

# Responsive Webdesign – présent et futur de l'adaptation mobile

---

 [www.alsacreations.com/article/lire/1559-responsive-web-design-present-futur-adaptation-mobile.html](http://www.alsacreations.com/article/lire/1559-responsive-web-design-present-futur-adaptation-mobile.html)

La problématique du responsive est bien plus complexe que ce qu'on peut lire sur certains blogs qui essaient de nous faire croire qu'il suffit aujourd'hui, pour optimiser un site pour mobile, d'ajouter 2 Media Queries pour l'iPhone et l'iPad et redimensionner toutes les images. Le Responsive Web Design est une technique toute jeune, loin d'être parfaite et en constante évolution. Beaucoup de choses sont aujourd'hui possible, mais hélas il reste encore pas mal de chemin à parcourir dans le domaine.

Après plusieurs mois de recherche sur le sujet, j'ai enfin publié mon article sur [Smashing Magazine](#) qui s'intitule "[The State Of Responsive Web Design](#)". Ce qui suit en est la traduction.

Avertissement avant la lecture : Je n'ai pas la prétention de changer le monde, d'avoir la vérité absolue. Dans cet article – qui est long, je le sais – je souhaitais juste rendre attentif le lecteur au reste de ce gigantesque iceberg dont les Media Queries n'en sont que la surface. **Le responsive webdesign reste une technique et une infime partie de ce qui est aujourd'hui appelé "[Adaptive Webdesign](#)".** Le but de l'article n'est pas non plus de décourager les gens qui optimisent des sites pour mobile, mais de **mettre le doigt sur ce qui aujourd'hui pose problème, est bancal, pour ensemble, trouver des solutions à ces différents problèmes.** Pour un peu plus de [détails sur le pourquoi de l'article, je vous renvoie à mon blog](#).

On entend parler de "Responsive Webdesign" depuis plusieurs années maintenant, et le sujet s'est vraiment démocratisé et popularisé en 2012. [Brad Frost](#), [Luke Wroblewski](#) et d'autres grands noms commencent à avoir une certaine expérience du sujet, et aident au quotidien par leurs publications à son amélioration. Cependant, beaucoup reste encore à faire.



Dans cet article, nous allons nous intéresser à ce qui est **aujourd'hui** déjà possible en terme d'optimisation de sites pour mobiles, mais également à ce qui sera possible dans le **futur**. Nous parlerons de propriétés non encore standardisées comme CSS level 4, HTML5 et d'APIs, ainsi que de techniques qui restent à être améliorées. Cet article est loin d'être exhaustif et nous n'aurons pas le temps d'approfondir chaque technique mentionnée, mais vous aurez à la fin de la lecture à votre disposition suffisamment de liens pour pouvoir continuer l'exploration par vous-même.







## L'état des images dans le responsive webdesign

Quelle meilleure manière d'aborder le sujet que de commencer par ce qui pose un gros souci : la gestion des images. Ce sujet n'est pas nouveau, il prend de plus en plus d'ampleur avec l'arrivée d'écrans dits "haute définition". Par ce terme j'entends des écrans avec un *ratio* de pixel supérieur à 2; ils sont appelés **Retina** chez Apple (iOS) ou encore **XHDPI** chez Google (Android). Quand on commence à s'intéresser aux images dans le domaine du responsive, on se retrouve confronté à deux difficultés : la taille (entendez par là leur poids en octets) et la performance.

Le site [screensiz.es](http://screensiz.es) fait l'inventaire de la densité de pixels

Beaucoup de designers adorent le "pixel-perfect", mais les images de taille "normale" sont souvent "pixelisées" lorsqu'elles sont affichées sur des écrans haute définition. L'idée plutôt simpliste de proposer des images qui auraient deux fois (voire plus) la taille de départ pour éviter la dégradation visuelle sur ce

genre

 SCREEN SIZES				
   				
PHONE 	W PX	H PX	DEVICE-W PX	PX DENSITY
Samsung Nexus S	480	800	320	150% HDPI
Nokia Lumia 920	768	1280	384	200% XHDPI
Nokia Lumia 900	480	800	320	150% HDPI

d'appareil semble bien tentante. Mais qu'en est-il des problématiques de performance ? Des images dont les dimensions seraient doublées verraient leur poids multiplié mathématiquement par 4, et mettraient plus de temps à charger. Même si vos utilisateurs ont un écran haute définition, ils n'ont pas forcément, à l'instant où ils consultent votre page, la connexion qui va avec pour télécharger des images plus lourdes. En fonction du pays où se trouvent vos utilisateurs, la bande passante est d'ailleurs plus ou moins coûteuse.

Le second problème est lié aux petits appareils : pourquoi un téléphone mobile devrait avoir à télécharger une image de 750 pixels de large, alors qu'il n'a que 320 pixels physiques sur son écran pour l'afficher ? Et comment faire pour recadrer ces petites images, pour aller à l'essentiel et mettre en avant seulement la partie que l'on juge importante ?

