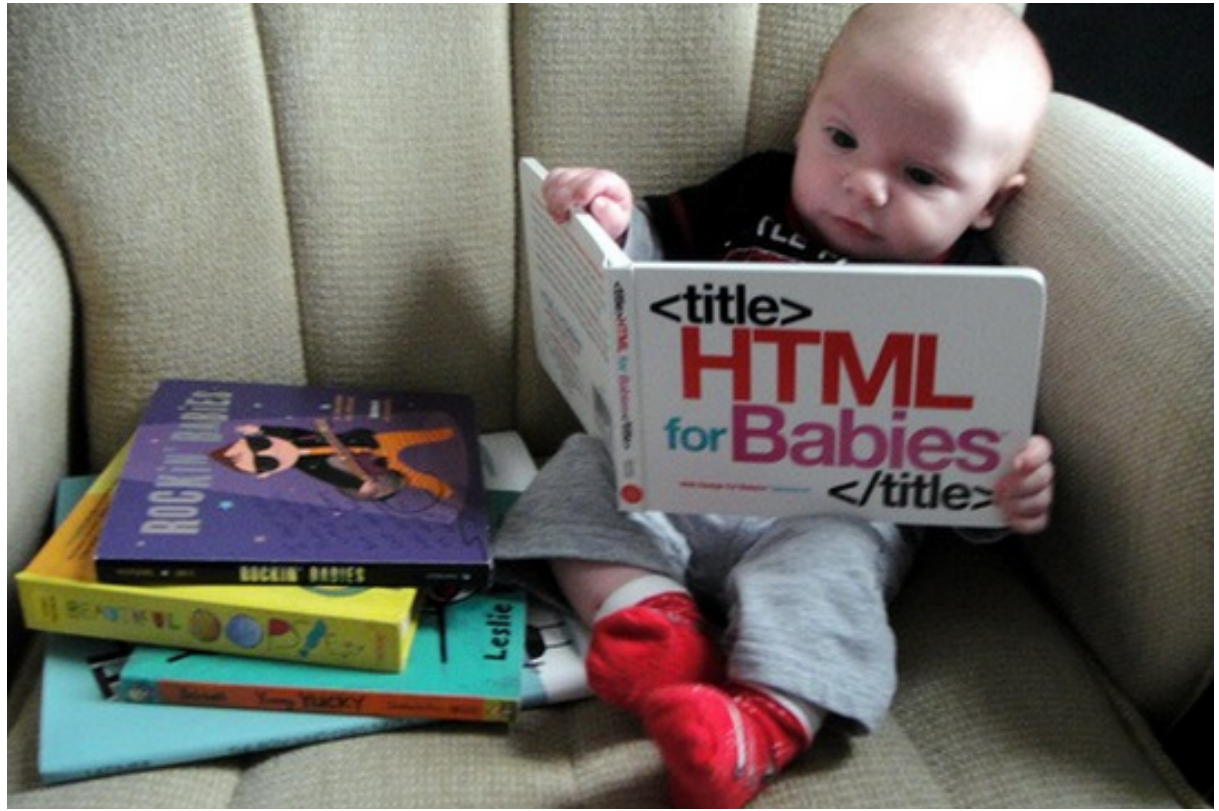
CSS

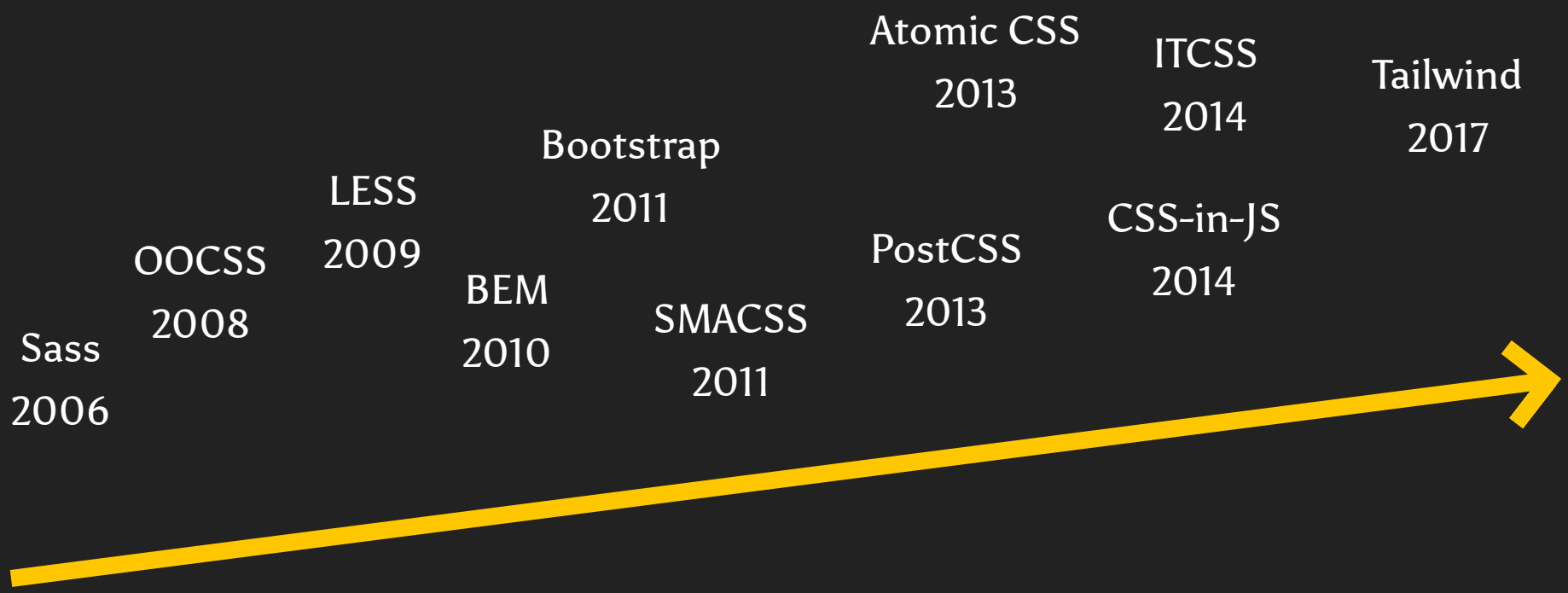
Méthodologies et conventions

conventions de nommage, BEM, Tailwind, etc.



