UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Frameworks CSS

Un framework, pourquoi ?

- Pour ne pas réinventer la roue à chaque fois
- Pour bénéficier de classes implémentant déjà les bonnes pratiques, notamment le responsive design
- Apporte des idées de styles qu'on saurait coder mais auxquelles on n'aurait pas forcément pensé

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Bootstrap

Bootstrap

- Créé par Twitter en 2011
- Framework CSS le plus populaire, et un des projets les plus populaires de GitHub
- Bootstrap 5 depuis 2021 (attention aux recherches Google qui mènent vers des anciennes versions de la doc)

Installation

Bootstap peut être utilisé avec un CDN en insérant ces balises dans le `head`

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/css/bootstrap.min.css"
    rel="stylesheet"
    integrity="sha384-sRIl4kxILFvY47J16cr9ZwB07vP4J8+LH7qKQnuqkuIAvNWLzeN8tE5YBujZqJLB"
    crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.8/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-FKyoEForCGlyvwx9Hj09JcYn3nv7wiPVlz7YYwJrWVcXK/BmnVDxM+D2scQbITxI"
    crossorigin="anonymous"></script>
```

Installation

Il peut également être installé avec npm

```
npm install bootstrap
```

Dans le cas d'un projet React, on ajoute en plus cet import à la racine de notre projet

```
import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
```

Les bases

Une fois Bootstrap installé, on va simplement ajouter à nos éléments HTML le nom des **classes** que l'on veut utiliser pour les styliser

```
<main class="container">
    <div class="row">
        <p class="col">Colonne 1</p>
        <p class="col">Colonne 2</p>
        <p class="col">Colonne 3</p>
    </div>
</main>
```

Les breakpoints

Bootstrap gère le responsive design avec des **breakpoints**

Breakpoint	Identifiant	Largeur
Extra small		< 576px
Small	`sm`	≥ 576px
Medium	`md`	≥ 768px
Large	`lg`	≥ 992px
Extra-large	`xl`	≥1200px
Extra-extra-large	`xxl`	≥1400px

À chaque breakpoint est associée une **media-query**

Mobile-first

- Bonne pratique : penser son design d'abord pour le mobile
- Par défaut, tout style est appliqué à la taille d'écran minimum, et jusqu'à ce qu'un breakpoint soit rencontré
- Les breakpoints peuvent être appliqués sur beaucoup de classes

```
<div class="text-center text-md-start">
```

Container

- Une classe englobante, qui est mise sur le main pour contenir tout le layout, mais qui peut également être utilisée dans d'autres cas
- De base, le `container` crée des marges de chaque côté, mais celles-ci peuvent être réduites via des breakpoints

```
<main class="container">
<!-- Marges tout le temps -->
<main class="container-md">
<!-- Marge réduite si l'écran est petit ou très petit, sinon marges normales -->
<main class="container-fluid">
<!-- Marges réduites dès le très petit -->
```

Le grid system

- Le layout se base globalement sur un système de grille, qui malgré son nom utilise des flexbox
- Elle se base sur un `row` qui par divise par défaut l'espace en 12 `col` qui vont automatiquement ajuster leur taille
- Une grille doit être placé dans un `container`

```
<div class="row">
    <p class="col">Element 1</p>
    <p class="col">Element 2</p>
    <p class="col">Element 3</p>
</div>
```

Le grid system

On peut également indiquer manuellement le nombre d'unités que l'on veut pour une `col`

```
<div class="row">
    <p class="col">Element 1</p>
    <p class="col-8">Element 2</p>
    <p class="col">Element 3</p>
</div>
```

Le grid system

Une `row` peut être définie avec un nombre de colonnes précise (de 1 à 6). Dans ce cas, chaque colonne prend uniquement une unité de place

```
<div class="row row-cols-2">
    <p class="col">Element 1</p>
    <p class="col">Element 2</p>
    <p class="col">Element 3</p>
</div>
```

Le grid system

Une grille peut en contenir une autre

```
<div class="row">
    <p class="col">Ligne 1 Colonne 1</p>
    <div class="col">
        <p>Ligne 1 Colonne 2</p>
        <div class="row">
            <p class="col">Ligne 2 Colonne 1 </p>
            <p class="col">Ligne 2 Colonne 2 </p>
        </div>
    </div>
    <p class="col">Ligne 1 Colonne 3</p>
</div>
```

L'espacement

- Plusieurs classes : `m` (margin), `p` (padding) et `g` (gutter)
- Valeurs allant de 0 à 5 sur chaque classe
- Possibilité d'appliquer à seulement un axe (x/y) ou une direction (s pour start, e pour end, t pour top et b pour bottom)

```
<div class="row p-3">
    <p class="col mx-2">Element 1</p>
    <p class="col mx-2">Element 2</p>
    <p class="col mx-2">Element 3</p>
</div>
```

L'espacement

On peut également utiliser `gap` sur une flexbox ou une grid pour espacer tous ses enfants
Ainsi le comportement de la slide précédente est proche de celui-ci

```
<div class="row p-3 gap-2">
    <p class="col">Element 1</p>
    <p class="col">Element 2</p>
    <p class="col">Element 3</p>
</div>
```

Attention, la valeur d'espacement n'est pas la même

Les gutters

Les gutters s'appliquent sur la ligne dans le grid system. Elles sont une sorte de padding compensé par du margin négatif pour tous les éléments englobés