## Deux solutions du côté du code : l'élément `<picture>` et l'attribut `srcset`

On pourrait trouver dans le code que nous utilisons quotidiennement pour afficher les images un début de solution au sujet épineux des images *responsive*.

Le [Responsive Images Community Group](#) soutient une proposition d'un nouvel élément HTML `<picture>` plus flexible que ce que l'on utilise pour le moment. L'idée est d'utiliser la syntaxe des [Media Queries](#) que l'on commence à bien connaître pour faire télécharger des images différentes en fonction de l'appareil, de sorte que des petits écrans récupéreraient des images plus petites. La syntaxe de l'élément est proche de celle de `<video>`, chaque image étant référencée dans un sous élément `<source>`.

Voici à quoi ressemble le code de cette proposition :

```
<picture width="500" height="500">
  <source media="(min-width: 45em)"
src="large.jpg">
  <source media="(min-width: 18em)" src="med.jpg">
  <source src="small.jpg">
  
  <p>Texte accessible</p>
</picture>
```

On imagine aisément qu'il serait ainsi facile de proposer la même image en différentes tailles, mais on peut aussi imaginer proposer plusieurs images pour mettre l'accent sur différents éléments en fonction de la taille de l'appareil. Le *use-case* “[Art Direction](#)” du W3C illustre en quelques exemples de ce qu'il serait alors possible de réaliser.



image large large.jpg



image moyenne med.jpg



petite image small.jpg

Cette proposition est en cours de discussion auprès du [W3C Responsive Images Community Group](#) mais n'est à l'heure actuelle utilisable dans aucun navigateur. Un *polyfill* appelé [Picturefill](#) est disponible et permet d'émuler `<picture>` en JavaScript à l'aide de `<div>` et de `data-attribute` pour éviter les conflits de syntaxe le jour où `<picture>` sera standardisé.

En parallèle, Apple a également fait une proposition pour la syntaxe d'images responsives au W3C : [The srcset Attribute](#). Elle est l'équivalent en HTML de la proposition CSS Level 4 [image-set\(\)](#). D'après les spécifications, le but de cet attribut est de forcer le navigateur (User Agent) à charger la ressource appropriée, plutôt que de lui proposer une liste de différentes ressources. Il ne s'agit donc ici non pas d'une nouvelle balise, mais de conserver `<img>` et de lui ajouter un nouvel attribut `srcset`. Voici la syntaxe décrite dans les spécifications :

```

```

Vous l'aurez remarqué, elle est loin d'être intuitive. Les valeurs sont données dans des chaînes de caractère séparées par des virgules. A l'intérieur de ces chaînes on retrouve le chemin vers l'image, une valeur de densité de pixels de l'appareil, et la taille maximum de viewport pour laquelle cette image devrait être utilisée.

En français dans le texte, voici ce que donne le code ci-dessus :

- Par défaut l'image utilisée est `banner.jpeg`
- Les appareils qui ont un ratio de pixel supérieur à 2 utiliseront `banner-HD.jpeg`
- Les appareils avec un viewport inférieur à 640px utiliseront `banner-phone.jpeg`
- Les appareils avec un viewport inférieur à 640px ET un ratio de pixel supérieur à 2 utiliseront `banner-phone-HD.jpeg`

La première source donnée dans `src` est l'image utilisée par défaut si cet attribut n'est pas supporté. Le suffixe `2x` pour `banner-HD.jpeg` signifie que cette image doit être utilisée pour les écrans avec un ratio de pixel supérieur à 2. La valeur `640w` pour `banner-phone.jpeg` représente la taille maximum du [viewport](#) pour laquelle cette image sera utilisée. En raison de la complexité technique de sa mise en place, cette syntaxe n'a pas encore été implémentée dans nos navigateurs.

La syntaxe de `image-set()` en CSS fonctionne de la même manière et permet de charger des images de fond qui vont dépendre des mêmes critères :

```
background-image: image-set( "foo.png"
1x,
"foo-2x.png" 2x,
"foo-print.png" 600dpi );
```

La proposition pour la version CSS est pour le moment dans l'état Editor's Draft au W3C et fonctionne dans Safari 6+ and Chrome 21+. Un [article sur picture, src-set et image-set](#) est disponible en français sur le blog Creative Juiz.

## Format d'image, compression et SVG : changer la façon dont on travail avec les images sur le web

Vous aurez constaté que ces deux propositions de syntaxe pour les images sont encore au stade expérimental. Ce qui pose la problématique du format d'image en tant que tel : un début de solution serait-il à trouver de côté là ?

La première étape serait de jeter un œil du côté des formats alternatifs d'image qui proposent un meilleur taux de compression. Google par exemple a développé de son côté un nouveau format, le [WebP](#) qui propose des compressions plus importantes pour des images 26% plus petites que le PNG et 25 à 34% plus petites que le JPEG. Ce format est supporté dans Chrome, Opera, Yandex, le navigateur natif Android et Safari. Il peut être activé pour Internet Explorer en utilisant le plugin [Google Chrome Frame](#) .

Cependant, Firefox ne prévoit pas d'implémenter ce format. En sachant cela, il est donc peu probable de voir (actuellement) le WebP être massivement utilisé.