Évolution des méthodologies CSS

un peu d'histoire

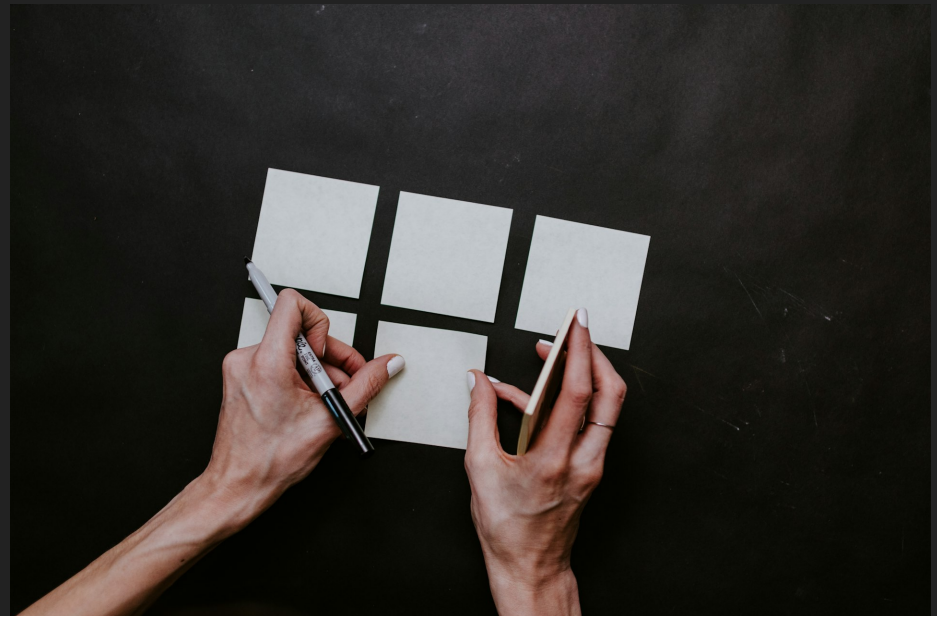


https://andreipfeiffer.dev/blog/2022/scalable-css-evolution

Au programme

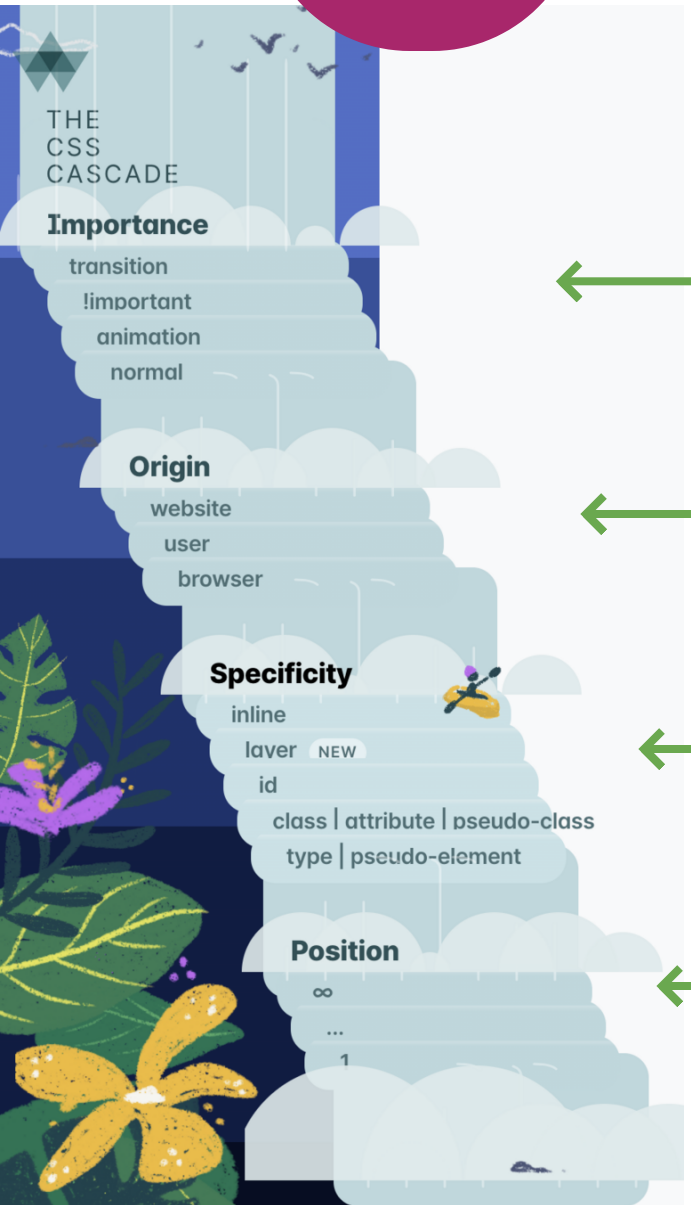
"un peu d'organisation, diantre !"

1. Les problématiques à résoudre
2. Appliquer une méthodologie (BEM, Atomique)
3. Du design à l'intégration



Les principales Problématiques à résoudre

1 La Cascade CSS



← !important, transitions, animations

← styles utilisateur, styles navigateur, styles du webdeveloppeur

← règles de spécificité (poids) des sélecteurs

← ordre d'affichage (le dernier écrase le précédent)

https://2019.wattenberger.com/blog/css-cascade

1 La Cascade CSS

```css
ul#navigation > .link { color: salmon }
ul#navigation li a { color: pink }
#navigation a.link { color: olive }
body ul#navigation li a { color: tomato }
body nav ul li a.link:first-child { color: chocolate }
```

Inconvénients majeurs :

1. Chaque type de sélecteur CSS a sa propre spécificité
2. Ajouter du poids pour écraser un sélecteur a un effet "boule de neige" pour la prochaine modification à opérer
3. Pourquoi ce ☠️ 🤬 😡 de bouton ne devient pas rose ?!?!

Solutions envisageables :

1. Connaître la cascade et la spécificité des sélecteurs
2. Employer des astuces modernes (@layer, :where, @scope)
3. Choisir des sélecteurs légers (.class et non #id)
4. C'est le principal job des méthodologies telles que BEM ou Atomique

2 Nommage des éléments

```
.img, .image, .blog-img, .img-blog, .blogImage, .thumbnail_img { ... }
.btn, .bouton, .button, .btn--primary, .close-btn, .btn-close { ... }
.wrapper, .container, .items-container, .inner-wrapper,
.wrapper__inner--alternate { ... }
.title-main, .post-title, .heading { ... }
```

Inconvénients majeurs :

1. Difficile de trouver des noms cohérents au fur et à mesure que le projet grossit
2. Risque de choisir un nom déjà existant et d'écraser une partie des styles

Solutions envisageables :

1. Avoir une convention de nommage (français/anglais, camelCase, BEM, etc.)
2. Préfixer les noms (.l-wrapper, .c-button)
3. Générer des noms dynamiquement (.abc23f4, .xyr421337) à l'aide d'un outil

3 Duplication des sélecteurs

```css
.wrapper { }
.wrapper h1, .wrapper h2, .wrapper p { }
.wrapper:hover, .wrapper:focus { }
.wrapper::before { }

@media (width > 640px) {
  .wrapper { }
  .wrapper::before { }
}
```

```scss
.wrapper {
  & h1,
  & h2,
  & p {}

  &:hover,
  &:focus {}

  &::before {}

  @media (width > 640px) {
    &::before {}
  }
}
```

Inconvénients majeurs :

1. Difficile de trouver, renommer, déplacer, supprimer les sélecteurs

Solutions envisageables :

1. Nesting Scss pour réduire les duplications
2. Nesting en CSS natif

… mais attention au nesting multiple !

4 Duplication des règles CSS

```css
.button-primary {
  display: inline-block;
  padding: 1rem 2rem;
  border: 1px solid hotpink;
  background: pink;
  cursor: pointer;
}
.button-secondary {
  display: inline-block;
  padding: 1rem 2rem;
  border: 1px solid chocolate;
  background: bisque;
  cursor: pointer;
}
```

Inconvénients majeurs :

1. Code inutilement répété
2. Maintenabilité complexe

```css
.button {
  display: inline-block;
  padding: 1rem 2rem;
  cursor: pointer;
}
.button-primary {
  border: 1px solid hotpink;
  background: pink;
}
.button-secondary {
  border: 1px solid chocolate;
  background: bisque;
}
```

Solutions envisageables :

1. Rassembler les styles identiques
2. Proposer des classes (variantes) complémentaires

5 Code mort

```html
<!-- lien à remplacer par un button -->
<a class="alternate special">
  Cliquez-moi !
</a>
```

```css
/* on préfère ne rien supprimer */
a.special {color: tomato;}
a.alternate {color: pink;}
```

Inconvénients majeurs :

1. Le code inutilisé s'accumule au fur et à mesure que le projet grossit
2. Il pollue les feuilles de styles et les rend difficilement compréhensibles

Solutions envisageables :

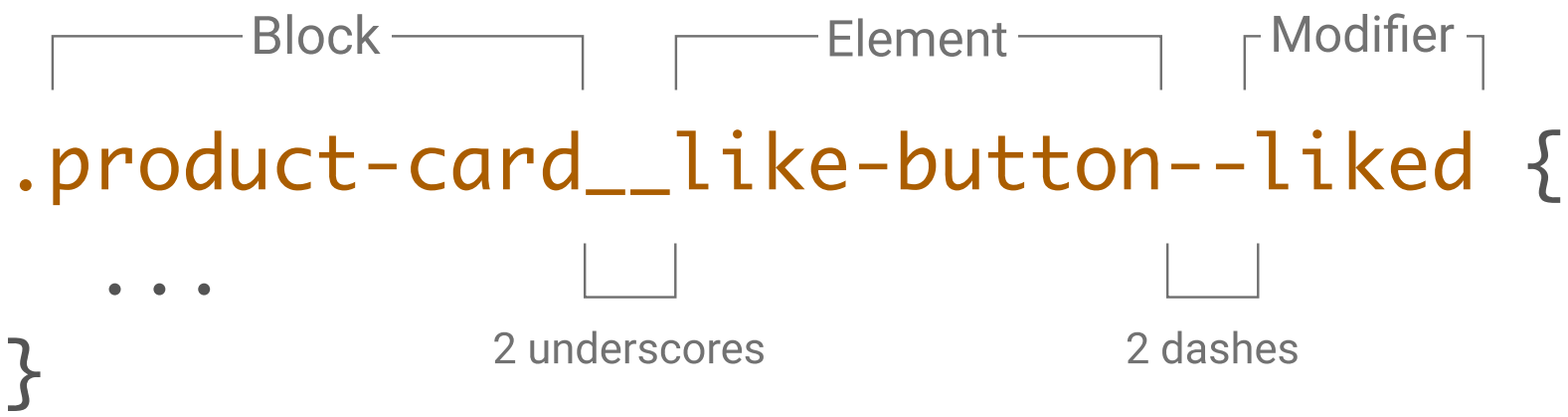
1. Détection des codes inutilisés (dans les DevTools)
2. Suppression des codes inutilisés (unusedCSS, PurgeCSS)

OK, alors on fait quoi ?

