Les Media Queries

Introduction au Responsive Web Design

Introduction

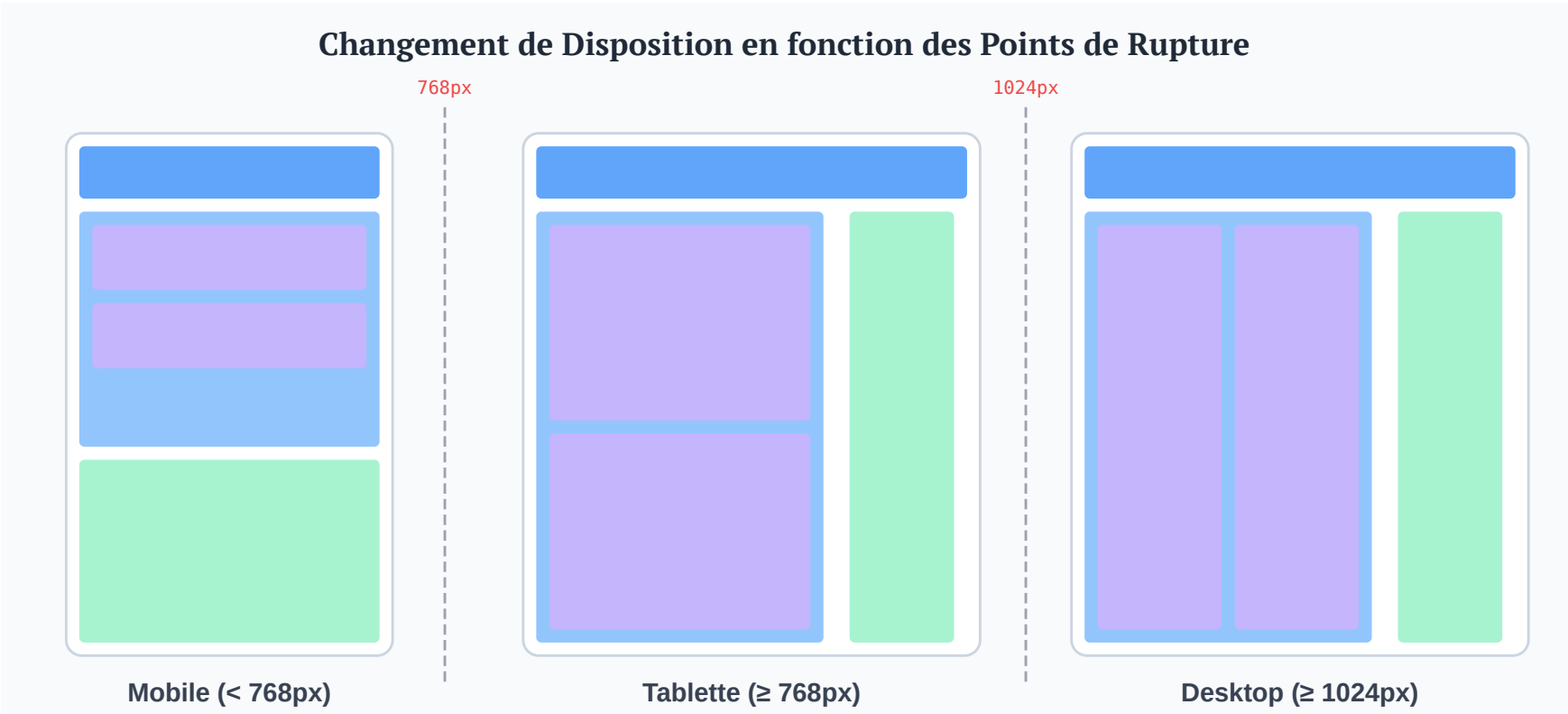
Le Responsive Design

Avec la multiplication des appareils (smartphones, tablettes, ordinateurs), le design web doit s’adapter.

- **L’objectif :** Offrir une expérience de navigation optimale, quelle que soit la taille de l’écran.
- **Le principe :** Le contenu et la mise en page "répondent" et se réorganisent automatiquement en fonction de l’appareil de l’utilisateur.
- **Les 3 modes de rendu :** En général, un site moderne est conçu pour 3 tailles d’écran :
  - 📱 **Petits écrans** (smartphones)
  - 📟 **Écrans moyens** (tablettes)
  - 💻 **Écrans larges** (ordinateurs)

Le Responsive Design

Les **points de rupture** sont des seuils de largeur d'écran qui déclenchent un changement dans la mise en page.