Une autre idée qui gagne de plus en plus en popularité est l'utilisation d'images en JPEG progressif. Comme son nom le suggère, l'image est affichée progressivement; le premier rendu est quelque peu flou, puis l'image s'affine et devient plus nette au fur et à mesure de son téléchargement. Par comparaison, les fichiers JPEG non progressifs s'affichent de haut en bas. Ann Robson met en avant dans son article [Progressive JPEGs: A New Best Practice](#) le fait que cette compression d'image donne à l'utilisateur une plus grande impression de vitesse que les JPEG classiques. Elle permet à l'utilisateur d'avoir un premier aperçu global de l'image sans avoir à attendre qu'elle soit totalement chargée. Cette technique ne résout en rien les problèmes techniques de performance et poids de chargement des images, mais améliore, d'après son auteur, l'expérience utilisateur.

Une dernière solution (du moins dans cet article) proposée pour tenter d'améliorer l'optimisation des images est de jouer sur leur taux de compression. Pendant longtemps beaucoup pensaient qu'à trop augmenter le taux de compression d'une image, on perdait (trop) en qualité. Daan Jobsis a fait des recherches plus approfondies sur le sujet et partage ses trouvailles dans son article [Retina Revolution](#). Dans cette expérience, il a testé différentes tailles d'image et taux de compression pour arriver à une solution intéressante : en doublant la taille des images de départ tout en utilisant un taux de compression très élevé, l'image finale a un poids plus petit que l'originale et restera nette sur les écrans "normaux" et haute définition. Avec cette technique, il a été capable de réduire à 75% du poids des images qu'il a testé.

Vu le casse-tête que représente l'optimisation des images pour un site responsive, devenir indépendant des pixels absolus est une idée qui séduit de plus en plus les designers et intégrateurs. Le [format SVG](#) propose par exemple une excellente alternative, et permet de créer des éléments d'interface vectoriels qui vont s'adapter à n'importe quelle résolution d'écran. Les éléments créés en SVG vont être facilement réduits et agrandis sans perte pour les écrans à haute résolution. À cette tendance s'ajoute celle des [font-icons](#) qui permet de remplacer les glyphes d'une police d'écriture par des icônes, tout en conservant la flexibilité de la police.

Ces solutions n'étant cependant pas envisageables pour des photos, un meilleur format d'image, ou une syntaxe plus flexibles sont donc attendus avec impatience pour le futur.

## Défis de mise en page : réarranger le contenu sans toucher au HTML c'est possible ?

Pour commencer, arrêtons de nous voiler la face plus longtemps : les grilles fluides construites en flottants et éléments en [inline-block](#) que l'on utilise aujourd'hui ne sont que des patchs plus ou moins solides, dans l'attente d'une meilleure solution. Jouer avec la mise en page et totalement réarranger les blocs d'une page pour un écran plus petit sans recourir au JavaScript est à l'heure actuelle cauchemardesque. Toutes les solutions à notre disposition manquent cruellement de flexibilité. Et c'est encore plus vrai quand on commence à travailler avec des sites créés sous un CMS : il est impensable de devoir modifier le HTML pour chaque taille d'écran, alors comment peut-on améliorer tout ça ?

## Une mise en page plus flexible grâce à 4 solutions utilisant des propriétés CSS3

### La solution flexbox

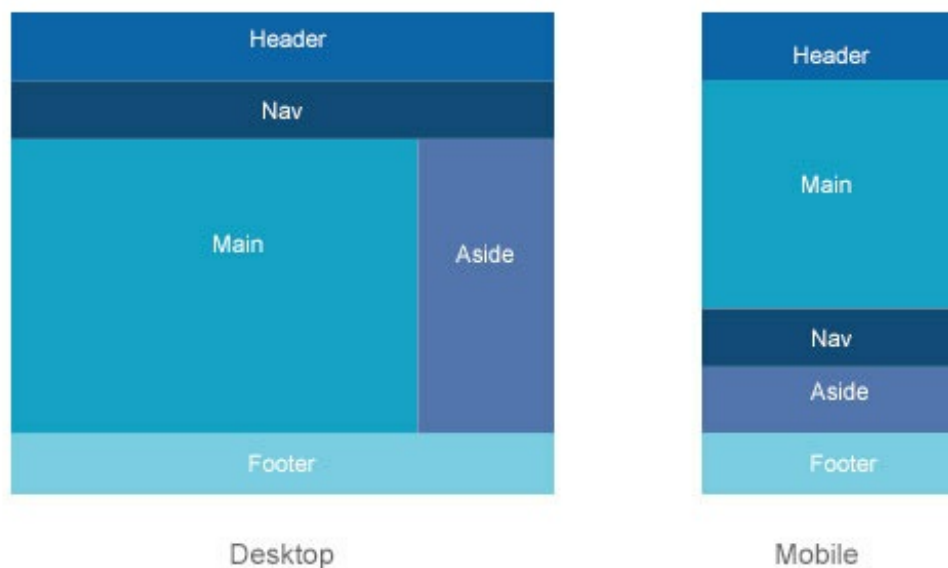
La première solution qui nous vient à l'esprit lorsque l'on pense au concept de mise en page est le module [CSS3 flexible box layout model](#) aussi connu sous le nom de *flexbox*. Pour le moment il a le statut de "Candidate Recommendation" au W3C mais est déjà supporté [par la plupart des navigateurs récents](#) (y



