Concepts de base

Principes fondamentaux de l'utilitaire d'abord

Construire des composants complexes à partir d'un ensemble restreint d'utilitaires primitifs.

Traditionnellement, chaque fois que vous avez besoin de styliser quelque chose sur le Web, vous écrivez CSS.

Utilisation d'une approche traditionnelle où les conceptions personnalisées nécessitent un CSS personnalisé

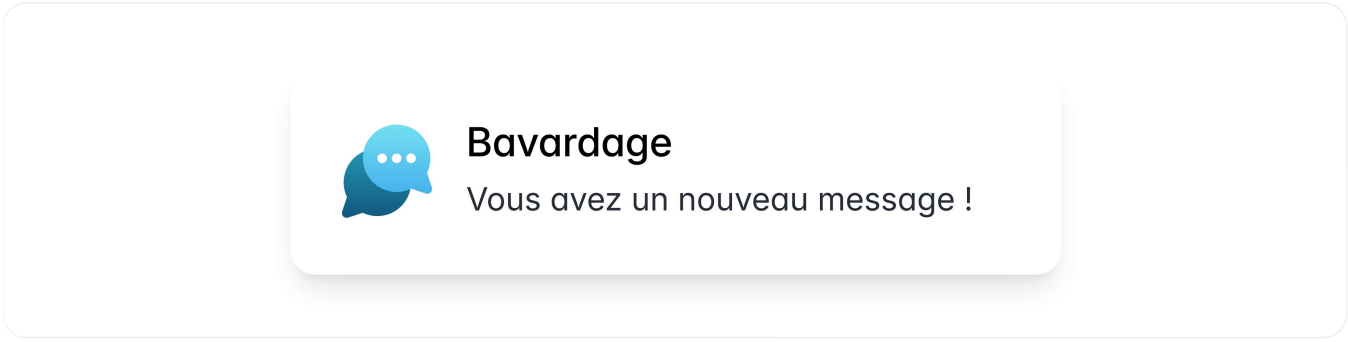

```
<div class="chat-notification">
  <div class="chat-notification-logo-wrapper">
    <img class="chat-notification-logo" src="/img/logo.svg" alt="ChitChat Lo
  </div>
  <div class="chat-notification-content">
    <h4 class="chat-notification-title">ChitChat</h4>
    <p class="chat-notification-message">You have a new message!</p>
  </div>
</div>

<style>
  .chat-notification {
    display: flex;
```

```
    align-items: center;
    max-width: 24rem;
    margin: 0 auto;
    padding: 1.5rem;
    border-radius: 0.5rem;
    background-color: #fff;
    box-shadow: 0 20px 25px -5px rgba(0, 0, 0, 0.1), 0 10px 10px -5px rgba(0
  }
  .chat-notification-logo-wrapper {
    flex-shrink: 0;
  }
  .chat-notification-logo {
    height: 3rem;
    width: 3rem;
  }
  .chat-notification-content {
    margin-left: 1.5rem;
  }
  .chat-notification-title {
    color: #1a202c;
    font-size: 1.25rem;
    line-height: 1.25;
  }
  .chat-notification-message {
    color: #718096;
    font-size: 1rem;
    line-height: 1.5;
  }
</style>
```

Avec Tailwind, vous stylisez des éléments en appliquant des classes préexistantes directement dans votre HTML.

- **Utilisation de classes utilitaires pour créer des conceptions personnalisées sans écrire de CSS**




```
<div class="p-6 max-w-sm mx-auto bg-white rounded-xl shadow-lg flex items-ce
  <div class="shrink-0">
    <img class="size-12" src="/img/logo.svg" alt="ChitChat Logo">
  </div>
  <div>
    <div class="text-xl font-medium text-black">ChitChat</div>
    <p class="text-slate-500">You have a new message!</p>
  </div>
</div>
```

Dans l'exemple ci-dessus, nous avons utilisé :

- Les utilitaires **flexbox** et **de remplissage** de Tailwind (, , et) pour contrôler la disposition globale de la carte`flex``shrink-0``p-6`
- Les utilitaires **max-width** et **margin** (et) pour contraindre la largeur de la carte et la centrer horizontalement`max-w-sm``mx-auto`
- La **couleur d'arrière-plan**, le **rayon de bordure** et les utilitaires **d'ombre de boîte** (, et) pour styliser l'apparence de la carte`bg-white``rounded-xl``shadow-lg`
- Les utilitaires **de taille** () pour définir la largeur et la hauteur de l'image du logo`size-12`
- Les utilitaires **d'espace** () pour gérer l'espacement entre le logo et le texte`gap-x-4`
- La **taille de la police**, la **couleur du texte** et les utilitaires **font-weight** (, , , etc.) pour styliser le texte de la carte`text-xl``text-black``font-medium`

Cette approche nous permet de mettre en œuvre une conception de composant entièrement personnalisée sans écrire une seule ligne de CSS personnalisé.