Les Media Queries

Définition & Objectif

Qu’est-ce que c’est ?

Une **Media Query** est une règle en CSS3 qui permet d’appliquer des blocs de styles CSS uniquement si une condition est remplie.

Cette condition est le plus souvent basée sur les caractéristiques de l’appareil, comme la largeur de la fenêtre du navigateur (*viewport*).

Quel est l’objectif ?

✅ **Adapter** le contenu et le style d’un site.

✅ **Cibler** des caractéristiques précises : taille d’écran, résolution, orientation (portrait/paysage)…

✅ **Améliorer** l’accessibilité et l’expérience utilisateur (UX) sur tous les appareils.

Les Media Queries

Quel est leur rôle ?

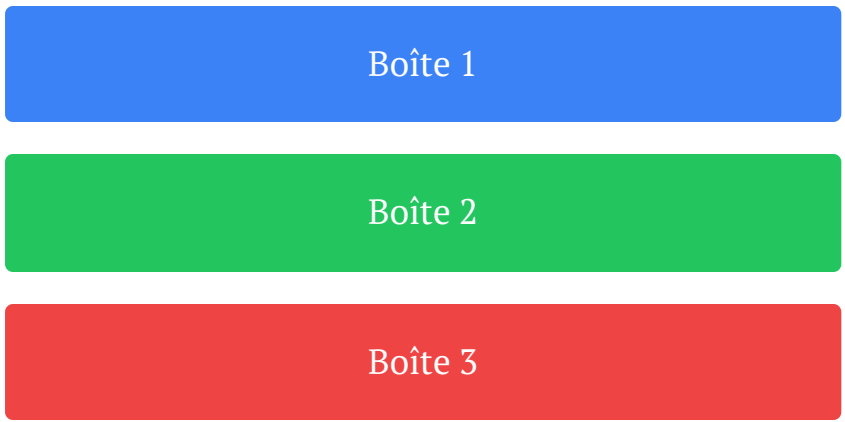
- Elles **détectent** les caractéristiques de l'appareil.
- Elles **appliquent** des styles de manière conditionnelle.
- Elles permettent de **transformer** une mise en page.

Par exemple, sur un petit écran, les éléments peuvent être empilés verticalement, tandis que sur un grand écran, ils seront affichés côte à côte.

Un même site, trois rendus différents grâce aux Media Queries.

Exemple Concret

Rendu Visuel



CSS

```css
.container {
  display: flex;
  /* Par défaut on set direction en colonne */
  flex-direction: column;
}

/* Si l'écran fait 768px ou plus... */
@media (min-width: 768px) {
  .container {
    flex-direction: row; /* ...on passe en ligne */
  }
}
```

Importance pour l’Accessibilité & l’UX

Les Media Queries ne sont pas qu’une question d’esthétique, elles sont essentielles.

♿ Accessibilité (A11Y)

- **Lisibilité optimale :** Garantit que le texte est lisible sur tous les appareils, sans besoin de zoomer.
- **Navigation aisée :** Assure que les boutons et les liens sont facilement cliquables, même sur un petit écran tactile.
- **Adaptation :** Permet de prendre en compte les préférences des utilisateurs, comme le mode sombre (`prefers-color-scheme`).

✨ Expérience Utilisateur (UX)

- **Confort visuel :** Évite les défilements horizontaux et les mises en page "cassées".
- **Cohérence :** Fournit une expérience familière et continue, qu’on passe du mobile à l’ordinateur.
- **Performance :** Permet de charger des images plus légères ou de masquer des éléments non essentiels sur mobile.

Historique Rapide

Des origines fixes à la flexibilité

Avant : Le Web Fixe 🖥️

- Les sites étaient conçus pour une seule résolution d'écran d'ordinateur.
- Les mises en page étaient rigides, souvent avec des largeurs fixes en pixels.
- La navigation sur mobile était une expérience frustrante, voire impossible.