compris à partir de IE10). Le support des navigateurs mobiles est d'ailleurs particulièrement bon pour ce module. Ce modèle de boîte permet de réordonner les éléments à l'écran de manière simple et totalement indépendante de leur placement dans le code HTML. Il est également possible de changer l'orientation des boîtes (*box orientation*), la façon dont elles se suivent (*box flow*) et distribuer l'espace restant et les alignements en fonction du contexte sur la page.

L'image ci-dessous présente ce qui peut être accompli pour une mise en page sur mobile à partir de la version desktop (navigateur classique de "bureau"). Et la syntaxe pour y parvenir ressemblerait à quelque chose comme ceci :

```
.parent {  
  display: flex;  
  flex-flow: column; /* afficher les éléments en colonne  
*/  
}  
.children {  
  order: 1; /* modifie l'ordre des éléments */  
}
```



## La solution Multi Column Layout

Une seconde technique CSS3 qui est utilisable aujourd'hui pour une mise en page *responsive* est le [CSS3 Multi Column Layout](#). Ce module est passé en "Candidate Recommendation" et fonctionne pour la majorité des navigateurs récents, à l'exception de IE9 et versions inférieures. L'atout principal de ce module est la possibilité d'avoir du contenu qui va automatiquement glisser d'une colonne à une autre de manière très flexible en fonction de la taille du parent. Pour une optimisation mobile, il est donc très facile de changer la taille de la police en fonction de la taille du viewport.

Il est possible de donner une taille fixe à chaque colonne, et laisser au navigateur le soin de calculer le nombre de colonnes qui vont pouvoir rentrer dans l'espace disponible. Il est également possible de faire l'inverse : fixer un nombre de colonnes prédéfini, avec ou sans espace entre et laisser au navigateur le soin de calculer la largeur optimale pour chacune d'entre elles.

Powder tiramisu cake. Soufflé cheesecake cupcake soufflé sugar plum. Soufflé carrot cake danish powder cupcake. Pie lollipop apple pie caramels pastry dessert. Caramels cotton candy chupa chups chocolate cake wafer	chupa chups chocolate cake wafer cookie liquorice cupcake. Jujubes toffee muffin jelly pudding. Oat cake applicake jelly beans caramels ice cream. Marzipan marzipan faworki danish candy. Lollipop wafer danish biscuit.
--	---

```
.content {columns: 2;}
```

Powder tiramisu cake. Soufflé cheesecake cupcake soufflé sugar plum. Soufflé carrot cake danish powder cupcake. Pie lollipop apple pie caramels pastry dessert. Caramels cotton candy chupa chups chocolate cake wafer cookie liquorice cupcake. Jujubes toffee muffin jelly pudding. Oat cake applicake jelly beans caramels ice cream. Marzipan marzipan faworki danish candy. Lollipop wafer danish biscuit.

```
.content {columns: 1;}
```

Voici à quoi ressemble la syntaxe :

```
.container {
  column-width: 10em ; /* le navigateur crée des colonnes de 10em, le nombre
dépend de l'espace disponible */
}
.container {
  columns: 5; /* Le navigateur crée 5 colonnes dont la taille dépend de
l'espace disponible */
  column-gap: 2em;
}
```

Pour en savoir plus sur ce module je vous conseille l'excellent article de David Walsh : [CSS Columns](#) et retrouvez également un article en français sur Alsacreations : [Les multicolonnés en CSS3](#).

## La solution Grid Layout

Une troisième propriété qui mériterait plus d'attention dans le futur est le module [CSS3 grid layout](#) qui propose aux designers et développeurs un système de grilles flexibles avec lesquelles ils peuvent travailler pour construire différentes mises en page. Les éléments peuvent être affichés en colonnes et en lignes sans avoir besoin de définir une structure au préalable. La première chose à faire est de déclarer une grille sur le parent, puis de placer les éléments enfants sur cette grille "virtuelle". Il est ensuite très facile de changer la grille pour des écrans plus petits, ou de changer la position de ces éléments sur la grille. Combiné à des Media Queries, ceci permettrait une gigantesque flexibilité pour l'optimisation mobile mais également les changements d'orientation.

La syntaxe du Working Draft est la suivante (à la date du 2 avril 2013) :



```

.parent {
  display: grid; /* déclarer la grille */
  grid-definition-columns: 1stgridsize 2ndgridsize
  ...;
  grid-definition-rows: 1strowsize 2ndrowsize ...;
}
.element {
  grid-column: 1;
  grid-row: 1
}
.element2 {
  grid-column: 1;
  grid-row: 3;
}

```

Comme précisé dans les spécifications, plusieurs unités sont utilisables pour exprimer la taille des lignes et colonnes de cette grille virtuelle. Pour placer les éléments sur la grille on va spécifier le numéro de colonne (`grid-column`) dans lequel on veut le placer, puis le numéro de la ligne (`grid-row`). Si l'élément s'étend sur plusieurs lignes, il est possible d'utiliser la valeur `span` pour préciser le nombre de lignes à étendre.