Les Méthodologies

la Méthodologie "BEM"

Une Convention de nommage à toute épreuve !



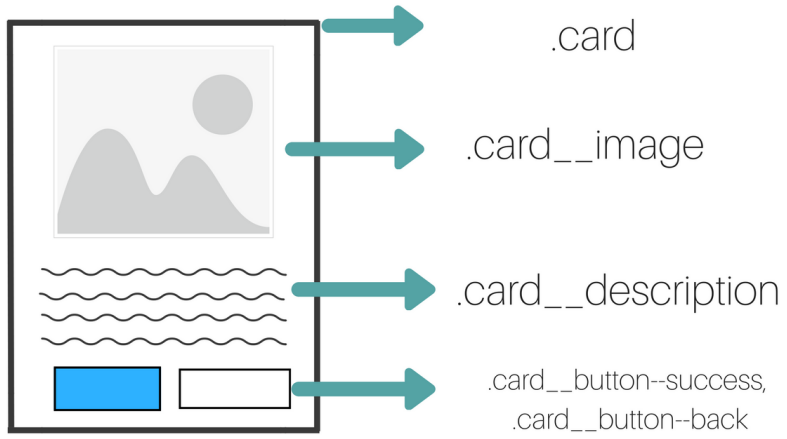
http://getbem.com/

Méthodologie "BEM"

Block - Element - Modifier

```
<p class="author__bio--current">
```

1. **Block**
   entité indépendante et autonome
2. **Element**
   entité dépendante de son Block
3. **Modifier**
   variante d'un Block ou d'un Element



Méthodologie "BEM"

une convention de nommage... contraignante

Nommage des blocks :

`.card` `.contact` `.post` `.main-navigation`

Nommage des éléments :

Règle : block name + __ + element name

`.card__title` `.post__author` `.post__date`

Nommage des modifieurs :

Règle : block name OU element name + -- + modifier name

`.card__title--disabled` `.post--important`

Méthodologie "BEM"

les principes essentiels

```
<p class="author__bio--current">
```

1. Tous les éléments HTML doivent chacun avoir un nom, sous forme de classe(s) CSS (pas de nommage via *id*)

2. Les sélecteurs CSS ne doivent cibler que des classes (pas de sélecteur *nav*, ni *a*, ni *ul* par exemple)

3. Les sélecteurs CSS composés sont à éviter (pas de *.menu .list*, ou de *.navigation > a*)

Frameworks "BEM" : Bootstrap (en partie), bemskel, BULMA

Méthodologie "BEM"

concrètement ?

```
<nav class="navigation">
  <ul>
    <li><a href="" class="active"></a></li>
    <li><a href=""></a></li>
    <li><a href=""></a></li>
    <li><a href=""></a></li>
  </ul>
</nav>
```

aheum, ça c'est pas du BEM hein ?

OK ouais je vois ce que tu veux dire

```
<nav class="navigation">
  <ul class="navigation__list">
    <li class="navigation__list-item"><a href="" class="navigation__link navigation__link--active"></a></li>
    <li class="navigation__list-item"><a href="" class="navigation__link"></a></li>
    <li class="navigation__list-item"><a href="" class="navigation__link"></a></li>
    <li class="navigation__list-item"><a href="" class="navigation__link"></a></li>
  </ul>
</nav>
```

La réutilisabilité dans BEM

BEM

```
.card
  .card__image
  .card__body
    .card__title
    .card__subtitle
    .card__button-cta
```

Ce titre, ce sous-titre et ce bouton doivent-ils vraiment être conditionnés par leur block "card" ?

Pourquoi ne pas les rendre réutilisables ?

pas BEM

```
.card
  img
  figcaption
    .heading-3
    .text-intro
    .button-primary
```

Ce titre, ce sous-titre et ce bouton sont réutilisables dans tout le projet.

(oui mais on retombe sur le problème du nommage des éléments 🤯)

(et il peut y avoir conflits de styles entre composants)

pros 👌

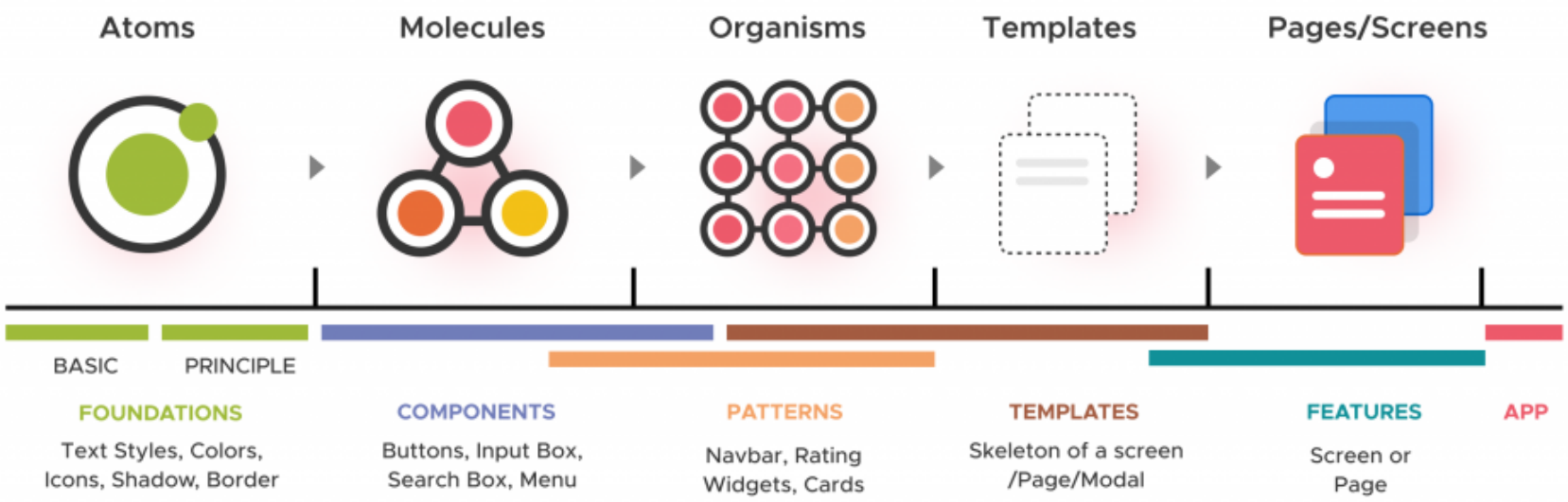
- ☑ Pas de surprises de nommage
- ☑ Pas de problème de cascade
- ☑ "Scalable" : excellente évolutivité du projet
- ☑ Écosystème et documentation riches (tutos, plugins navigateurs, etc.)
- ☑ S'intègre facilement à d'autres outils CSS (Bootstrap, Sass, etc.)

cons 🤨

- ☑ Fichier HTML alourdi et complexifié en raison de (longues) classes sur chaque élément
- ☑ Pas toujours évident de trancher (Block ou Element ?)
- ☑ Problème de réutilisabilité des composants

la Méthodologie "Atomique"

l'ère des classes utilitaires !