```
<div class="row g-5">
    <p class="col">Très long texte inséré dans l'élément 1</p>
    <p class="col">Très long texte inséré dans l'élément 2</p>
    <p class="col">Très long texte inséré dans l'élément 3</p>
</div>
```

Flexbox

Pour créer une Flexbox, on utilise `d-flex` (d = display). On a ensuite nos propriétés habituelles:

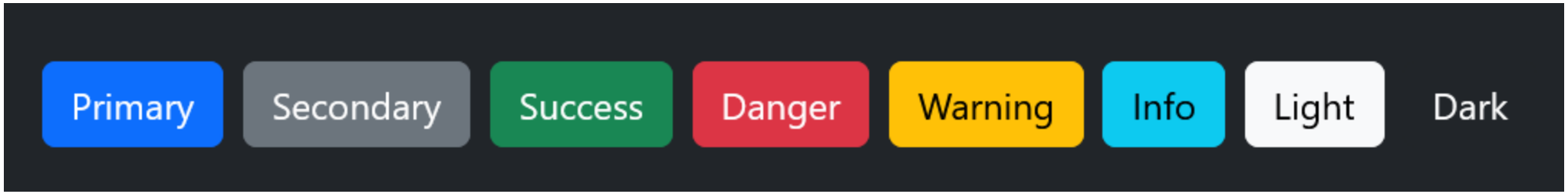
- `justify-content`
- `align-content`
- `align-items`
- `flex-wrap`
- etc

Flexbox

```
<div class="d-flex justify-content-around p-3">
    <p>Element 1</p>
    <p>Element 2</p>
    <p>Element 3</p>
</div>
```

Les couleurs

Bootstrap définit 8 couleurs personnalisées, utilisables facilement pour colorer du texte, un fond, des bordures...



Les couleurs

On peut l'ajouter à `text` pour colorer le texte, à `bg` pour colorer le fond, à `text-bg` pour colorer le fond également mais que Bootstrap choisisse automatiquement une couleur de texte contrasté, ou à `border` pour colorer une bordure par exemple

```
<div class="text-primary">Texte bleu</div>
<div class="bg-primary">Fond bleu</div>
<div class="text-bg-primary">Fond bleu et texte adapté</div>
<div class="border border-primary">Bordure bleue</div>
```

Taille des éléments

- Pour définir la taille d'un élément, on peut se référer au pourcentage du parent avec `w` et `h`, de 25 en 25
- On peut également utiliser `vw` et `vh` pour le pourcentage de l'écran, toujours de 25 en 25

```
<div class="bg-primary w-25">-</div>
```

Le texte

- Les éléments `h1`, `h2` etc ont une taille par défaut dans Bootstrap
- On peut également appliquer la classe `fs` (font-size) avec des valeurs de 1 (plus grand) à 5 (plus petit)
- Pour du plus grand texte, on a aussi la classe `display` marchant de la même façon

```
<p class="display-1">Très grand texte</p>
<p>Texte normal</p>
<p class="fs-5">Texte légèrement grand</p>
```

Le texte

- On peut facilement aligner du texte, le mettre en **gras**, en *italique* ou même en MAJUSCULE

```
<p class="text-center">Je suis centré</p>
<p class="text-end">Je suis à la fin</p>
<p class="fw-bold">Je suis gras</p>
<p class="text-uppercase">Je suis en majuscule</p>
```

Généralités

- Beaucoup de propriétés sont transparentes : `text-center` pour aligner le texte au centre, `border` pour ajouter une bordure, `border-[color]` pour la colorer, `text-[color]` pour colorer du texte...
- Attention, il faut souvent d'abord entrer une classe globale puis une classe précisant le comportement (`row` puis `row-cols-[nombre]` pour limiter le nombre de colonnes, ou `border` puis `border-danger` pour déclarer une bordure rouge par exemple)

Les boutons

La classe `btn` permet de styliser un élément comme un bouton. On peut rajouter une couleur en ajoutant une seconde classe `btn-[color]` , ou limiter la couleur au cadre et au texte avec `btn-outline-[color]`

```
<button class="btn">Bouton basique</button>
<a class="btn btn-primary">Bouton bleu</a>
<input type="submit" class="btn btn-outline-success"
    value= "Bouton à cadre vert">
```

Les tableaux

La classe `table` permet de styliser un tableau

```
<table class="table">
<!-- ... -->
</table>
```

On peut ajouter plusieurs options comme créer un tableau à bandes avec `table-striped`, rajouter une couleur (possible également sur une seule ligne) avec `table-[color]` ou rajouter des bords avec `table-bordered`

Les listes

On peut créer une liste avec `list-group` puis ajouter `list-group-item` sur chaque élément.

```
<ul class="list-group">
    <li class="list-group-item">Element 1</li>
    <li class="list-group-item">Element 2</li>
    <li class="list-group-item">Element 3</li>
</ul>
```

On peut ajouter une numérotation avec `list-group-number`, l'afficher en horizontal avec `list-group-horizontal` ou encore supprimer les bords avec `list-group-flush`

Les formulaires

Dans un form, on va typer d'un côté les label avec `form-label` et de l'autre les input avec `form-control`. On peut également ajouter du texte avec `form-text`.