## La solution Template layout

La dernière technique de mise en page qui pourrait devenir intéressante dans le futur si un navigateur se décidait à l'implémenter est le module [CSS3 template layout](#). Il permet d'associer un élément avec une zone de mise en page que l'on aura au préalable pris la peine de nommer, puis d'organiser visuellement ces éléments sur une grille invisible, encore une fois. Cette grille peut être soit fixe, soit flexible en fonction de la taille du viewport.

Voici ce à quoi ressemble la syntaxe pour le moment :

```

.parent {
  display: "ab"
  "cd" /* créer la grille invisible
  */
}
.child1 {
  position: a;
}
.child2 {
  position: b;
}
.child3 {
  position: c;
}
.child4 {
  position: d;
}

```

Ce qui visuellement donnerait ceci :

Hélas, comme vous l'aurez deviné à mon introduction, le support de cette propriété est pour le moment inexistante. Si des designers, développeurs et intégrateurs montrent de l'intérêt pour cette syntaxe, peut-être sera-elle un jour implémentée (avec des préfixes navigateurs ?), mais en attendant vous pouvez toujours, si elle vous intéresse, la tester avec un [polyfill](#).

## De nouvelles unités relatives au viewport et la fin de la mise en page basée sur des pixels ?

Les unités comme `vw`, `vh`, `vm`, `vmin` et `vmax` sont proposées dans la partie "[Viewport-based percentage lengths](#)" des spécifications et sont des unités de mesure relatives à la dimension du viewport.

Un `vw` est par exemple égal à 1% de la largeur du conteneur initial. Si la largeur du viewport est de 320px, `1vw` est égal à  $1 \times 320 / 100$  soit 3,2pixels.

L'unité `vh` fonctionne de la même manière mais est relative à la hauteur du viewport. `50vh` sont donc équivalents à 50% de la hauteur du document. Vous vous demandez sans doute maintenant ce qu'est la différence avec le fait d'utiliser directement une hauteur en pourcentage. Les unités en pourcentage sont relatives à la taille de leur parents, alors que `vh` et `vw` seront toujours relatives à la taille du viewport, quelle que soit la taille de leurs éléments HTML parents.

Cela devient particulièrement intéressant si vous souhaitez par exemple créer une boîte et être sûr qu'elle ne pourra jamais passer sous la hauteur du viewport (la ligne de flottaison), pour faire en sorte que l'utilisateur n'ait jamais besoin de faire défiler le contenu pour la voir. Cela permettrait également en théorie de créer des mises en page avec des éléments qui feraient 100% de la taille du navigateur mobile (donc viewport) sans avoir besoin de passer par des *hacks* de 100% sur tous leurs parents.

L'unité `vmin` est quant à elle égale à la plus petite d'entre `vm` ou `vh`, alors que `vmax` est égale à la plus grande. Ces unités sont donc flexibles aux changements d'orientation de l'appareil.

Hélas pour le moment ces unités ne sont **pas** supportées pour les mobiles sur les [navigateurs Android natifs](#). Il va donc peut-être falloir attendre un peu avant de pouvoir les utiliser correctement.

Retrouvez [quelques tests de ces unités](#) (sur des éléments textuels et un bloc flexible) en français sur Creative Juiz.

## À propos d'une typographie qui s'adapte

Au final, la mise en page d'un site reste très dépendante du contenu que l'on va y mettre. Je ne pouvais donc pas conclure une section dédiée aux possibilités de mise en page sans vous parler de typographie. CSS3 propose une nouvelle unité particulièrement utile pour la gestion de la typographie : [l'unité rem](#). Là où la taille d'une police exprimée en `em` est relative à la taille de la police de ses ancêtres, la taille d'une police exprimée en `rem` est, et sera toujours relative à la taille de la police de l'élément racine (à savoir `<html>`). Pour un site responsive, il est donc possible d'écrire le code suivant, puis d'adapter la taille de toutes les polices du site d'un coup en changeant la taille de police de l'élément html :

```

html {
  font-size: 14px;
}
p {
  font-size: 1rem /* 14px */
}
@media screen and (max-width:380px) {
  html {
    font-size: 12px; /* rendre les polices plus petites pour mobile */
  }
  p {
    font-size: 1rem /* cet élément est désormais équivalent à 12px */
  }
}

```

## Trouver un meilleur moyen de gérer d'autres types de contenus complexes

Nous commençons tout doucement à trouver de meilleures solutions pour gérer les images et les mises en page responsive, mais il faut encore trouver d'autres solutions pour les contenus plus complexes.