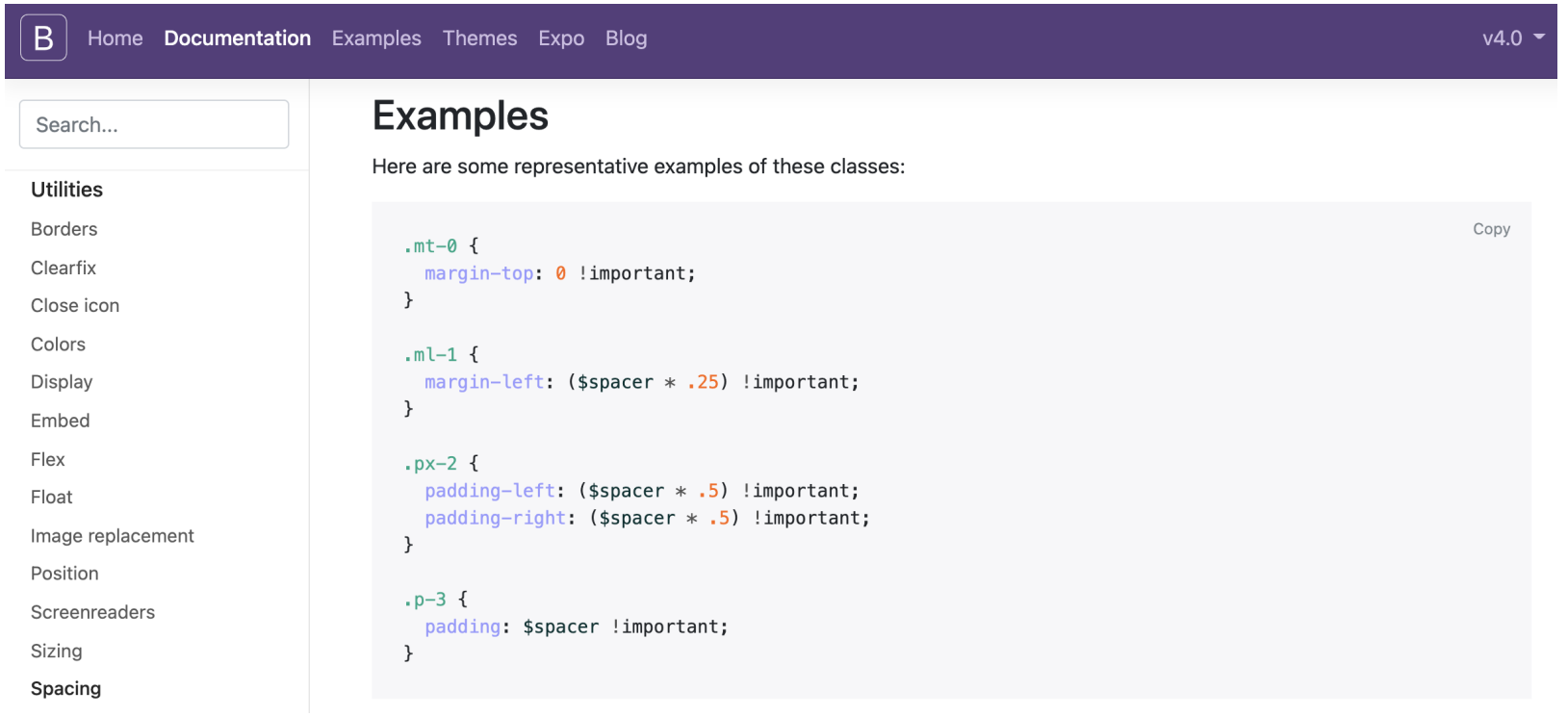


L'organisation atomique, source Laptrinhx

Méthodologie "Atomique"

les classes atomiques dans Bootstrap

```
<div class="d-block mr-1 p-2 bg-primary text-white">
```



https://getbootstrap.com/docs/4.0/utilities/spacing/

Méthodologie "Atomique"

les classes atomiques dans Tailwind

```
<div class="block mr-1 rounded bg-white md:bg-hotpink">
```

```
<div class="md:flex bg-white rounded-lg p-6">
  <img class="h-16 w-16 md:h-24 md:w-24 rounded-full mx-auto md
  <div class="text-center md:text-left">
    <h2 class="text-lg">Erin Lindford</h2>
    <div class="text-purple-500">Product Engineer</div>
    <div class="text-gray-600">erinlindford@example.com</div>
    <div class="text-gray-600">(555) 765-4321</div>
  </div>
</div>
```

Erin Lindford

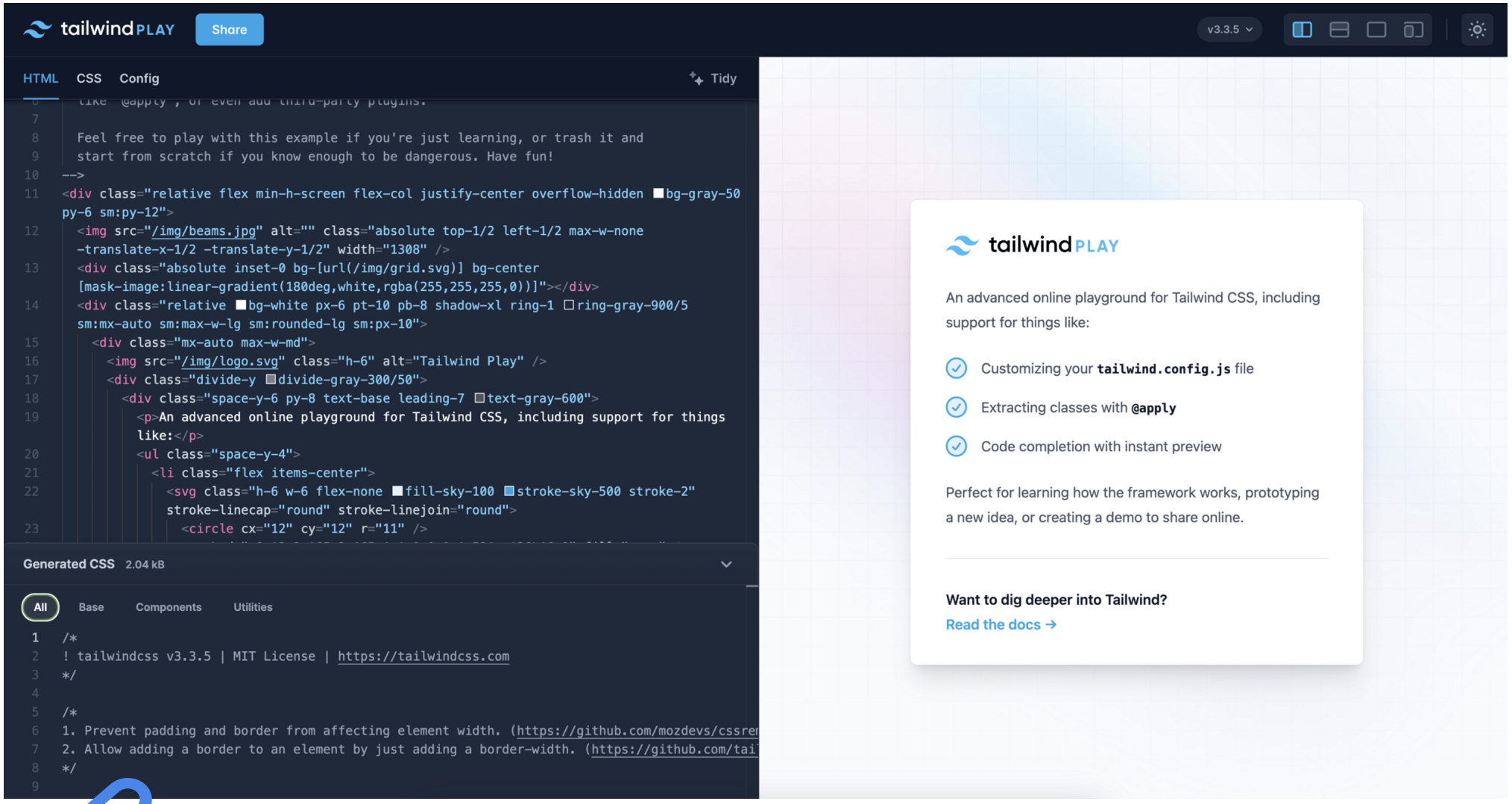
Product Engineer

erinlindford@example.com

(555) 765-4321

https://tailwindcss.com/

Tu veux tester ?



https://play.tailwindcss.com/

Méthodologie "Atomique"

les principes essentiels

```
<p class="block mr-1 rounded bg-white md:bg-hotpink">
```

1. À chaque classe CSS correspond une seule fonction
(ex. *.text-left {text-align: left}* ou *.mr-2 {margin-right: 2rem}*)

2. Les styles CSS sont dénués de contexte pour être totalement indépendants de la structure HTML

3. Les sélecteurs CSS composés n'existent pas
(pas de *.menu .list*, ou de *.navigation > a*)

Frameworks "atomiques" : Bootstrap (en partie), Tailwind, Tachyons, Windi CSS, ...

pros 👌

- ☑ Pas de code mort
- ☑ Pas de problème de nommage
- ☑ Pas de problème de cascade
- ☑ Pas de duplication de sélecteurs
- ☑ Pas besoin de chercher du CSS dans X fichiers
- ☑ Toutes les valeurs dans un seul fichier de config
- ☑ Les classes utilitaires gèrent très finement les styles et leurs variations.
- ☑ L'outil s'adapte à différents contextes (responsive, survol/focus, dark mode, etc.).
- ☑ Écosystème riche (tutos, bibliothèques de composants, plugins navigateurs, plugins IDE : intellisense, etc.)
- ☑ Pas besoin d'autres outils CSS (Bootstrap, Sass, etc.)

cons 🤨

- ☑ Lecture du code HTML rendue fouillie et complexe (ouais, ça pique les yeux !)
- ☑ Nécessite une rigueur (guidelines) pour ne pas écrire n'importe quoi et dans n'importe quel ordre.
- ☑ Besoin d'une cheatsheet TW, parce que... *items-baseline backdrop-invert-0 md:row-start-5 sm:content-around leading-snug dark:tracking-wider*
- ☑ Impossible de trouver des éléments précis dans le code (modale, wrapper, navigation, etc.).
- ☑ Inadapté pour les layouts "complexes" (gabarits, grilles non classiques, flexbox exotique, etc.).
- ☑ Inadapté pour pas mal de fonctionnalités (transitions, animations, filtres, transformations)