Maintenant : Le Responsive Design 📱💻

- En 2010, **Ethan Marcotte** popularise le concept de "Responsive Web Design".
- Il le définit par trois piliers techniques :
  1. Grilles fluides (*Fluid Grids*)
  2. Images flexibles (*Flexible Images*)
  3. **Media Queries**

Historique Rapide

La Révolution CSS3

- Avec la spécification **CSS3**, les Media Queries deviennent un standard du W3C.
- Elles offrent aux développeurs un outil natif, puissant et précis pour créer des designs adaptatifs.
- Aujourd'hui, elles sont au cœur du développement front-end moderne.
- Elles sont la base de tous les frameworks CSS populaires (Bootstrap, Tailwind CSS, Bulma…) et des méthodologies comme le **Mobile First**.

Syntaxe

Syntaxe d’une Media Query

La structure d’une Media Query est simple et déclarative.

```
@media (condition) {
  /* Règles CSS à appliquer */
  /* si la condition est vraie */
}
```

- `@media` : Le mot-clé qui déclare le début d’un bloc de règles conditionnelles.
- `(condition)` : Le critère à évaluer. Par exemple, `(max-width: 600px)`.
- `{ ... }` : Le bloc contenant les règles CSS qui ne s’appliqueront que si la condition est remplie.

Types de Media Queries

Basées sur la largeur (Width)

C’est l’utilisation la plus courante. Elle permet d’adapter la mise en page pour les mobiles, tablettes et ordinateurs.

Les deux propriétés principales sont :

- `min-width` : Cible les écrans **à partir d’une** certaine largeur (`>=`).
- `max-width` : Cible les écrans **jusqu’à une** certaine largeur (`<=`).

Exemple : Styles pour les écrans de 600px et moins

```
/* Pour tous les écrans dont la largeur est de 600px ou moins... */
@media (max-width: 600px) {
  body {
    background-color: lightblue;
  }
  /* ...on applique un fond bleu. */
}
```

Types de Media Queries

Basées sur l'orientation

Idéal pour ajuster la disposition lorsque l'utilisateur bascule son appareil.

- `orientation: portrait` 📱
  Pour les écrans plus hauts que larges (ex: un smartphone tenu verticalement).
- `orientation: landscape` ↔️
  Pour les écrans plus larges que hauts (ex: une tablette à l'horizontale).

Exemple : Masquer la barre latérale en paysage

```
@media (orientation: landscape) {
  .sidebar {
    display: none;
  }
}
```

Autres Propriétés Notables

De nombreuses autres caractéristiques de l'appareil peuvent être ciblées :

- `width` / `height` : Dimensions du *viewport* (la zone d'affichage du navigateur).
- `device-width` / `device-height` : Dimensions de l'écran physique de l'appareil.
- `resolution` : Densité des pixels (ex: `150dpi`). Utile pour les écrans haute définition "Retina".
- `aspect-ratio` : Rapport largeur/hauteur du *viewport* (ex: `16/9`).
- `device-aspect-ratio` : Rapport largeur/hauteur de l'appareil.

> **Astuce :** La plupart de ces propriétés peuvent être préfixées par `min-` ou `max-` pour créer des intervalles (ex: `min-resolution`).

Combinaison de Conditions

Il est possible de cumuler plusieurs conditions pour un ciblage encore plus précis en utilisant des opérateurs logiques.

- `and` : Pour exiger que **toutes** les conditions soient vraies.
- `,` (virgule) : Agit comme un opérateur **OU** logique.
- `not` : Pour inverser une condition (cible tout ce qui ne correspond pas).

Exemple : Écrans d'au moins 600px ET en mode portrait

```
@media (min-width: 600px) and (orientation: portrait) {
  /* Ces styles s'appliquent si ET SEULEMENT SI */
  /* les deux conditions sont réunies. */
}
```

Combinaison de Conditions

Il est possible de cumuler plusieurs conditions pour un ciblage encore plus précis en utilisant des opérateurs logiques.

- `and` : Pour exiger que **toutes** les conditions soient vraies.
- `,` (virgule) : Agit comme un opérateur **OU** logique.
- `not` : Pour inverser une condition (cible tout ce qui ne correspond pas).

