

Frameworks CSS

Comprendre, choisir et maîtriser



⌚ Pourquoi les frameworks CSS existent

❓ Quand les utiliser

🛡️ Comment garder le contrôle de son CSS

Un framework CSS est un ensemble préconstruit de règles CSS, de classes et de composants conçu pour accélérer le développement de l'interface d'un site ou d'une application.

Autrement dit, c'est comme un kit de construction visuel et structurel pour ton projet web.

Un framework CSS est un outil qui **accélère le développement et impose une structure**, mais il ne **remplace jamais la maîtrise du CSS ni la réflexion sur le design**.

Le CSS avant les frameworks

Le CSS avant les frameworks

CSS “bricolé”

- Styles ajoutés sans logique

```
.navbar ul li a { ... }  
header nav ul li a { ... }  
    !important!
```

Fichiers géants

- 2000 lignes mélangées

Spécificité incontrôlée

- Sélecteurs qui se contredisent
-

Le CSS avant les frameworks

Les développeurs ajoutaient des règles CSS au fur et à mesure du projet. Il n'y avait souvent aucune organisation, ni méthodologie.

```
h1 { font-size: 24px; color: blue; }
h1.title { margin-top: 20px; }
.header h1 { font-weight: bold; }
```

- Les règles se contredisent parfois
- Chaque nouvelle feature peut casser une précédente

Le CSS avant les frameworks

Tous les styles étaient regroupés dans un seul gros fichier ou quelques fichiers mal structurés.

Impossible de retrouver facilement une règle, surtout dans un projet complexe.

```
style.css (2000 lignes)
- header
- footer
- boutons
- formulaire
- homepage
- page produit
- media queries
- overrides par ci par là
```

- Temps perdu à chercher une classe ou un style
- Difficultés à travailler en équipe
- Risque d'effets de bord (une modification casse autre chose)

Le CSS avant les frameworks

Le CSS applique les règles selon la spécificité des sélecteurs.

Dans les fichiers non organisés, on a souvent des sélecteurs trop complexes ou conflictuels.

```
.navbar ul li a { color: red; }
.header nav ul li a { color: blue; }
```

- Quelle couleur l'élément <a> prendra-t-il ?
- Les développeurs ajoutent parfois !important pour “forcer” la règle → chaos garanti

Reset CSS / Normalize.css



Reset CSS / Normalize.css

coucou

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Voluptatem obcaecati asperiores nemo, libero quia cumque reprehenderit accusantium, cum, voluptas non fuga quos. Cumque assumenda possimus unde blanditiis ea mollitia alias?

Lien test ▾
test

coucou

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Voluptatem obcaecati asperiores nemo, libero quia cumque reprehenderit accusantium, cum, voluptas non fuga quos. Cumque assumenda possimus unde blanditiis ea mollitia alias?

Lien test ▾
test

coucou

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Voluptatem obcaecati asperiores nemo, libero quia cumque reprehenderit accusantium, cum, voluptas non fuga quos. Cumque assumenda possimus unde blanditiis ea mollitia alias?

Lien test ▾
test

Autre problèmes, chaque navigateur avait ses styles par défaut. Un même HTML s'affichait différemment selon le navigateur

Reset CSS / Normalize.css

Du coup, les devs se sont à créer des “reset”

Avantages :

Base uniforme

Contrôle total

Inconvénients :

Trop brutal

Supprime aussi des styles utiles (listes, titres...)

<https://meyerweb.com/eric/tools/css/reset/>

```
/* http://meyerweb.com/eric/tools/css/reset/
v2.0 | 2010126
License: none (public domain)
*/
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
  font: inherit;
  vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
  display: block;
}
body {
  line-height: 1;
}
ol, ul {
  list-style: none;
}
blockquote, q {
  quotes: none;
}
blockquote::before, blockquote::after,
```

Reset CSS / Normalize.css

Avantages :

- Ne supprime pas tout
- Harmonise les styles par défaut
- Respecte les standards HTML
- Plus doux qu'un reset

Inconvénients :

- Uniformité entre les sites
- Meilleure accessibilité par défaut

```
/*! normalize.css v8.0.1 | MIT License | github.com/necolas/normalize.css */

/* Document
=====
*/

/**
 * 1. Correct the line height in all browsers.
 * 2. Prevent adjustments of font size after orientation changes in iOS.
 */
html {
  line-height: 1.15; /* 1 */
  -webkit-text-size-adjust: 100%; /* 2 */
}

/* Sections
=====
*/

/**
 * Remove the margin in all browsers.
 */
body {
  margin: 0;
}

/**
 * Render the `main` element consistently in IE.
 */
main {
  display: block;
```

<https://github.com/necolas/normalize.css/blob/master/normalize.css>

Reset CSS / Normalize.css

“Reset et Normalize sont les premières briques vers les frameworks CSS.”