Joshua
@CreeCoder

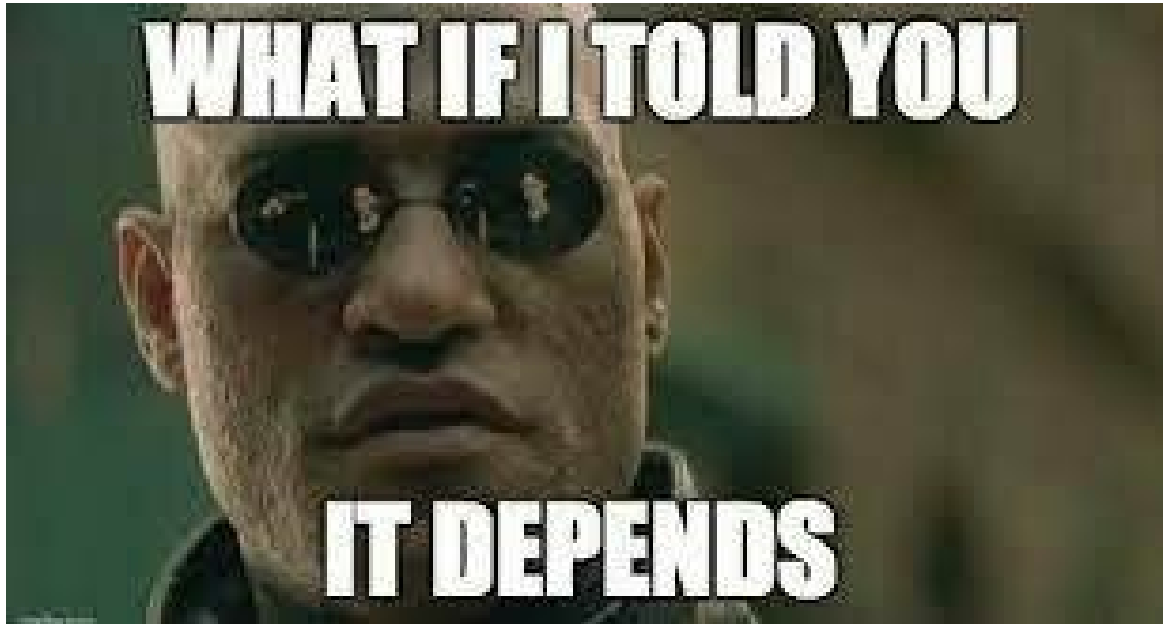
tailwind devs typing out a class name

Traduire le post

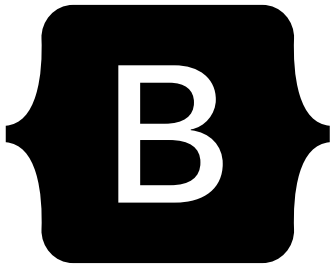
6:51 PM · 19 oct. 2024 · **79,3 k** vues

Bon alors au final

on fait quoi de tout ça maintenant ?

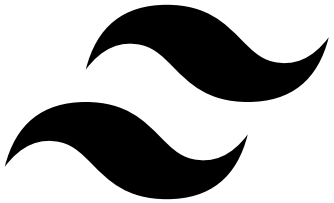


Bootstrap



- spotify
- etsy
- fox news
- reuters
- twitter -> 𝕏
- linkedin
- snapchat
- mastercard
- duolingo
- alibaba

Tailwind
(utility / atomic)



- netflix
- shopify
- twitch
- medium
- starbucks
- prestashop
- stackoverflow
- leafly
- blizzard
- github copilot

BEM

- yandex
- the guardian
- BBC
- financial times
- linkedin
- shopify
- duckduckgo
- uk.gov
- dropbox
- jetbrains

sources : https://stackshare.io/ , builtwith.com, wappalyzer.com (entre autres)

Choix du full-utilities

> ***“ Nous utilisons TailwindCSS pour tous nos projets en front-end.***

source : Guidelines CSS Alsacréations (**2021**)

🤷 **Au bout de deux ans, notre bilan est mitigé**

Concrètement : on a besoin de nommer les choses (ex. .navigation, .modal)

Choix du "ça dépend"

> ***“ Nous utilisons les classes utilitaires uniquement lorsqu'il s'agit de propriétés liées aux espacements (margin, padding, gap), à la typographie et aux couleurs.***

source : Guidelines CSS Alsacréations (**2023**)

🤦 **L'information est éparpillée et on ne trouve pas forcément ce qu'on cherche au bon endroit**

Concrètement : *display: grid* vs *gap* ou *position* vs *top|right|bottom|left* ou *border-width* vs *border-color*

Choix du no-utilities

> ***“ On n'utilise pas Tailwind par défaut, sauf si les contraintes du projet, ou le client, l'exigent.***

source : Guidelines CSS Alsacréations (2025)

🤔 Pas de classes utilitaires, mais…

Concrètement : notre outillage interne permet de s'en passer

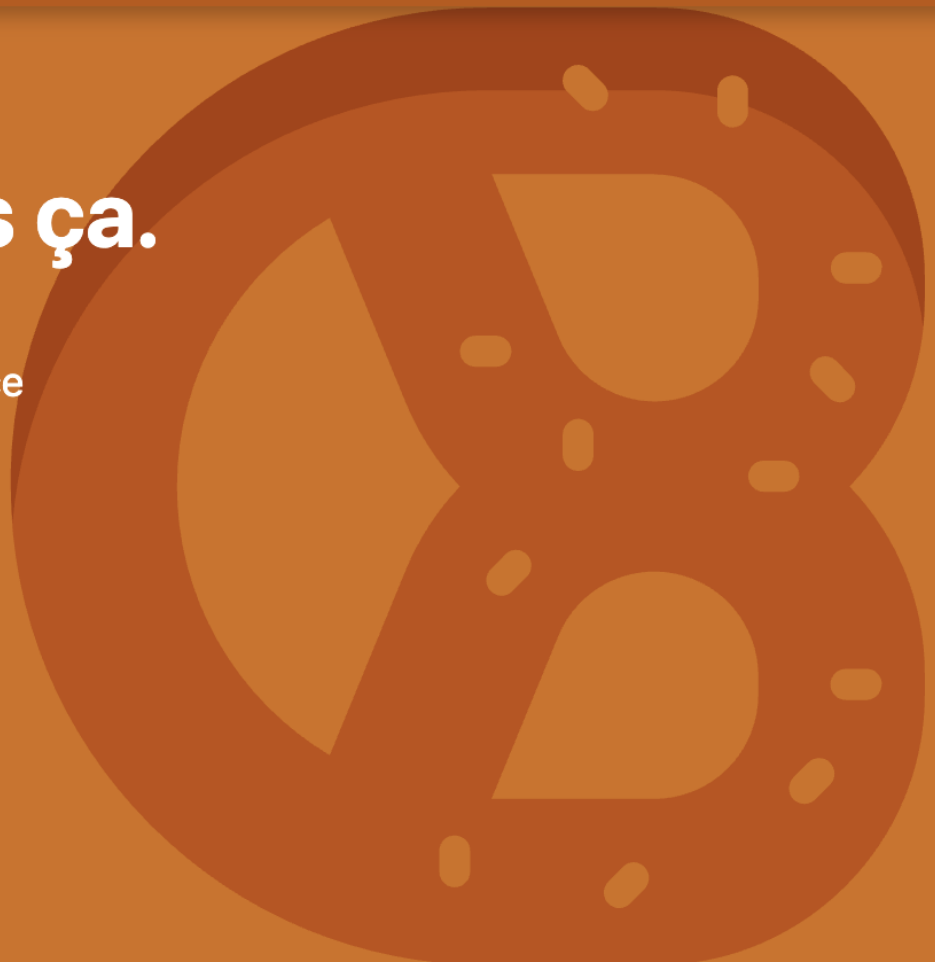
BRETZEL

pour avoir la *class* en Alsace.

Tu vois Bootstrap ? Bah, c'est pas ça.

Bretzel est un ensemble de Layouts HTML/CSS utilitaires réunis par l'agence Alsacréations pour ses projets d'intégration.