Exemple : Écrans d'au moins 600px ET en mode portrait

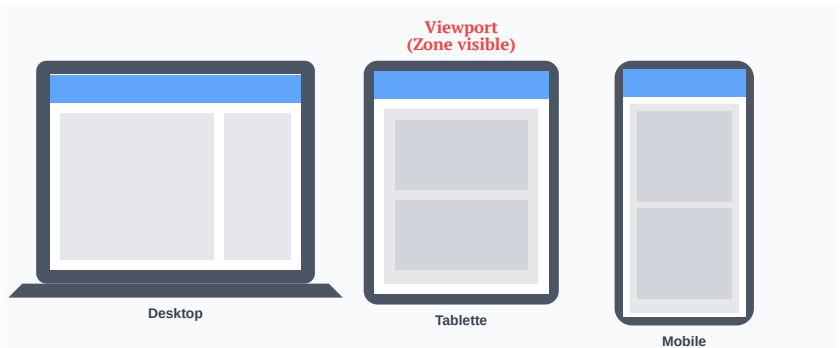
```
@media (min-width: 600px) and (orientation: portrait) {
  /* Ces styles s'appliquent si ET SEULEMENT SI */
  /* les deux conditions sont réunies. */
}
```

Le Viewport

La zone visible de la page

Le **viewport** est la surface visible d'une page web pour l'utilisateur. Sa taille et sa forme dépendent de l'appareil utilisé.

- C'est la "fenêtre" à travers laquelle on consulte le site.
- Sur ordinateur, c'est la taille de la fenêtre du navigateur.
- Sur mobile, c'est la taille de l'écran.
- Contrôler le viewport est la **première étape indispensable** pour un design responsive.



La Balise Meta Viewport

Le Contrôleur Essentiel

Pour s'assurer qu'un site s'affiche correctement sur mobile, on doit dire au navigateur comment se comporter. On utilise pour cela une balise `<meta>` à placer dans le `<head>` du document HTML.

C'est une instruction non-négociable pour tout site web moderne.

```
<!DOCTYPE html>
<html>
<head>
  <title>Mon Site Responsive</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
  ...
</body>
</html>
```

La Balise Meta Viewport

Analysons les deux instructions les plus importantes de cette balise.

```
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
```

- `width=device-width`
  Cette instruction force la largeur du viewport à être égale à la largeur de l'écran de l'appareil. Le navigateur n'essaiera plus de simuler un grand écran.

- `initial-scale=1.0`
  Ceci définit le niveau de zoom initial lorsque la page est chargée pour la première fois. Une valeur de `1` signifie "pas de zoom".

Comportement du Viewport

Le comportement par défaut du viewport varie grandement entre les appareils.

💻 Sur Ordinateur

- Le viewport correspond simplement à la taille de la fenêtre du navigateur.
- L'utilisateur peut le redimensionner à tout moment.
- Les Media Queries réagissent en temps réel à ce redimensionnement.

📱 Sur Mobile (Sans la balise meta)

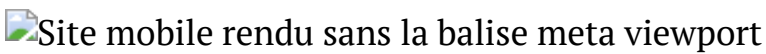
- Par défaut, le navigateur mobile ment ! Il prétend avoir un viewport large (souvent `980px`).
- Il affiche la version "ordinateur" du site.
- Puis, il effectue un **dézoom** pour que tout rentre dans le petit écran.
- Le résultat : un site illisible où il faut zoomer manuellement.

L’Impact de la Balise Viewport

La différence est radicale. L’oubli de cette balise est une erreur fréquente qui rend les Media Queries inefficaces.

Avant : Sans `<meta name="viewport">`

Le site est dézoomé et inutilisable. Les media queries pour petits écrans ne se déclenchent pas car le navigateur pense que l’écran est large.



Après : Avec `<meta name="viewport">`

Le navigateur utilise la vraie largeur de l’appareil. Le site s’affiche au bon niveau de zoom et les media queries peuvent appliquer les styles "mobiles".

Les Points de Rupture

Qu’est-ce qu’un Point de Rupture ?

Un **point de rupture** (ou *breakpoint*) est un seuil, défini en CSS, qui déclenche un changement de mise en page. C’est la pierre angulaire du responsive design.