Maintenant, je sais ce que vous pensez, *« c'est une atrocité, quel horrible gâchis ! »* et vous avez raison, c'est un peu moche. En fait, il est à peu près impossible de penser que c'est une bonne idée la première fois que vous la voyez – **vous devez vraiment l'essayer**.

Mais une fois que vous avez réellement construit quelque chose de cette façon, vous remarquerez rapidement des avantages vraiment importants :

- **Vous ne gaspillez pas d'énergie à inventer des noms de classe**. Plus besoin d'ajouter des noms de classe stupides comme juste pour pouvoir styliser quelque chose, et plus de se tourmenter sur le nom abstrait parfait pour quelque chose qui n'est vraiment qu'un conteneur flexible.`sidebar-inner-wrapper`
- **Votre CSS cesse de croître**. En utilisant une approche traditionnelle, vos fichiers CSS s'agrandissent chaque fois que vous ajoutez une nouvelle fonctionnalité. Avec les utilitaires, tout est réutilisable, vous avez donc rarement besoin d'écrire de nouveaux CSS.
- **Faire des changements semble plus sûr**. Le CSS est mondial et vous ne savez jamais ce que vous cassez lorsque vous effectuez un changement. Les classes de votre HTML sont locales, vous pouvez donc les modifier sans vous soucier de quelque chose d'autre qui se casse.

Lorsque vous réalisez à quel point vous pouvez être productif en travaillant exclusivement en HTML avec des classes d'utilitaires prédéfinies, travailler d'une autre manière vous fera l'effet d'une torture.

Pourquoi ne pas simplement utiliser des styles en ligne ?

Une réaction courante à cette approche est de se demander : « N'est-ce pas juste des styles en ligne ? » et d'une certaine manière, c'est le cas : vous appliquez des styles directement aux éléments au lieu de leur attribuer un nom de classe, puis de styliser cette classe.

Mais l'utilisation de classes utilitaires présente quelques avantages importants par rapport aux styles en ligne :

- **Concevoir avec des contraintes**. À l'aide des styles en ligne, chaque valeur est un nombre magique. Avec les utilitaires, vous choisissez des styles à partir d'un **système de conception** prédéfini, ce qui facilite grandement la création d'interfaces utilisateur visuellement cohérentes.
- **Conception réactive**. Vous ne pouvez pas utiliser les media queries dans les styles en ligne, mais vous pouvez utiliser les **utilitaires réactifs** de Tailwind pour créer facilement des interfaces entièrement réactives.
- **Survol, focus et autres états**. Les styles en ligne ne peuvent pas cibler des états tels que le survol ou le focus, mais **les variantes d'état de** Tailwind facilitent le style de ces états avec des classes utilitaires.

Ce composant est entièrement réactif et inclut un bouton avec des styles de pointage et de focus, et est entièrement construit avec des classes utilitaires :

Erin Lindford

Ingénieur Produit

Message

```
<div class="py-8 px-8 max-w-sm mx-auto space-y-2 bg-white rounded-xl shadow-
  <img class="block mx-auto h-24 rounded-full sm:mx-0 sm:shrink-0" src="/img
  <div class="text-center space-y-2 sm:text-left">
    <div class="space-y-0.5">
      <p class="text-lg text-black font-semibold">
        Erin Lindford
      </p>
      <p class="text-slate-500 font-medium">
        Product Engineer
      </p>
    </div>
    <button class="px-4 py-1 text-sm text-purple-600 font-semibold rounded-f
  </div>
</div>
```

Problèmes de maintenabilité

Le plus grand problème de maintenabilité lors de l'utilisation d'une approche axée sur l'utilité est la gestion des combinaisons d'utilitaires couramment répétées.

Ce problème est facilement résolu **en extrayant des composants et des partiels**, et en utilisant des **fonctionnalités de l'éditeur et du langage** telles que l'édition de plusieurs curseurs et des boucles simples.

```
<!-- PrimaryButton.vue -->
<template>
  <button class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-
    <slot/>
  </button>
</template>
```

En dehors de cela, la maintenance d'un projet CSS utilitaire s'avère beaucoup plus facile que la maintenance d'une grande base de code CSS, simplement parce que HTML est beaucoup plus facile à maintenir que CSS. De grandes entreprises comme GitHub, Netflix, Heroku, Kickstarter, Twitch, Segment et bien d'autres utilisent cette approche avec beaucoup de succès.

Si vous souhaitez connaître l'expérience d'autres personnes avec cette approche, consultez les ressources suivantes :

- **En chiffres : un an et demi avec Atomic CSS** par John Polacek
- **Non, les classes utilitaires ne sont pas les mêmes que les styles en ligne** par Sarah Dayan d'Algolia
- **Diana Mounter sur l'utilisation des classes utilitaires sur GitHub**, une interview podcast

Pour en savoir plus, consultez **The Case for Atomic/Utility-First CSS,** organisé par **John Polacek**.

Guide de mise à niveau

Survol, focus et autres états

Politique relative aux marques de commerce

Modifier cette page sur GitHub