```
<form class="d-flex flex-column align-items-center border border-dark rounded w-25 mx-auto my-4 p-4">
    <div class="mb-2">
        <label for="email" class="form-label">Adresse e-mail</label>
        <input type="email" class="form-control" name="email">
    </div>
    <div class="mb-2">
        <label for="password" class="form-label">Mot de passe</label>
        <input type="password" class="form-control" name="password">
    </div>
    <input type="submit" value="Valider" class="btn btn-outline-dark mt-3">
</form>
```

Les formulaires

On peut intégrer le label à l'input avec `form-floating`. Dans ce cas, on doit obligatoirement mettre un placeholder à notre input, et le déclarer avant son label

```
<div class="mb-2 form-floating">
    <input type="email" class="form-control" name="email" placeholder="">
    <label for="email" class="form-label">Adresse e-mail</label>
</div>
```

Les formulaires

Les possibilités de formulaires sont foisonnantes, et souvent on ajoutera simplement le type d'input particulier au suffix `form` (`form-check`, `form-select`, `form-range`...)

Les cards

Les cards sont des conteneurs supportant header, footer, image, titre et corps... Ils représente en quelque sorte l'affichage d'un objet avec toutes ses informations.

```
<div class="card w-25">
    <img src="fraise.jpeg" alt="Image de fraises" class="card-img-top">
    <h3 class="card-header">La fraise</h3>
    <div class="card-body">
        <h5 class="card-title">Le meilleur des fruits</h5>
        <p class="card-text">Sa chair délicieuse peut être consommée nature tout comme dans quantité de recettes
        </p>
    </div>
    <div class="card-footer">À consommer sans modération</div>
</div>
```

Les modals

- Une modale est simplement une fenêtre apparaissant au-dessus du contenu de la page web et prenant le contrôle jusqu'à ce qu'elle soit fermée.
- C'est un composant Bootstrap complexe utilisant Javascript. Elle nécessite donc l'inclusion du CDN approprié.
- Il est recommandé de placer le code de la modale au début de notre `body`.
- Le contenu d'un élément englobant de classe `modal` ne s'affichera qu'au clic d'un bouton possédant les attribut `data-bs-toggle="modal"` et `data-bs-target="#exemple"` où `exemple` correspond à l'id donné à l'élément de classe `modal`

Les modals

```
<div class="modal" id="exemple" tabindex="-1">
        <div class="modal-dialog">
            <div class="modal-content">
                <div class="modal-header">
                    <h1 class="modal-title">Titre</h1>
                    <button type="button" class="btn-close" data-bs-dismiss="modal"></button>
                </div>
                <div class="modal-body">Contenu</div>
                <div class="modal-footer">
                    <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Annuler</button>
                    <button type="button" class="btn btn-primary">Accepter</button>
                </div>
            </div>
        </div>
    </div>
    <button data-bs-toggle="modal" data-bs-target="#exemple">Afficher modale</button>
```

Les navbars

Bootstrap met à disposition une navbar responsive, qui va s'étendre verticalement sur un petit écran et s'afficher horizontalement sur un grand écran

C'est un autre exemple de composant complexe avec beaucoup de classes, d'éléments et de propriétés différentes, nécessitant l'import du Javascript de Bootstrap

Les navbars

```
<nav class="navbar navbar-expand-lg border rounded">
    <div class="container-fluid">
        <a class="navbar-brand" href="#">
            <img src="ecole-it.png" alt="icone de l'école" height="35">
        </a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav">
                <li class="nav-item">
                    <a class="nav-link active" aria-current="page" href="#">Home</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="#">Features</a>
                </li>
            </ul>
        </div>
    </div>
</nav>
```

Les accordions

L'accordion est un autre composant complexe utilisant encore le principe de collapse, utilisant du JS.

```
<div class="accordion" id="payments" data-bs-theme="dark">
    <div class="accordion-item">
        <h2 class="accordion-header">
            <button class="accordion-button" type="button" data-bs-toggle="collapse" data-bs-target="#collapseOne">
                Carte de crédit
            </button>
        </h2>
        <div id="collapseOne" class="accordion-collapse collapse show" data-bs-parent="#payments">
            <div class="accordion-body">
                Vous devrez entrer votre numéro de carte de crédit et obtenir l'autorisation de votre banque
            </div>
        </div>
    </div>
    <div class="accordion-item">
        <h2 class="accordion-header">
            <button class="accordion-button collapsed" type="button" data-bs-toggle="collapse" data-bs-target="#collapseTwo">
                Paypall
            </button>
        </h2>
        <div id="collapseTwo" class="accordion-collapse collapse" data-bs-parent="#payments">
            <div class="accordion-body">
                Vous devrez valider la transaction depuis votre compte Paypall
            </div>
        </div>
    </div>
</div>
```

Les offcanvas

Le offcanvas est encore un composant complexe. Il ressemble un peu à un intermédiaire entre une modale et une navbar. C'est un élément apparaissant sur le côté, à l'appui d'un bouton, et qui surplombe le contenu de la page