- Voir et récupérer nos Layouts CSS
- Télécharger la Cheatsheet (PDF 700ko)
- Installer l'extension Visual Studio Code

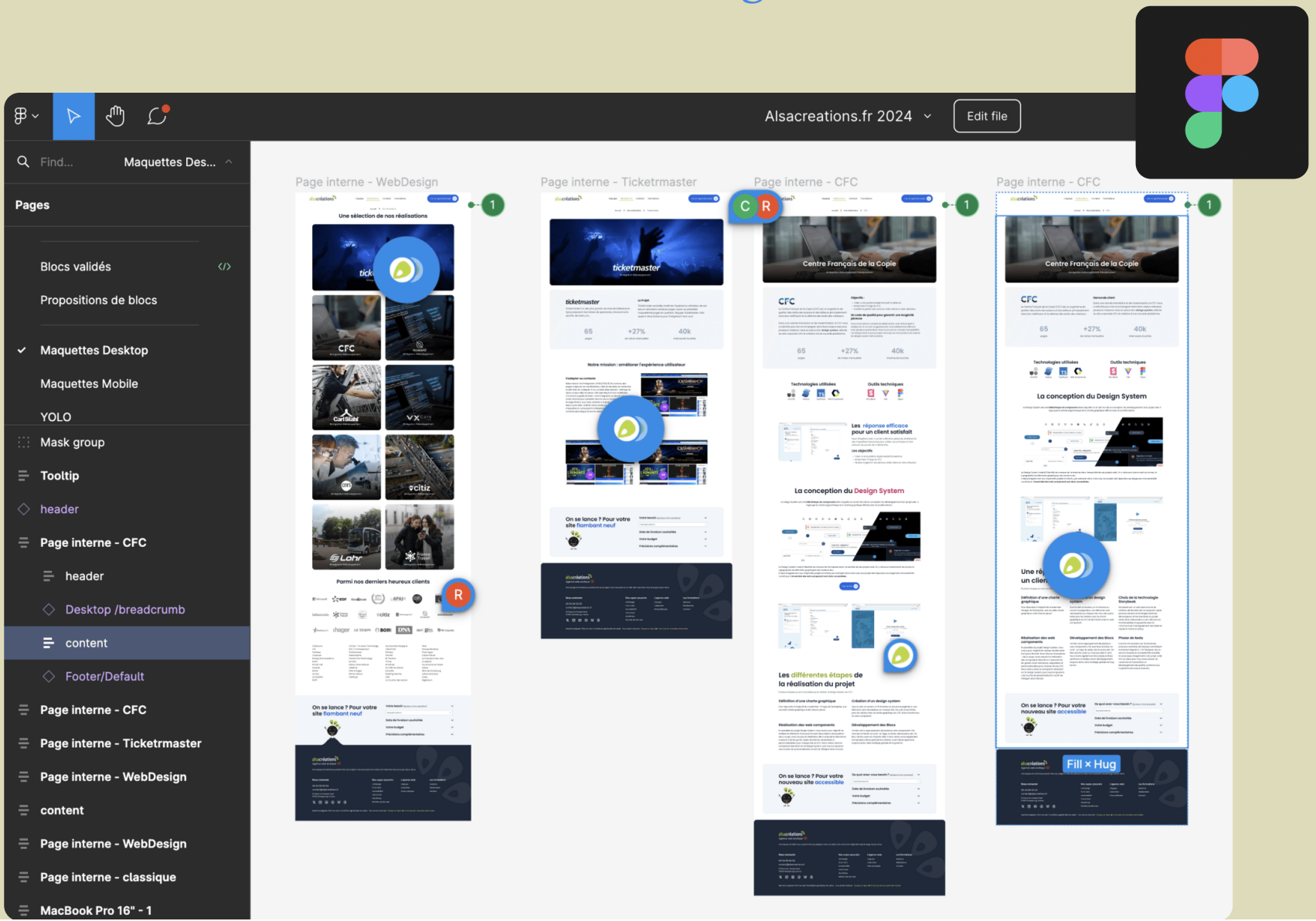


https://bretzel.alsacreations.com/

	tailwindcss	alsacréations
Reset CSS	`@import "tailwindcss/preflight.css" layer(base);`	`@import "reset.css" layer(config);`
Tailles de polices	`text-4 md:text-6 lg:text-8`	`--text-m (variables avec clamp())`
Espaces, gouttières	`mt-2 md:mt-4`	`--spacing-s (variables avec clamp())`
Couleurs (dark mode)	`bg-hotpink dark:bg-pink`	`--on-surface (variables avec light-dark())`
Layouts	`grid grid-cols-2 gap-4`	`data-layout="" (layouts HTML)`