- C’est la largeur d’écran précise où le design "casse" et a besoin d’être réorganisé.
- Il permet de passer d’une disposition à une autre (ex: de 1 à 2 colonnes).
- Techniquement, c’est la condition de largeur (`min-width` ou `max-width`) utilisée dans une Media Query.

```
/* À partir de 768px, le layout change. */
/* 768px est donc notre point de rupture. */
@media (min-width: 768px) {
  .container {
    flex-direction: row;
    justify-content: space-between;
  }
}
```

Pourquoi sont-ils Cruciaux ?

Sans points de rupture, un design fluide finit par atteindre ses limites sur de très grands ou très petits écrans.

❌ Sans Points de Rupture

- Le texte s'étire sur des lignes interminables.
- Les images deviennent trop petites ou trop grandes.
- Les espaces blancs sont mal gérés.
- La composition visuelle est perdue.

✅ Avec Points de Rupture

- La mise en page est réorganisée en colonnes.
- La taille de la police et des éléments est ajustée.
- La lisibilité et l'esthétique sont préservées.
- L'expérience utilisateur reste optimale.

Comment Définir ses Points de Rupture ?

L’approche "Content-First"

La meilleure pratique n’est **pas** de cibler des appareils spécifiques, mais de laisser le **contenu** dicter les points de rupture.

👎 Approche Déconseillée : "Device-First"

- **Principe :** Créer des breakpoints basés sur les tailles d’écrans populaires (ex: 375px pour l’iPhone, 768px pour l’iPad…).
- **Problème :** C’est une approche fragile. De nouveaux appareils sortent constamment. Le design n’est pas adapté au contenu réel.

👍 Approche Recommandée : "Content-First"

- **Principe :** Observer le design à différentes largeurs et ajouter un breakpoint uniquement lorsque la mise en page se dégrade.
- **Avantage :** Le design est plus robuste, durable et est véritablement au service du contenu qu’il présente.

Trouver le Point de Rupture Idéal

La méthode est simple, visuelle et itérative :

1. **Commencez petit :** Affichez votre site dans une fenêtre de navigateur très étroite, simulant un mobile.
2. **Élargissez progressivement :** Tirez lentement le bord de la fenêtre pour l'agrandir.
3. **Observez la "casse" :** Dès qu'un élément semble mal placé, que le texte devient difficile à lire, ou que l'espace est mal utilisé… **STOP !**
4. **Notez la largeur :** Regardez la largeur actuelle du navigateur (avec les outils de développement). Voilà votre nouveau point de rupture !
5. **Codez la solution :** Écrivez la Media Query pour cette largeur et réorganisez votre CSS pour corriger le problème.

Moins, c’est Mieux

Limiter le Nombre de Points de Rupture

Trop de points de rupture complexifient inutilement le code et sa maintenance. Visez la simplicité.

- ✅ **Maintenance simplifiée :** Moins de code à lire, à comprendre et à déboguer.
- ✅ **Tests plus rapides :** Moins d’états de page à vérifier sur différents appareils.
- ✅ **Design plus cohérent :** Encourage à penser en termes de "mises en page majeures" (mobile, tablette, bureau) plutôt qu’en micro-ajustements constants.
- ✅ **Meilleures performances :** Moins de règles CSS complexes à analyser pour le navigateur.

> **Règle d'or :** N'ajoutez un point de rupture que lorsque c'est absolument nécessaire pour la lisibilité et l'ergonomie.

Points de Rupture Courants

Un point de départ pragmatique

Même avec une approche "Content-First", il est utile de connaître les conventions issues des frameworks CSS comme Tailwind ou Bootstrap. Ils fournissent un excellent point de départ.

Voici une base typique (les valeurs peuvent varier) :

- `sm (small)` : **640px**
  Pour les petits appareils en mode paysage.
- `md (medium)` : **768px**
  Cible les tablettes en mode portrait. C'est souvent le premier breakpoint majeur.
- `lg (large)` : **1024px**
  Pour les tablettes en paysage et les petits ordinateurs portables.
- `xl (extra large)` : **1280px**
  Pour les écrans de bureau standards.

Approches de Conception

Desktop First vs. Mobile First

Le "Desktop First" et le "Mobile First" sont deux **stratégies de développement**. Elles dictent l'ordre dans lequel vous écrivez votre CSS. Ce sont des approches qui se concentrent sur un type d'appareil au départ.