```
<!-- Permet d'ouvrir l'offcanvas -->
<button class="btn btn-primary" type="button" data-bs-toggle="offcanvas" data-bs-target="#offcanvasExample">
    Afficher offcanvas
</button>
<!-- Offcanvas (masqué par défuat) -->
<div class="offcanvas offcanvas-start" tabindex="-1" id="offcanvasExample">
    <div class="offcanvas-header">
        <h5 class="offcanvas-title" id="offcanvasExampleLabel">Offcanvas</h5>
        <button type="button" class="btn-close" data-bs-dismiss="offcanvas"></button>
    </div>
    <div class="offcanvas-body">
        <div>
            Ceci est un contenu intéressant mais pas primordial, puisqu'il ne s'affiche que si on le lui demande.
        </div>
        <ul class="list-group">
            <li class="list-group-item">Element</li>
            <li class="list-group-item">Un autre élémént</li>
            <li class="list-group-item">Encore un élément</li>
        </ul>
    </div>
</div>
```

Les icônes

Bootstrap fournit également des icônes utilisables en plaçant les classes appropriées dans un élément `i`

```
<i class="bi bi-trash3"></i>
```

On peut appliquer les autres classes sur nos icônes, notamment les couleurs

```
<i class="bi bi-trash3 text-primary"></i>
```

Les icônes

Utiliser les icônes nécessite d'utiliser un autre CDN

```
<link rel="stylesheet" href=
"https://cdn.jsdelivr.net/npm/bootstrap-icons@1.13.1/font/bootstrap-icons.min.css">
```

Ou d'installer un autre package

```
npm install bootstrap-icons
```

Personnalisation

- Les options offertes par Bootstrap sont pratiques mais limitées
- Si on veut modifier le comportement des classes Bootstrap dans nos projets, par exemple modifier les variables de couleur (`primary`, `secondary`, `success` etc) on doit utiliser **Sass**

Sass

Sass (**S**yntaxically **a**wesome **s**tyle**s**heets) est un préprocesseur CSS : il ajoute des fonctionnalités n'existant pas en CSS (mixins, boucles, tests conditionnels) et est ensuite compilé en du CSS.
Pour l'utiliser, il est nécessaire de l'installer. Et on ne peut plus utiliser le CDN de Bootstrap, on doit utiliser le package node ou avoir les fichiers non-compilés en local

```
npm i -g sass
npm i bootstrap
```

Sass

On crée ensuite un fichier .scss dans lequel on va redéfinir nos variables, puis importer le Sass de Bootstrap

```
$primary:#000000;

@import "./node_modules/bootstrap/scss/bootstrap";
```

Puis, on doit compiler notre fichier, et insérer un `link` dans notre HTML (et supprimer celui du CDN)

Sass

Pour compiler, on utilise cette commande

```
sass nom.scss nom.css
```

Comme pour Typescript ou Node, on peut utiliser `--watch` pour recompiler automatiquement à chaque modification

```
sass --watch nom.scss nom.css
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Tailwind

Tailwind

Tailwind a une philosophie radicalement différente de Bootstrap : il ne fournit quasiment aucun composant ou style pré-définie, mais va plutot nous laisser personnaliser nos éléments à outrance.

C'est l'approche utility-first : chaque classe n'a qu'un seul but, contrairement à Bootstrap qui est dit component-based

- Avantage : CSS minimal et design précis
- Inconvénient : HTML très verbeux et apprentissage plus difficile

Installation

Tout comme Bootstrap, Tailwind peut soit être utilisé avec un CDN

```
<script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
```

Ou installé via npm

```
npm install tailwindcss @tailwindcss/cli
```

Installation

On importe ensuite Tailwind dans un fichier CSS racine

```
@import "tailwindcss";
```

Puis on build ce fichier, et on le référence dans notre HTML

```
npx @tailwindcss/cli -i ./input.css -o ./output.css --watch
```

Généralités

Tailwind partage de nombreux noms de classes avec Bootstrap:

- `m` pour margin (`ms`, `mx`, `mt` etc) et `p` pour padding
- `text-center` ou `text-right` pour positionner le texte
- `bg-` ou `text-` pour indiquer une couleur de fond ou de texte
- `w` pour width et `h` pour height pour les tailles
- `border` pour créer une bordure
- etc

Généralités

La différence avec Bootstrap, c'est les possibilités offertes

- Pas de limite sur les valeurs de margin, de padding ou de taille
- Des couleur très détaillés (sur le modèle `nom-intensité`)
- De nombreuses valeurs de taille de texte
- Plusieurs types d'arrondis pour les bordures
- etc

Espacement

- Le margin,le padding et le gap n'ont pas de limite de valeur dans Tailwind. On peut donc positionner nos éléments bien plus finement.
- Contrairement à Bootstrap, les valeurs négatives de margin sont activées par défaut, et utilisables simplement en rajoutant un - devant la classe

```
<p class="-ms-10">Hello world</p>
<!-- Marge négative -->
```

Espacement

- On peut utiliser l'unité pixel plutôt que l'unité de Tailwind avec

```
<p class="ms-[10px]">Hello world</p>
```

- On peut également utiliser les propriétés `ml` pour margin-left et `mr` pour margin-right
- Enfin, on peut définir un espacement sur axe commun aux éléments englobés avec `space`

```
<div class="flex justify-center space-x-20">
    <p>Du texte</p>
    <p>Du texte</p>
    <p>Du texte</p>
</div>
```