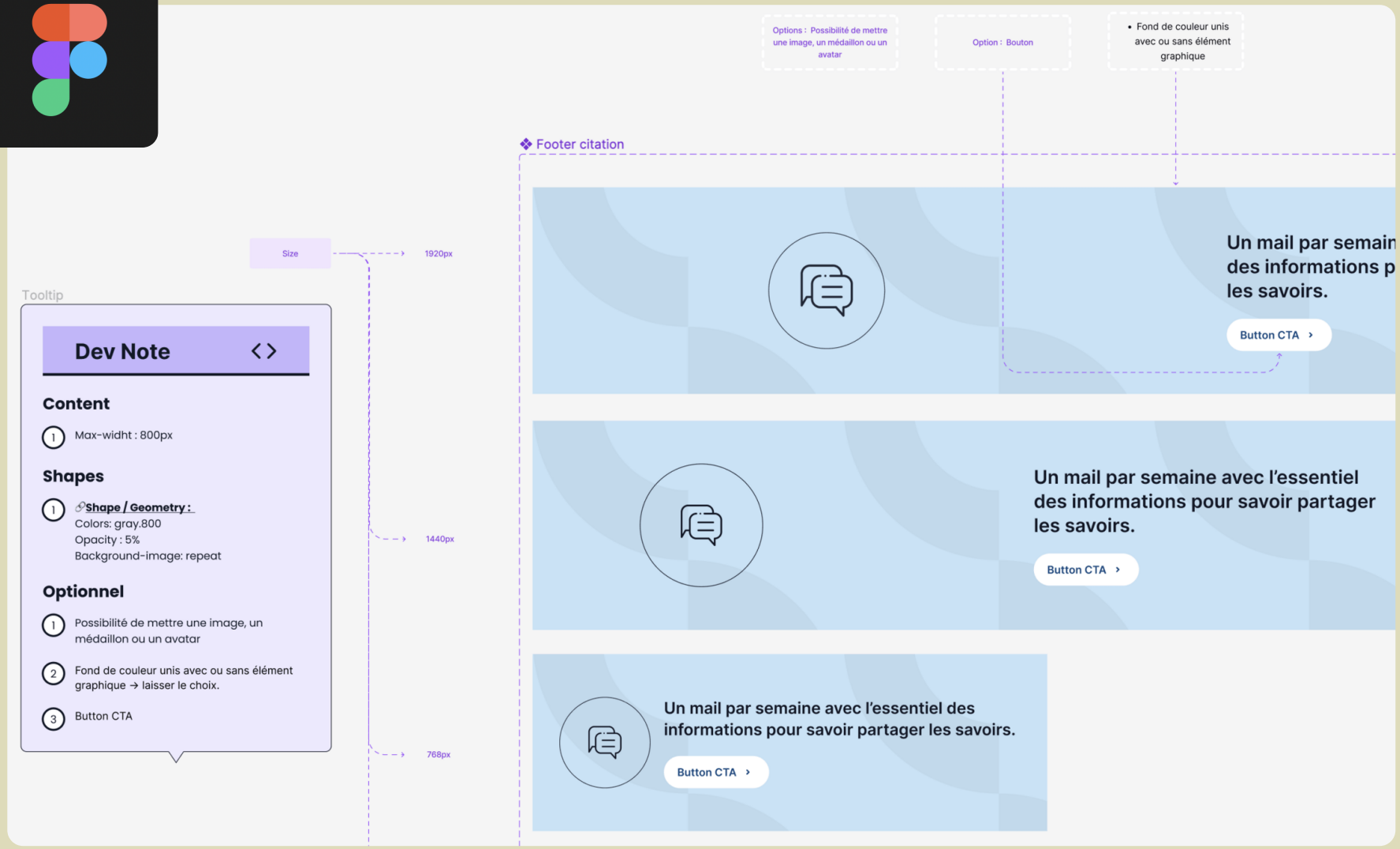
du design à l'intégration

UI-Kit + bonne webdesigneuse = 💖



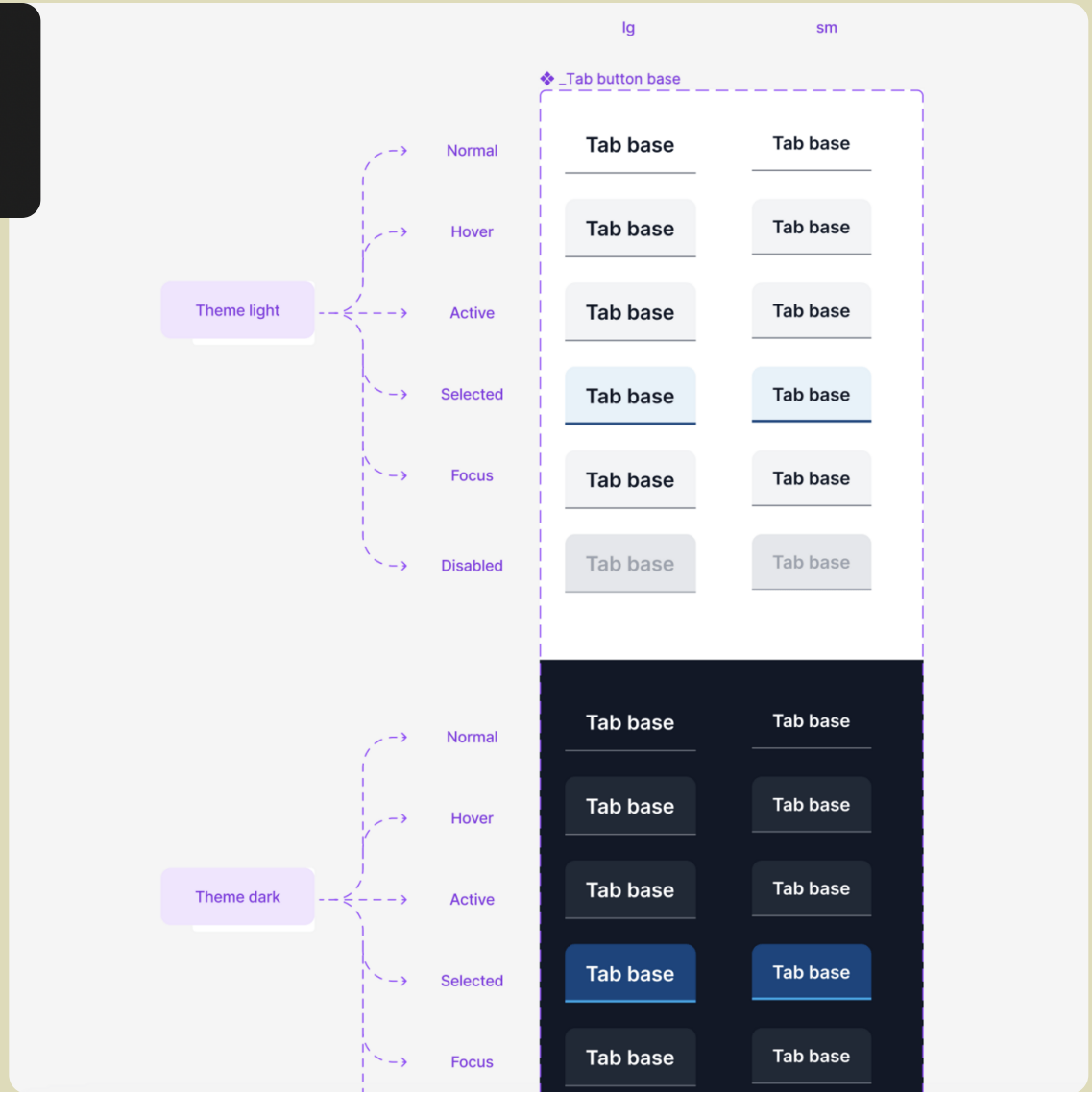
UI-Kit indispensable

afficher des specs dans l'UI-Kit



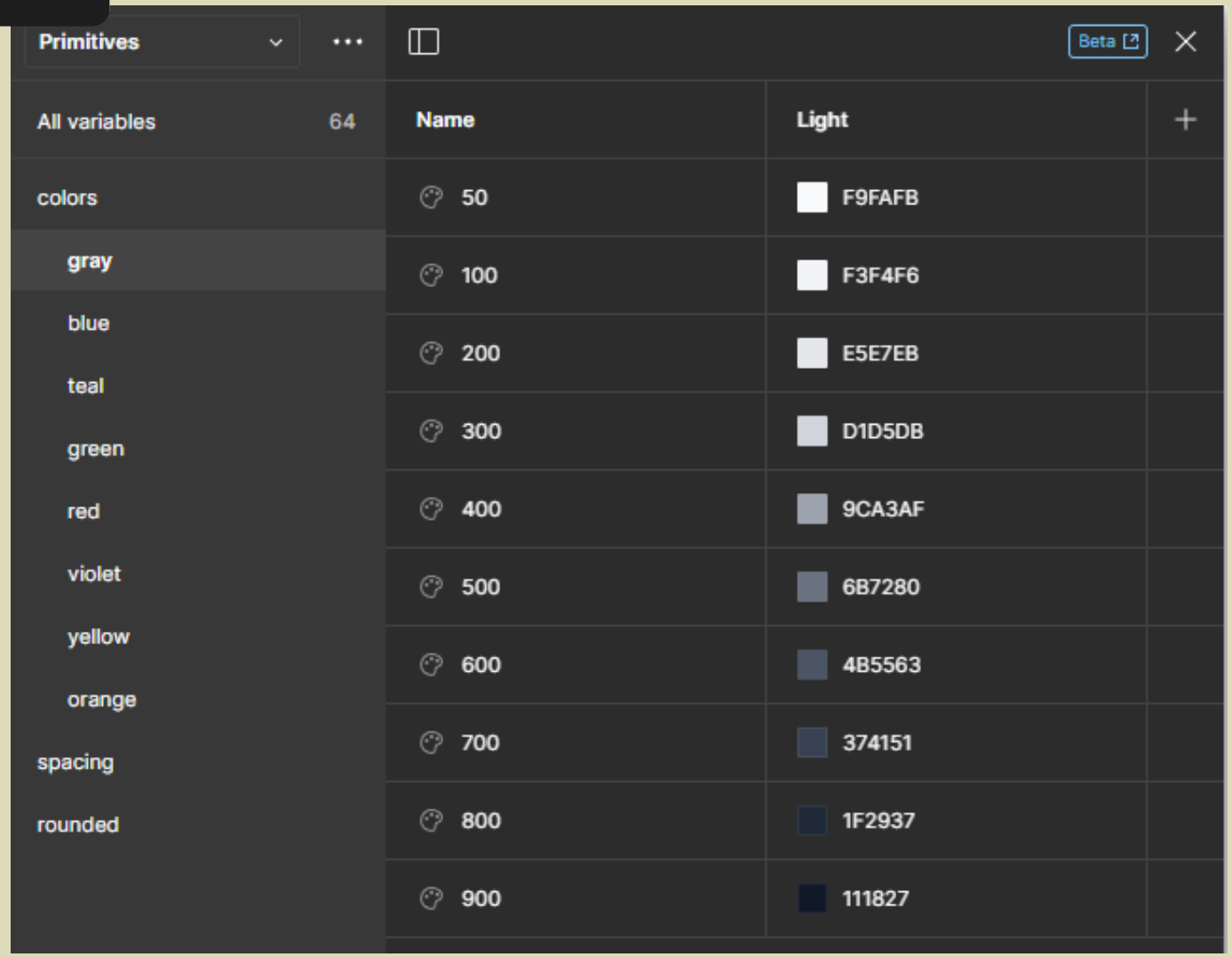
UI-Kit *complet*

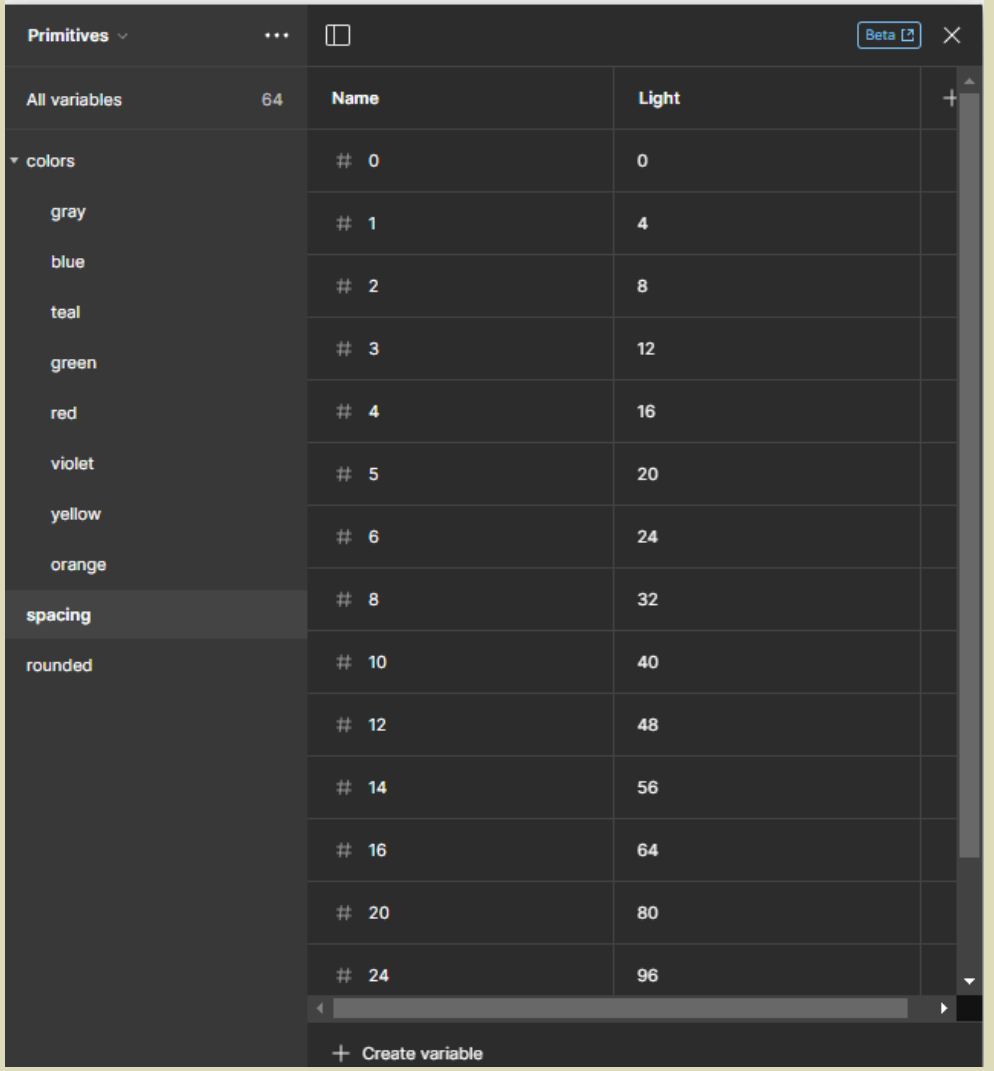
Penser à présenter toutes les variantes dans l'UI-Kit