## Gérer les formulaires sur un site responsive

De manière générale la gestion de formulaire, et tout particulièrement de longs formulaires, est un défi en "mobilité" (entendez par là "dans l'univers du mobile"). Plus long sera le formulaire, plus il sera compliqué à adapter pour des petits appareils. L'adaptation physique visuelle n'est pas bien compliquée ; la plupart des designers vont simplement placer les éléments les uns sous les autres en une seule colonne et élargir les champs pour qu'ils prennent toute la taille de l'écran. Mais rendre ces formulaires visuellement attractifs n'est pas suffisant : il nous faut également les rendre facile d'utilisation sur mobile.

[Luke Wroblewski](#) conseille d'éviter les champs texte quand c'est possible, et de proposer des cases à cocher, boutons radios et menus déroulants de sélection de sorte que l'utilisateur ait à entrer le moins d'informations possible au clavier. Un autre conseil important est de proposer à l'utilisateur les retours quant à la validité des champs avant qu'il appuie sur le bouton "envoyer". Vérifier les erreurs au fur et à mesure de la saisie est important sur mobile, car la plupart des formulaires dépassent la hauteur de l'écran. Si l'utilisateur a mal rempli un champ et doit attendre l'envoi pour le savoir, il y a de fortes chances qu'il ait du mal à retrouver où se trouve le(s) champ(s) mal rempli(s).

Les nouveaux types de champs et attributs HTML5 vont être d'une grande aide dans le futur pour construire des formulaires faciles d'utilisation, sans besoins d'avoir recours à (trop) de JavaScript. Il est par exemple possible d'utiliser l'attribut HTML [required](#) pour proposer un retour à l'utilisateur sur un champs qu'il doit obligatoirement remplir. Hélas [le support mobile](#) de cet attribut est pour le moment très limité.

L'attribut [autocomplete](#) peut également être utilisé pour améliorer l'expérience utilisateur lors de la saisie des champs. Dans la mesure où un appareil mobile (de type smartphone) est un objet personnel, on peut imaginer que des données comme le nom, ou encore l'adresse postale ne changeront pas souvent pour son propriétaire. En utilisant l'attribut [autocomplete](#), il serait donc possible d'améliorer la saisie de ces champs pour que l'utilisateur n'ait pas besoin de taper encore une fois en entier ce genre de données.

Il existe également une liste entière de [nouveaux types d'inputs HTML5](#) qui pourront être utilisés dans un futur plus ou moins proche pour aider à l'optimisation des formulaires.

Le cas des dates est un bon exemple de ce qui peut être amélioré à l'aide de HTML5. Nous avons longtemps mis en place des *datepickers* (calendriers) en JavaScript, facilement utilisables sur navigateur de bureau, mais très difficiles à manier pour un appareil tactile. Appuyer sur la bonne date avec un doigt peut s'avérer assez difficile voire impossible si les zones sont très petites.

*Comment suis-je supposée arriver à ne toucher qu'une date quand la zone est si petite ?*

Les nouveaux types de champs HTML5 `input type="date"`, `input type="datetime"`, etc qui permettent de créer une chaîne de caractère au format de date/heure offrent des possibilités prometteuses ici. Le gros avantage de ces nouveaux types de champs est que leur rendu visuel est contrôlé par le navigateur et le système. De cette manière, leur UI (apparence graphique pour l'utilisateur) est automatiquement optimisée au mieux pour le périphérique utilisé.

*Rendu de `input type="date"` dans différents navigateurs*

Notons également que l'UI s'adapte également à la langue du navigateur. Utiliser des composants natifs permet donc également une adaptation linguistique automatique de certains contenus.

Pour le moment le support de `input type="date"` est presque absent pour les navigateurs de bureau, à part Opera et Chrome. Les navigateurs natifs Android ne le supportent pas non plus, par contre Chrome pour Android et Safari sur iOS le font. Il va donc falloir encore attendre un peu avant de pouvoir utiliser totalement le potentiel de ces nouveaux champs pour une solution mobile. Si vous êtes impatients, vous pouvez cependant utiliser un polyfill comme [Mobiscroll](#) qui va imiter le comportement natif de ces champs en JavaScript.

Au-delà des solutions natives de champs HTML5 d'autres expériences ont été menées pour proposer des patterns de designs plus adaptées comme "[Mobile Design Details: Hide/Show Passwords](#)" ou encore "[Input Masks Design](#)". Notons ne qu'il s'agit encore une fois que de solutions expérimentales et que là encore, la solution parfaite pour gérer les formulaires en responsive webdesign n'existe pas pour le moment. Il reste encore beaucoup de chemin à parcourir de ce côté.

Consultez également l'article [les éléments de formulaire HTML5](#) sur Alsacrérations pour aller plus loin.

## **La gestion des tableaux sur un site responsive.**

Les tableaux sont un autre type de contenu qui reste difficilement gérable. La plupart des tableaux ont une orientation verticale pour présenter le plus de données possibles, il est donc difficile de faire rentrer toutes ces données sur un petit écran. Les tableaux HTML sont plutôt flexibles, il est possible d'utiliser des pourcentages pour la largeur des colonnes. Cependant le contenu en devient très vite illisible.