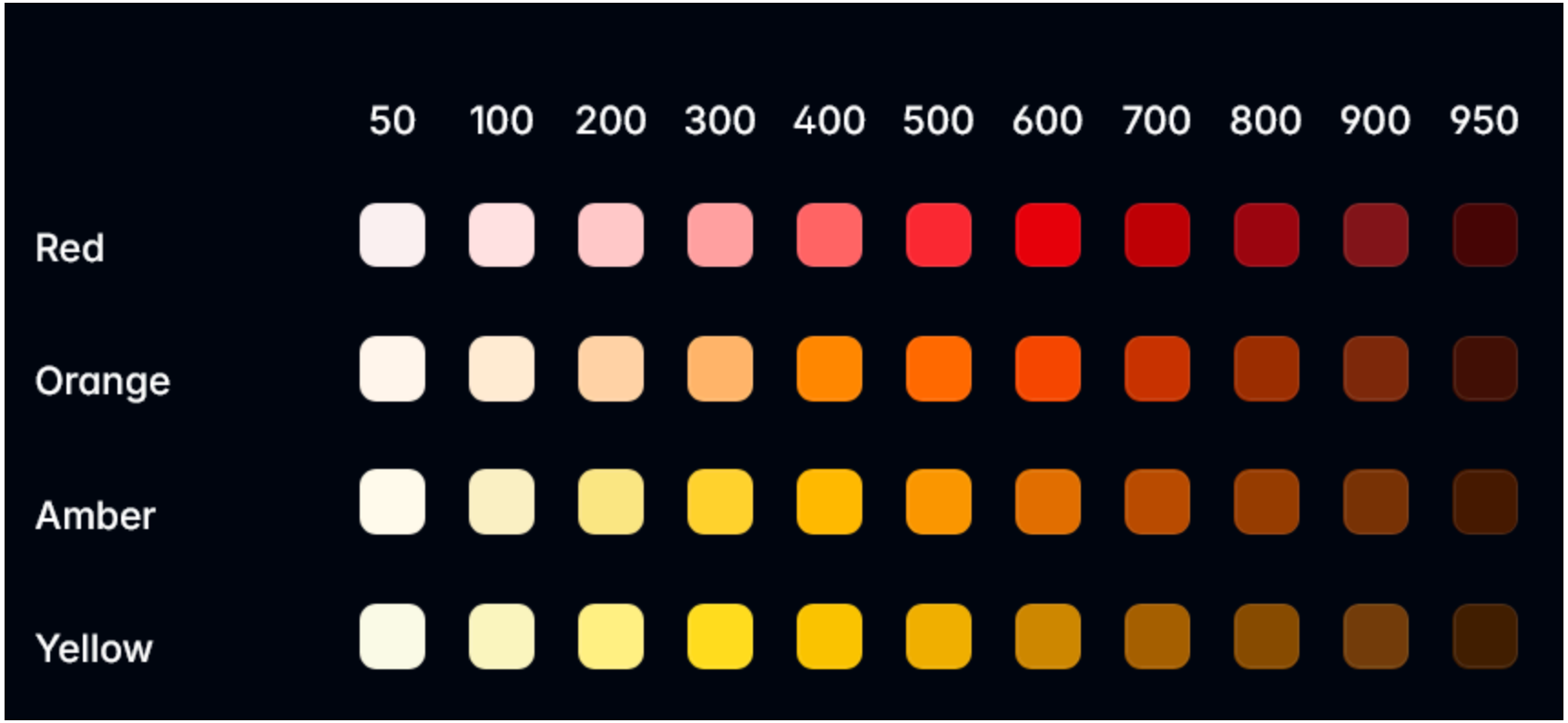
Les tailles

On peut régler la taille avec `w-` et `h-` en passant des valeurs absolus. On peut aussi passer `w-full` ou une fraction `w-1/2`. Il y a également un certain nombre de tailles prédéfinies, de `w-3xs` jusqu'à `w-7xl`. Enfin, on peut utiliser `size-` pour donner la même valeur à la hauteur et à la largeur.

```
<p class="mx-auto w-2/3">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam velit arcu,
    sagittis vitae velit id, finibus scelerisque neque. Vivamus vulputate scelerisque sodales.
</p>
```

Couleurs

Tailwind propose une vingtaine de couleurs par défaut



Couleurs

Les couleurs sont applicables sur le fond, le texte, les bordures, les ombres, les décorations...

```
<p class="underline decoration-red-500">Hello world</p>
```

L'opacité se contrôle en ajoutant `/` puis la valeur d'opacité désirée

```
<p class="underline decoration-red-500/20">Hello world</p>
```

On peut également utiliser `white` et `black` comme valeur de couleur

Typographie

- La classe `text-` permet à la fois de colorer, aligner et contrôler la taille du texte

```
<p class="text-center text-red-500 text-xl">Hello world</p>
```

- Il y a 13 valeurs de taille de texte par défaut, de `xs` à `9xl`
- Comme pour l'espacement, on peut aussi indiquer une valeur en pixel avec `text-[2px]`