Donc, si on résume, avant les frameworks CSS :

- Maintenance & évolutions difficiles
- Beaucoup de temps de développement
- Des incohérences visuelles
- Travail en équipe difficile

Structure et cohérence

- Fournit un système de grille pour le layout
- Définit des styles de base pour titres, paragraphes, boutons, formulaires
- Assure que tout a un aspect cohérent

Composants prêts à l'emploi

- Boutons, cartes, modals, menus, etc.
- Évite de réinventer la roue

Productivité

- Gagner du temps au lieu de coder chaque composant à partir de zéro
- Facilite le travail en équipe grâce à des conventions communes

Compatibilité navigateur

- Gère les différences entre navigateurs

Ce n'est pas un moteur de design → le framework propose un style,
mais il n'impose pas forcément le bon choix graphique

Ce n'est pas obligatoire → on peut créer un site 100% custom avec du
CSS structuré

Ce n'est pas magique → comprendre le CSS reste essentiel

TYPE DE FRAMEWORKS

Component-based

Exemple : Bootstrap, Bulma

Classes “sémantiques”, composants pré-stylés

Avantages : rapide, cohérent

Inconvénients : peut imposer un design uniforme, lourd

Utility-first

Exemple : Tailwind

Classes atomiques (p-4, text-center)

Avantages : flexibilité, design sur mesure

Inconvénients : HTML plus chargé, lisibilité réduite

Minimalistes / Low-level

Exemple : Pico.css, Skeleton

Peu d'opinion, petites bases

Avantages : léger, flexible

Inconvénients : moins de composants prêts

TYPE DE FRAMEWORKS

Creation : 19 août 2011



Dans quel contexte Bootstrap a été créé ?

- Explosion du web responsive
- Besoin de rapidité et de cohérence
- Équipes nombreuses (Twitter à l'origine)

Ce que Bootstrap apporte :

- Grille responsive
- Composants prêts à l'emploi
- Styles par défaut cohérents
- Documentation claire

```
<button class="btn btn-primary">Valider</button>
```

Les limites de Bootstrap :

- Design reconnaissable
- CSS parfois lourd
- Personnalisation complexe

TYPE DE FRAMEWORKS

Creation : 2017



Dans quel contexte Tailwind a été créé ?

- Rejet des designs “Bootstrap-like”
- Meilleure maîtrise du CSS
- Tooling moderne (PurgeCSS, PostCSS)

Ce que Tailwind apporte :

- Une classe = une propriété CSS
- Pas de composants imposés
- Design 100% custom
- Pas de CSS mort
- Grande flexibilité

Les limites de Tailwind :

- HTML plus chargé
- Courbe d'apprentissage
- Dépendance forte à l'outil
- Régression sur le principe du css

```
<div class="p-4 bg-blue-500 text-white rounded">
```

```
<div className="relative flex place-items-center before:absolute before:h-[300px] before:w-[480px] before:-translate-x-1/2 before:rounded-full before:bg-gradient-radial before:from-white before:to-transparent before:blur-2xl before:content-[''] after: absolute after:z-20 after:h-[180px] after:w-[240px] after:translate-x-1/3 after:bg-gradient-conic after:from-sky-200 after:via-blue-200 after:blur-2xl after:content-[''] before:dark:bg-gradient-to-br before:dark:from-transparent before:dark:to-blue-700 before:dark:opacity-10 after:dark:from-sky-900 after:dark:via-[#0141ff] after:dark:opacity-40 before:lg:h-[360px] z-[-1]">
  <Image
    className="relative dark:drop-shadow-[0_0_0.3rem_#ffffff70] dark:invert"
    src="/next.svg"
    alt="Next.js Logo"
    width={180}
    height={37}
    priority
  />
</div>
```

Le CSS évolu....

```
@import url('header.css');  
@import url('home.css');  
@import url('contact.css');  
@import url('footer.css');
```

Les “@import” peuvent aider à améliorer l’organisation.

Cependant.... **1 import = 1 appel au serveur** donc c'est organisé mais pas optimisé !

Le CSS évolu....

```
.header{  
    nav{  
        a{  
        }  
    }  
    .navbar{  
        a{  
        }  
    }  
}
```

La nouvelle syntaxe CSS “nested” permet de mieux éviter les conflits et d'avoir une meilleure structure du code.

Le CSS évolu....

```
:root{  
  --white: #fff;  
  --gap:50rem;  
}  
  
.content > *:not(:last-child){  
  ...  
}  
  
.card-wrapper {  
  container-type: inline-size;  
}  
  
@container (min-width: 400px) {  
  .card {  
    display: flex;  
  }  
}
```

Les variables CSS permettent une meilleure homogénéisation. En plus, elle sont accessibles depuis le JS !

Il y a aussi :

- Les selecteurs relationnels (`;has()`, `:is()`, `:not()`,...)
- Layout unidimensionnel (flexbox)
- Layouts complexes (grid)

Beaucoup de problèmes autrefois résolus par les frameworks sont désormais résolus nativement par le CSS et le tooling.

Pourquoi les frameworks étaient nécessaires avant

- Normaliser les navigateurs
- Gérer le responsive
- Fournir une grille
- Mutualiser les styles
- Accélérer le développement
- Éviter le CSS spaghetti

→ À l'époque : indispensable

Ce qui a changé
 **CSS moderne**

- Flexbox
- Grid
- Custom properties
- Container queries
- Sass / PostCSS
- Autoprefixer
- PurgeCSS
- CSSNano

Un framework n'est plus une nécessité technique, c'est un choix stratégique.