Personne n'a pour le moment trouvé la solution idéale pour gérer les tableaux sur un site *responsive*, mais quelques suggestions intéressantes ont été faites.

Une première approche consisterait à cacher ce qui pourrait être considéré comme des colonnes "moins importantes", et proposer un système de checkboxes à l'utilisateur pour qu'il puisse choisir lesquelles il souhaite afficher. Toutes les colonnes seraient affichées sur un navigateur de bureau, alors que sur mobile, leur nombre dépendrait de la taille disponible à l'écran. Le Filament Group [explique et propose une démonstration de cette approche dans un de leurs articles](#). C'est également cette solution qui est proposé dans le module [table column toggle de jQuery Mobile](#).

Une seconde approche est basée sur l'idée de tableaux qui défilent. Elle consiste à fixer une seule colonne à gauche, et laisser une barre de défilement sur une petite partie du tableau à droite pour que

l'utilisateur puisse afficher le reste des données. [David Bushell implémente cette idée](#) dans un article en utilisant du CSS pour afficher tout le contenu du `<thead>` sur la partie gauche du tableau, laisse l'utilisateur faire défiler le reste sur la partie de droite. [Zurb utilise la même idée dans leur plugin](#) mais au lieu de coller le header à gauche, c'est la première colonne qui est dupliquée en JavaScript et le reste du tableau passe en dessous de sorte que l'utilisateur voit à gauche une colonne fixe, et à droite le reste des colonnes qu'il peut encore une fois faire défiler.

### *Deux exemples de tableaux que l'on peut faire défiler*

Le gros problème avec la propriété CSS `overflow:auto` est que la barre de défilement créée n'est pas visible sur certains appareils mobiles et tablettes. L'utilisateur peut toujours faire défiler la partie de droite, mais il n'a aucune indication visuelle qu'il peut s'y cacher plus de contenu que ce qu'il voit. Il faut donc, si on utilise ces techniques, trouver un moyen d'indiquer visuellement à l'utilisateur qu'il peut faire défiler le contenu à droite.

Une troisième approche est de ré-agencer les tableaux et de découper les colonnes en ce qui ressemble à des listes avec des titres. Cette technique est utilisée pour le ["reflow mode"](#) de jQuery mobile et est expliquée par Chris Coyier dans son article ["Responsive Data Tables."](#)

Il existe encore [plusieurs autres techniques](#) et l'utilisation de l'une ou de l'autre dépendra de votre projet. Aucun projet n'est identique, donc je peux seulement vous montrer ici ce que d'autres personnes ont créé pour contourner le problème des tableaux. Si vous avez vous aussi une solution à proposer, partagez-la avec tout le monde, dans les commentaires, sur Twitter, Github ou n'importe où. Nous sommes dans cette galère ensemble, les tableaux en mobilité sont une plaie, à nous de les améliorer !

## **Encapsuler des contenus tiers : le problème des iframes**

Le contenu de beaucoup de sites dépend aujourd'hui d'applications et services tiers : Youtube ou Vimeo pour les vidéos, des présentations Slideshare, des widgets Facebook, des flux Twitter, des cartes Google maps, etc. Beaucoup de ces services tiers utilisent aujourd'hui des iframes pour proposer leur contenu sur d'autres sites. Mais ne nous voilons pas la face : les iframes sont encore une fois une plaie à utiliser dans un contexte de mobilité. Les largeurs et hauteurs sont fixées en direct dans le code HTML. Forcer une largeur de 100% sur `<iframe>` pourrait marcher, mais cela voudrait dire perdre le ratio du contenu embarqué. Les vidéos sont souvent en 16:9 par exemple, pour garder ce ratio il va falloir trouver des solutions alternatives.

## **Du bricolage HTML et CSS**

[Thierry Koblentz](#) a écrit un très bon article intitulé ["Creating Intrinsic Ratios for Video"](#) dans lequel il propose une méthode pour avoir des vidéos dans des iframes tout en préservant le ratio 16:9 de départ. Cette solution peut être étendue à toutes sortes d'iframe : vidéos, présentations SlideShare et même cartes GoogleMaps.

Koblentz entoure son iframe d'un conteneur auquel il va donner une classe pour lui appliquer ensuite du CSS. Ce conteneur permet d'avoir un *iframe fluid*, même si les valeurs de largeur et hauteur sont données en pixels et en dur dans le HTML. Voici à quoi ressemble [ce code adapté par Anders M. Anders](#) :

```
.embed-container {
  position: relative;
  padding-bottom: 56.25%; /* 16:9 ratio
*/
  padding-top: 30px; /* IE 6 workaround*/
  height: 0;
  overflow: hidden;
}
.embed-container iframe,
.embed-container object,
.embed-container embed {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
}
```

Ce code va fonctionner pour tous les iframes, à partir du moment où vous les aurez entourées de `<div class="embed-container">iframe</div>`.