Typographie

Quelques autres classes :

- `italic`
- `font-bold`
- `uppercase`
- `underline`
- `font-serif`
- etc

Variants

Un variant est une condition à l'application d'une classe. Cela peut être un `hover`, un `focus`, la sélection du mode `dark`, mais aussi l'attribut `required`, `read-only` ou encore `odd` et `even` quand vous voulez cibler un enfant pair/impair

```
<p class="rounded-md px-5 py-2
        bg-blue-500 text-white
        cursor-pointer hover:bg-blue-600">
    Hello world
    </p>
```

Variants

Le responsive design se fait aussi via les variants. Les breakpoints sont assez proches de ceux de Bootstrap

Breakpoint	Identifiant	Largeur
Extra-small		< 640px
Small	`sm`	≥ 640px
Medium	`md`	≥ 768px
Large	`lg`	≥ 1024px
Extra-large	`xl`	≥1280px
Extra-extra-large	`2xl`	≥1536px

Variants

```
<p class="
    text-xs
    sm:text-base
    md:text-lg
    lg:text-2xl
    xl:text-4xl
    2xl:text-6xl
">Hello world</p>
```

Variants

On peut aussi spécifier un range de breakpoints avec `:max-`. Ainsi on veut avoir un comportement spécifique seulement entre deux breakpoints

```
<p class="md:max-lg:underline">Hello world</p>
<!-- Texte souligné pour un écran moyen ou grand -->
 <!-- Texte nus pour les écrnas plus petits ou plus grands -->
```

Variants

Comme les autres variants, les breakpoints sont utilisables sur **toutes** les classes de Tailwind, toujours avec la même syntaxe. Ils sont également combinables si on veut des conditions plus précises

```
<p class="md:max-lg:hover:underline">Hello world</p>
<!-- Texte souligné seulement quand l'écran est moyen ou grand
 ET que la souris est dessus -->
```

Personnalisation

On peut personnaliser Tailwind en créant ses propres variables de thème, par exemple de couleur, en respectant la syntaxe Tailwind et les plaçant dans le fichier CSS racine

```
@import "tailwindcss";
@theme {
    --color-mint-500: oklch(0.72 0.11 178);
    --text-godzilla: 200px;
    }
```

Personnalisation

Les variables de thèmes sont très utiles pour par exemple définir des polices ou des couleurs spécifiques à un projet, mais elles peuvent aussi permettre de créer de nouveaux breakpoints personnalisés

```
@theme {
    --breakpoint-xxs: 400px;
}
```

Ce breakpoint sera ensuite utilisable comme n'importe quel autre breakpoint

```
<p class="text-md xxs:text-xl">Hello world</p>
```

Personnalisation

On peut aussi appliquer des styles par défaut à certains éléments dans notre projet avec `@layer base`

```
@layer base {
    h1 {
        color: green;
    }
    p {
        color:blue;
    }
}
```

Personnalisation

`@layer` permet aussi de se créer des classes totalement personnalisée, sur lesquelles on pourra toujours appliquer des utilitaires de Tailwind qui écraseront les valeurs par défaut

```
@layer components {
    .card {
        background-color: var(--color-white);
        border-radius: var(--radius-lg);
        padding: --spacing(6);
        box-shadow: var(--shadow-xl);
        }
    }
```

Personnalisatoin

`@apply` permet d'appliquer dans un style personnalisé autant de classes utilitaires qu'on veut

```
.button{
     @apply px-4 py-2 rounded bg-blue-500 cursor-pointer text-white
}
```

Personnalisation

On peut également utiliser la personnalisation pour créer un thème sombre

On doit ajouter cette section dans notre style CSS

```
@custom-variant dark (&:where(.dark, .dark *));
```

Qui permet de changer le comportment du variant `dark`. Par défaut : détecte les préférences de l'utilisateur. Avec ce code : applique le variant quand un parent a la class `dark`

Personnalisation

On rajoute la classe `dark` à notre élément HTMl si on le veut par défaut puis

```
<body class="bg-neutral-200 text-neutral-800
    dark:bg-neutral-800  dark:text-neutral-200">
    <h1 class="text-3xl mb-4 text-center">Bienvenue</h1>
    <button class="cursor-pointer border rounded px-3 py-2 block mx-auto
        bg-neutral-800 hover:bg-neutral-700 text-neutral-200
        dark:bg-neutral-200 hover:dark:bg-neutral-300 dark:text-neutral-800"
        onclick="document.documentElement.classList.toggle('dark')">Switch</button>
</body>
```

On a maintenant un bouton permettant de switcher entre mode sombre et clair

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Purge CSS

Purge CSS

Purge CSS est un outil permettant de supprimer le CSS non-utilisé d'un projet afin d'avoir à la compilation des fichiers CSS plus petits.

PurgeCSS peut s'utiliser avec PostCSS ou Next.js, mais également de manière indépendante. Elle s'installe via npm.

```
npm i -g purgecss
```

PurgeCSS

On va ensuite utiliser cette commande pour produire un fichier purgé

```
purgecss --css css-a-purger.css --content html-a-analyser.html  --output chemin-fichier-final
```

Tailwind met nativement en oeuvre des mécanismes de PurgeCSS

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Bulma

Présentation et installation

Bulma est, comme Bootstrap, un framework component based, mais il est plus léger et a des possibilités différentes. Contrairement à Bootstrap, il est totalement exempt de JS.