🖥️ Desktop First (L'ancienne méthode)

On commence par styler la version pour grand écran. Ensuite, on utilise des Media Queries avec `max-width` pour "simplifier" ou "réarranger" le design pour des écrans plus petits.

Logique : "Si l'écran est *plus petit que* X, alors supprime cette colonne."

```css
/* Styles par défaut pour le bureau */
.container {
  display: flex;
}

/* On adapte pour les tablettes ET plus petit */
@media (max-width: 1024px) {
  /* ... */
}

/* On adapte pour les mobiles */
@media (max-width: 768px) {
  .container {
    flex-direction: column;
  }
}
```

📱 Mobile First (L’approche moderne)

On commence par styler la version pour mobile, la plus simple. Ensuite, on utilise des Media Queries avec `min-width` pour "enrichir" ou "complexifier" le design pour des écrans plus grands.

Logique : "Si l’écran est *au moins aussi grand que* X, alors ajoute cette colonne."

```css
/* Styles par défaut pour mobile */
.container {
  flex-direction: column;
}

/* On enrichit pour les tablettes ET plus grand */
@media (min-width: 768px) {
  .container {
    flex-direction: row;
  }
}

/* On enrichit pour le bureau */
@media (min-width: 1024px) {
  /* ... */
}
```

Quelle Approche Choisir ?

L'approche **Mobile First** est aujourd'hui considérée comme la meilleure pratique par la majorité des développeurs.

Raisons de la Recommandation

- 📈 **Trafic Majoritaire :** La plupart des utilisateurs naviguent sur mobile. Il est logique de les prioriser.
- ⚡ **Performance :** Les appareils mobiles chargent uniquement le CSS de base, plus léger. Le CSS additionnel pour les grands écrans n'est chargé que si nécessaire.
- 🧠 **Contraintes Positives :** Commencer par le plus petit écran force à se concentrer sur l'essentiel du contenu et des fonctionnalités.
- 🛠️ **Code plus simple :** Il est souvent plus facile d'ajouter de la complexité (passer de 1 à 3 colonnes) que d'en retirer. Le CSS est donc plus propre et plus facile à maintenir.

Mobile First vs. Content First

Deux Idées Complémentaires

Il ne faut pas les opposer, mais les combiner !

- Le **Mobile First** est une **stratégie de développement** : "Par où est-ce que je commence à écrire mon code CSS ?".
  ➡️ **Réponse :** Je commence par le mobile.

- Le **Content First** est une **philosophie de design** : "Qu'est-ce qui doit déclencher un changement de mise en page ?".
  ➡️ **Réponse :** C'est le contenu qui "casse", pas la taille d'un appareil spécifique.

> **La meilleure pratique est donc :** Adopter une stratégie **Mobile First**, où les points de rupture sont définis selon une approche **Content First**.

Les Images Responsives

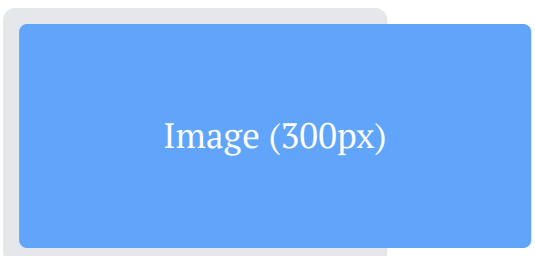
Le Problème Classique

L’image qui dépasse

Par défaut, une image a une taille fixe. Si son conteneur devient plus étroit que l’image, celle-ci déborde, créant un défilement horizontal très désagréable.

Avant : L’Image Rigide

Ici, le conteneur gris est plus petit que l’image bleue, qui sort donc de son cadre.



Les Conséquences

- 📜 **Ascenseur horizontal :** Le symptôme le plus évident et le plus frustrant.
- 📉 **Expérience utilisateur dégradée :** La navigation devient pénible, l’utilisateur doit dézoomer ou faire défiler pour voir le contenu.
- 🔧 **Mise en page "cassée" :** L’harmonie visuelle du site est complètement détruite.

La Solution Simple : Les Images Fluides

La solution de base est purement CSS et incroyablement efficace. Elle transforme les images rigides en images "fluides" qui s'adaptent à leur parent.

Le Code Magique

Ce simple bloc de CSS devrait être inclus dans la base de tous vos projets web.

```css
img {
  max-width: 100%;
  height: auto;
}
```

Comment ça marche ?