Des variables dès la conception

ces variables seront transmises au développeurs

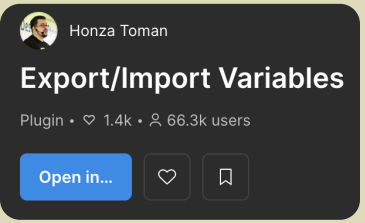


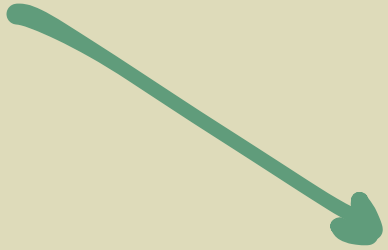


Figma vers CSS

Figma

Plugin Export/Import Variables





theme.css

```css
/* ---------------------------------
 * Theme du projet (valeurs primitives)
 * ---------------------------------
 */

/* stylelint-disable */
/* Breakpoints custom (si compatibles) */
@custom-media --md (width >= 48rem);
@custom-media --lg (width >= 64rem);
@custom-media --xl (width >= 80rem);
@custom-media --xxl (width >= 96rem);
@custom-media --until-md (width < 48rem);
@custom-media --until-lg (width < 64rem);
@custom-media --until-xl (width < 80rem);
@custom-media --until-xxl (width < 96rem);
/* stylelint-enable */

:root {
```

theme-tokens.css

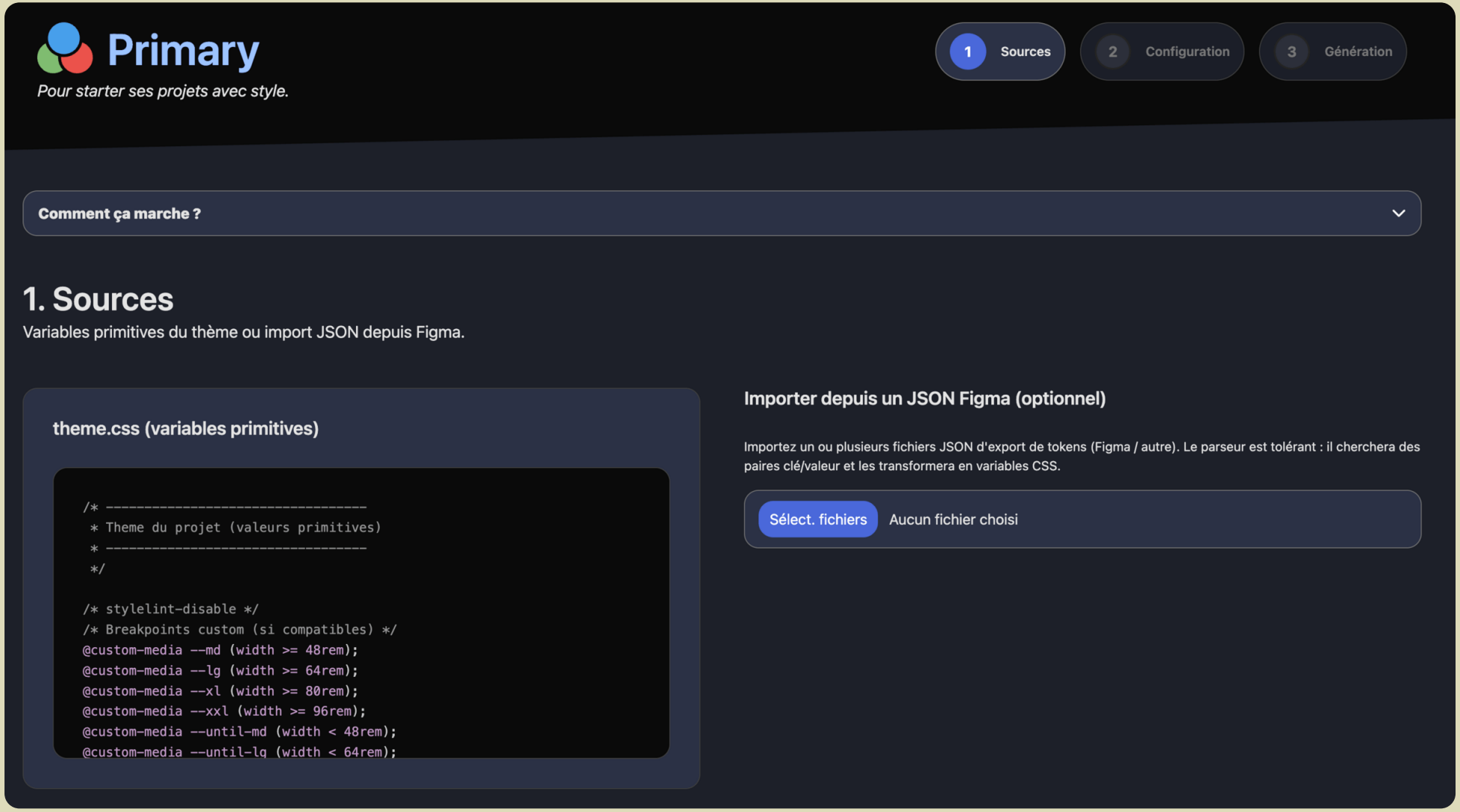
```css
/* ---------------------------------
 * Theme-tokens, généré par primary.alsacreations.com
 * Surcouche de theme.css
 * Configuration :
 * - Couleur primaire : blue
 * - Theme : light et dark
 * - Typographie responsive : oui
 * - Espacements responsive : oui
 * ---------------------------------
 */
:root {
  color-scheme: light dark;

  &[data-theme="light"] {
    color-scheme: light;
  }

  &[data-theme="dark"] {
```

Figma vers CSS

primary.alsacreations.com



kiwipedia

ressources et guidelines Alsacréations

README

Kiwipedia, made by Alsacréations 🥝

Kiwipedia est une base de connaissance technique librement partagée et dont les objectifs sont :

- D'uniformiser et d'harmoniser les processus de conception de l'agence web Alsacreations.fr.
- De favoriser l'intervention de profils de compétences variés au sein d'une équipe.
- De faciliter la maintenance des projets.

💬 Le Web est un métier de plus en plus tentaculaire, comportant un vaste ensemble de thématiques qui vont avoir au final un impact sur le succès auprès de l'internaute : **l'accessibilité, la qualité, l'ergonomie, la performance, le référencement (SEO), la maintenabilité**. Tous les critères ne seront pas toujours applicables et le seront bien souvent en fonction du temps alloué.

Cette présente base de connaissances techniques est le fruit de note veille technologique quotidienne et se compose de :

- **Guidelines** (bonnes pratiques internes concernant divers langages et technologies web)
- **Ressources** (documentations diverses, checklists, cheatsheets)
- **Starters** (tutoriels rapides pour initier des projets)
- **Configs** (fichiers de configuration tels que `.editorconfig`, `stylelint.config.js`, `tsconfig.json`, `settings.json` pour VScode)

Contributors 11

Deployments 11

github-pages 2 years ago

+ 10 deployments

www.kiwipedia.fr

OK, la suite ?

et sois en forme !