Il est lui aussi mobile-first, et il peut lui aussi être utilisé via CDN

```
<link rel="stylesheet"
  href="https://cdn.jsdelivr.net/npm/bulma@1.0.4/css/bulma.min.css">
```

Ou installé avec npm

```
npm install bulma
```

Exemples d'applications

- Signal
- JustWatch
- Buefy (utilisation avec Vue.js)

Les colonnes

Bulma a un système de `columns` fonctionnant un peu comme le grid system de Bootstrap. Un conteneur `columns` redimensionne et répartit automatiquement les éléments `column` qu'il contient

```
<div class="columns">
    <p class="column button is-link mx-3">Du texte</p>
    <p class="column button is-link mx-3">Du texte</p>
    <p class="column button is-link mx-3">Du texte</p>
</div>
```

La grosse différence, c'est que sur mobile, elles s'empilent verticalement par défaut

Les colonnes

Un `column` a un gap par défaut, qu'on peut personnaliser avec `is-`, mais comme Bootstrap, c'est un système de padding et pas de margin

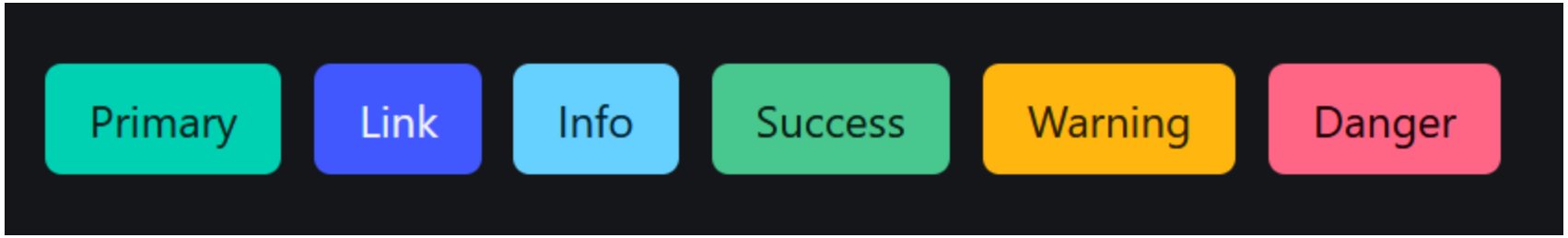
```
<div class="columns is-8">
    <p class="column button is-link">Du texte</p>
    <p class="column button is-link">Du texte</p>
    <p class="column button is-link">Du texte</p>
    <p class="column button is-link">Du texte</p>
    <p class="column button is-link">Du texte</p>
</div>
```

Les colonnes

On peut également modifier la taille d'une colonne avec le mot-clé `is-`. Les valeurs vont de 1 à 12, mais peuvent également être `is-full`, `is-half` ainsi que des multiples de cinquième, quarts et tiers

```
<div class="columns">
    <p class="column button is-primary mx-3 is-one-fifth">Du texte</p>
    <p class="column button is-primary mx-3 is-one-quarter">Du texte</p>
    <p class="column button is-primary mx-3 is-one-third">Du texte</p>
    <p class="column button is-primary mx-3 is-1">Du texte</p>
</div>
```

Les couleurs



Le système de couleurs est également assez proche de celui de Bootstrap. On peut également utiliser `white`, `black`, `light` et `dark`, et rajouter ces deux derniers sur n'import quelle couleur.

Les couleurs

On utilise `has-background-` pour colorer un arrière-plan, `has-text-` pour colorer du texte, et `is-` pour un bouton.

```
<div class="has-background-info-dark has-text-light">Hello world</div>
<button class="button is-link">Valider</button>
```

Responsive design

Bulma a également un système de breakpoints, mais utilisable pour moins de choses que Bootstrap ou Tailwind.

Breakpoint	Largeur
`mobile` ou rien	< 769px
`tablet`	≥ 769px
`desktop`	≥ 1024px
`widescreen`	≥1216px
`fullhd`	≥1408px

Responsive design

Il peut s'utiliser sur la taille, l'orientation et le gap des colonnes, la taille du texte et l'orientation du texte, et la propriété display.

```
<div class="columns">
    <p class="column button is-primary mx-3 is-one-fifth is-three-quarters-widescreen">
        Du texte
    </p>
</div>
<p class="has-text-centered has-text-left-tablet has-text-right-widescreen">Du texte</p>
<p class="is-size-5 is-size-4-tablet is-size-3-desktop is-size-2-widescreen is-size-1-fullhd">
    Hello world
    </p>
```

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Foundation

Foundation

Foundation est un framework plus lourd et plus complexe, créé par l'agence de design ZURB, pensé pour être très personnalisé et offrant moins de styles par défaut que Bulma ou Bootstrap. Il offre aussi plus de support pour l'accessibilité, notamment une partie de la documentation dédiée à intégrer ARIA. Il est censé être orienté projet entreprise

Installation

Foundation peut être utilisé via un CDN

```
<link rel="stylesheet" href=
"https://cdn.jsdelivr.net/npm/foundation-sites@6.9.0/dist/css/foundation.min.css"
crossorigin="anonymous">
<script src=
"https://cdn.jsdelivr.net/npm/foundation-sites@6.9.0/dist/js/foundation.min.js"
crossorigin="anonymous"></script>
```

Ou installé avec npm

```
npm install foundation-sites
```

Généralités

Comme en Bootstrap, de nombreux éléments sont stylisés par défaut : les titres, les listes, les paragraphes...