- `max-width: 100%`
  L'image ne pourra **jamais** être plus large que son élément parent. Si le parent rétrécit, l'image rétrécit avec lui.

- `height: auto`
  La hauteur de l'image s'ajuste automatiquement pour conserver les proportions d'origine. Cela évite que l'image ne soit déformée.

Les Limites de l’Image Fluide

La solution CSS est excellente, mais elle a des inconvénients dans certaines situations.

📉 Poids Inutile & Perte de Qualité

Si vous servez une très grande image (ex: 1920px de large) à un mobile, le navigateur la télécharge en entier puis la réduit.

- **Performance :** L’utilisateur mobile télécharge des données inutiles, ce qui ralentit la page.
- **Qualité :** La réduction par le navigateur peut parfois dégrader la netteté de l’image.

🖼️ Cadrage Inadapté (Direction Artistique)

Une image conçue pour un grand écran peut devenir illisible ou perdre son impact une fois réduite. Le sujet principal peut devenir trop petit.

La Solution Avancée : La Balise `<picture>`

Pour résoudre ces problèmes, HTML5 propose la balise `<picture>`. Elle permet de faire de la **direction artistique** : servir une image complètement différente en fonction du contexte (taille d'écran, etc.).

Pour les grands écrans (≥ 800px)

On affiche une image large et détaillée.

Pour les petits écrans

On affiche une image recadrée, plus percutante.

Anatomie de la Balise `<picture>`

La balise `<picture>` fonctionne comme un conteneur qui propose plusieurs sources au navigateur. Le navigateur choisit la première source qui correspond aux critères.

```
<picture>
  <source srcset="images/policier-large.jpg" media="(min-width: 800px)">
  <source srcset="images/policier.jpg">
  <img src="images/policier.jpg" alt="Un policier regardant au loin">
</picture>
```

- `<source>` : Propose une version de l'image. Il peut y en avoir plusieurs.
  - `srcset` : (Obligatoire) Spécifie le chemin vers le fichier image.
  - `media` : (Optionnel) Une Media Query qui agit comme condition. Si la condition est vraie, cette source est choisie.
- `<img>` : (Obligatoire) C'est l'élément de **repli** (fallback). Il s'affiche si aucune condition `<source>` n'est remplie ou sur les vieux navigateurs. C'est aussi lui qui porte le texte alternatif `alt` pour l'accessibilité.

Les Avantages de `<picture>`

Utiliser la balise `<picture>` est une pratique avancée avec des bénéfices concrets.

- ⚡ **Performance Optimale**
  Le navigateur ne télécharge **que l'image dont il a besoin**. Un utilisateur mobile ne téléchargera jamais l'image lourde destinée aux grands écrans. C'est un gain majeur pour la vitesse de chargement.

- 🎨 **Contrôle Artistique Total**
  Vous garantissez que la meilleure version de l'image est affichée dans chaque contexte. C'est crucial pour les photos de produits, les graphiques contenant du texte ou les logos.

- ✅ **Accessibilité Préservée**
  Comme l'élément `<img>` est obligatoire, le texte alternatif est toujours présent, garantissant que le contenu reste accessible à tous.

Conclusion

Ce qu’il faut retenir

Voici les concepts clés à maîtriser pour un design responsive efficace :

- 1️⃣ **Le Viewport est la base :** Sans la balise `<meta name="viewport">`, vos media queries seront inefficaces sur mobile. C’est la toute première étape.
- ⚙️ **Les Media Queries sont conditionnelles :** Elles appliquent des styles CSS uniquement si une condition (`min-width`, `orientation`...) est remplie.
- 📱 **Pensez Mobile First :** Concevez d’abord pour le plus petit écran et enrichissez la mise en page pour les plus grands avec `@media (min-width: ...)`. C’est plus simple et plus performant.
- 🔍 **Laissez le contenu dicter les breakpoints :** N’ajoutez un point de rupture que lorsque le design "casse", et non pour cibler un appareil spécifique.
- 🖼️ **Rendez vos images fluides et intelligentes :** Utilisez `max-width: 100%` sur les images comme base, et la balise `<picture>` pour optimiser la performance et la direction artistique.

Questions ?