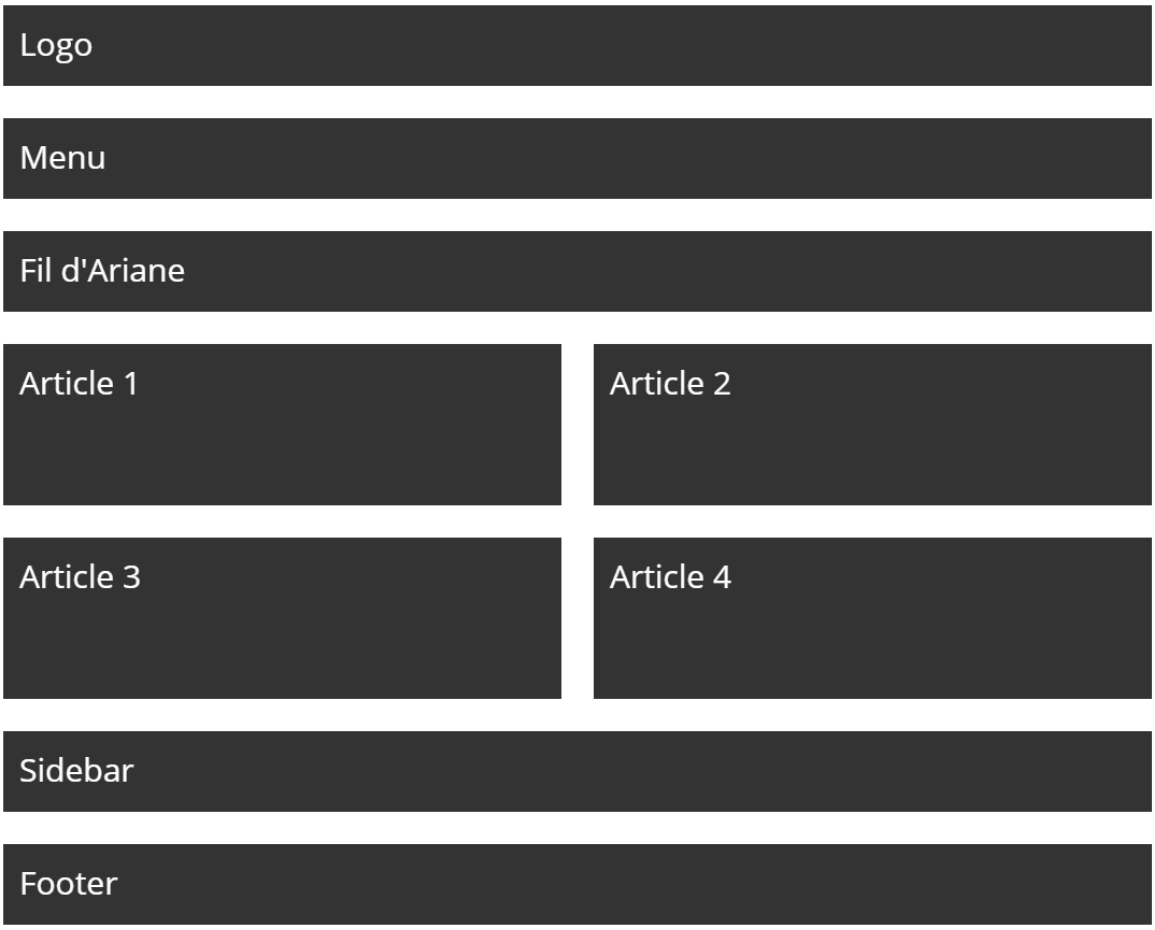
Le gros avantage de la mise en page avec Grid est le fait que le code HTML se retrouve très allégé. Fini les imbrications de  avec des classes .row et .col-xxx qui ajoutaient de la complexité à la structure HTML et qui étaient surtout utilisées pour un besoin purement graphique. Maintenant, tout se positionne presque par magie (ou grâce au talent de l'intégrateur). En plus de cela, l'autre avantage est que les Grid Items peuvent s'affranchir de leur ordre d'apparition dans la page HTML.

Généralement, plus le code HTML est léger, plus facile est le Responsive Design. Changer de mise en page ne représente aucune difficulté car cela se fait en modifiant les propriétés grid-template-columns, grid-template-rows et grid-template-areas.

Supposons que nous souhaitons partir sur 1 colonne et 6 lignes pour le format mobile

Trouver la solution à l'aide du media query suivant :
@media screen and (min-width: 768px) {}
pour le format mobile pour obtenir l'interface avec la disposition suivante des éléments :

En redéfinissant le **Grid Container**




Evidemment, les possibilités de Grid ne s'arrêtent pas à notre exemple, loin de là. Tout comme Flexbox, Grid permet très facilement de gérer les alignements verticaux et horizontaux au niveau du Container et des Items avec justify-* et align-* par exemple….

Voir la webographie ci-dessous pour aller plus loin...

Pour ceux qui veulent aller plus loin…

## WEBOGRAPHIE

https://css-tricks.com/snippets/css/complete-guide-grid/

https://gridbyexample.com/examples/

https://grid.layoutit.com/

https://cssgridgarden.com/