Il n'existe pas de classe utilitaire permettant directement de colorer du texte ou un fond, mais il existe plusieurs couleurs pouvant être précisées sur certains composants : `button`, `badge`.

Les classes pour la typographie et l'espacement sont majoritairement désactivées par défaut

XY Grid

Foundation possède un système de grille de 12 éléments un peu différent de ceux de Bootstrap ou Bulma.
Il se fait en déclarant explicitement l'axe de notre grille. Par défaut, chaque élément prend toute la largeur possible.
On peut également définir des marges horizontales ou verticales. Pour personnaliser leur taille, on sera obligé d'entrer dans le Sass.

```
<div class="grid-x grid-margin-x">
    <div class="cell">Cellule prenant toute la largeur</div>
    <div class="cell small-3">Cellule prenant 1/4 de la largeur</div>
    <div class="cell small-3">Cellule prenant 1/4 de la largeur</div>
</div>
```

XY Grid

On peut définir la largeur de nos colonnes en lui donnant une valeur de 1 à 12, mais seulement en suivant un breakpoint.

Breakpoint	Largeur
`small`	< 640px
`medium`	≥ 640px
`large`	≥ 1024px

Ces breakpoints sont utilisables sur les colonnes, la taille et l'alignement du texte, la visibilité ou l'espacement.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

CSS-in-JS

CSS-in-JS

Comme son nom l'indique, le CSS-in-JS est une technique consistant à écrire du style en Javascript. Plus précisement, il s'agit de définir le style d'un composant directement dans celui-ci.

Cette méthode permet d'avoir le style le plus spécifique possible, et également d'utiliser les props des composants.

Elle ne dépend pas d'un framework en particulier, mais on peut noter que c'est la méthode de base en React Native.

CSS-in-JS

Exemple en React Native

```
const InputAndLabel = () => {
    return (
        <View>
            <Text style={styles.text}>{label}</Text>
            <TextInput
                placeholderTextColor={"#EBEBEB"}
                style={styles.input} />
        </View>
    )
};
```

```
const styles = StyleSheet.create({
    text: {
        marginLeft:15,
        marginBottom:15,
        color: "#0E0E0E"
    },
    input: {
        backgroundColor: "#707899",
        width: 220,
        borderRadius: 12,
        padding: 15,
        color : "#EBEBEB"
        }
});
```

Styled components

Styled Components est une bibliothèque JS implémentant ce pattern en React.

```
npm i styled-components
```

Après avoir défini un composant avec sa syntaxe, on l'appelle par une balise de son nom

```
const Element = styled.element`
    cssProperty: value;
    cssProperty: value;
`;

return (
    <><Element/></>
);
```

Emotion

Emotion est une autre bibliothèque permettant de faire du CSS-in-JS mais peut être implémenté dans n'importe quel framework front.

```
npm i @emotion/css
```

Pour la version React :

```
npm i @emotion/react
```

Emotion en React

La syntaxe est quasiment la même, la différence est qu'on doit placer cet import particulire tout en haut du fichier

```
/** @jsxImportSource @emotion/react */
```

Que la méthode utilisée est `css` et que cela ne crée pas un élément, mais simplement la valeur d'un attribut `css` qu'on ajoutera à notre élément.

```
const elementStyle = css`
    cssProperty: value;
    cssProperty: value;
`;

return (
    <><div css={elementStyle}></div></>
);
```

Comparatif

- Bootstrap : prototypes rapides ou deadline serrées, sites vitrines classiques, design basiques
- Tailwind : design stricts, applications modernes, projets longs
- CSS-in-JS : frameworks front comme React ou Vue
- Bulma et Foundation : cas particuliers

Comparatif - Taille et JS

Framework	Taille	JavaScript
Bootstrap	152 KB	Inclus
Tailwind	~10-30 KB	Non
Bulma	44 KB	Non
Foundation	90 KB	Inclus

Les tailles sont données après purge, en contexte de production (exemple : Tailwind a plutôt un poids de quelques Mb pendant le développement)

Précisions sur le SASS

Une **mixin** est un peu l'équivalent en SASS d'une fonction. Elle permet de contenir des éléments de style qui peuvent être réutilisés sans se répéter, mais peut également inclure des paramètres.

On en déclare avec `@mixin`, et on l'utilise avec `@include`

Les différents frameworks utilisent beaucoup des mixins pour leurs fonctionnalités

Précisions sur le SASS

```
@mixin theme($theme) {
    background: $theme;
    color: #FFFFFF;
}
.info {
    @include theme(blue);
}
.alert {
    @include theme(red);
}
```

Tendances actuelles

L'approche utility-first prend le pas sur le component-based. Le CSS-in-JS prend également de l'importance.

Frameworks à surveiller:

- UnoCSS
- Master CSS
- Vanilla Extract
- Open Props