Même si cette solution peut-être facilement mise en place sur un site sur lequel on a un contrôle total, il est difficilement envisageable de demander à un client sans compétences techniques d'ajouter ce HTML à chaque fois qu'il va récupérer une vidéo sur Youtube. Il serait bien sûr possible d'utiliser du JavaScript pour détecter tous les iframes de la page et les encapsuler automatiquement dans notre code. Mais il s'agit hélas encore une fois de bricolage et non pas d'une solution pérenne.

## La gestion des vidéos responsive dans le futur

HTML5 propose de nouvelles possibilités pour les vidéos avec l'élément [video](#). La bonne nouvelle : [son support est étonnamment bon pour les mobiles](#). En dehors d'Opera Mini, la majorité des navigateurs supportent la balise `<video>`. L'autre bonne nouvelle : cette balise est très flexible en CSS. Pour présenter une vidéo responsive, il suffirait d'écrire :

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

Et là vous vous demandez : où est le problème alors ?

Le problème vient tout simplement des plateformes qui vont proposer les contenus. Même si Youtube ou encore Vimeo supportent la balise video, ils produisent toujours cet horrible `<iframe>` lorsque l'on veut afficher la vidéo sur son site. Nous sommes donc condamnés à trouver des bidouilles et des bricolages comme ceux évoqués plus hauts tant que Youtube, Vimeo ou Dailymotion ne se décident pas à proposer un moyen plus propre d'afficher leurs vidéos sans utiliser d'iframe. En attendant ce jour, Chris Coyier propose un plugin jQuery appelé [FitVids.js](#) qui utilise la technique proposée plus haut et se charge de faire l'encapsulage de l'iframe dans un conteneur pour améliorer sa fluidité tout en préservant le ratio de la vidéo.



## Afficher les cartes Google Maps (ou équivalent)

Si vous décidez à afficher pour votre site une carte Google Maps, la technique décrite plus haut avec le conteneur va fonctionner. Mais encore une fois, c'est une bidouille pas propre du tout. En plus dans le cas des cartes, elles vont se redimensionner en gardant leurs proportions ce qui veut dire que sur un petit écran, vous risquez de vous retrouver avec une carte toute petite qui aura perdu le focus sur l'endroit que vous vouliez montrer à l'utilisateur. La documentation officielle de [Google Maps pour mobile](#) stipule qu'il vaut mieux utiliser [l'API de cartes statiques](#) pour un affichage sur mobile. Et en effet, utiliser une carte statique (via une balise `<img>`) fera disparaître notre souci d'iframe. Brad Frost a écrit un excellent article sur le sujet intitulé [adaptive maps](#) dans lequel il propose une démonstration qui illustre cette technique. Un JavaScript détecte la taille de l'écran et en fonction va remplacer l'iframe par une carte statique sur les petits écrans. Et encore une fois en l'absence de meilleure solution native (proposée par Google par exemple), nous avons recours à une bidouille pour pouvoir nous débarrasser de notre problème d'iframe.

## Il nous FAUT de meilleures APIS !

J'ai jusque là parlé de bidouille, de bricolage : existe-t-il une meilleure solution ? Le plus gros souci des iframes est notre manque de contrôle en tant que designers et intégrateurs sur le code et design qu'elles vont générer. Nous sommes totalement dépendants de ces services tiers et par extension, du HTML et CSS qu'ils veulent bien nous fournir. Dans la mesure où le nombre de services et sites qui proposent ce genre de contenus tierces est en constante augmentation, il va rapidement nous falloir de meilleures solutions que les iframes pour afficher leur contenu sur nos sites.

Soyons réalistes deux secondes : utiliser le widget Facebook pour afficher les *likes* sur son site est une plaie. Le peu de contrôle sur le CSS généré par Facebook rend notre travail hasardeux et peut aller jusqu'à ruiner un joli design. Le web est un endroit libre et ouvert, il est donc peut-être enfin temps de commencer à penser en terme d'APIs ! Dans le futur, nous aurons à la fois besoin de plus d'APIs (Application Programming Interface) pour récupérer ces contenus, mais enfin et surtout, besoin d'APIs plus flexibles et surtout d'APIs plus simples d'utilisation pour récupérer ces contenus sans les iframes. Mais jusqu'à ce que ces grands services se décident à nous créer ces APIs, nous sommes coincés avec des iframes hasardeux et condamnés au bricolage et à la bidouille.

## La navigation responsive: tour d'horizon des techniques actuelles

Un autre gros défi quand on parle de responsive et d'optimisation est la navigation du site. Point souvent critique pour l'utilisateur, plus elle est complexe et dense, plus il va falloir ruser d'inventivité pour proposer une expérience agréable et simple d'utilisation.

Lorsque le Responsive Web Design n'en était qu'à ses débuts, une première solution était de [transformer la navigation en listes déroulantes](#). Cette solution est hélas loin d'être idéale. Elle devient très compliquée dès que l'on a plusieurs niveaux de navigation, et peut poser des soucis d'accessibilité. Je vous conseille de lire l'excellent article [Stop Misusing Select Menus](#) pour en apprendre un peu plus sur cette technique et les problèmes liés.

Quelques personnes comme [Brad Frost](#) ou encore Luke Wroblewski ont alors essayé de proposer différentes solutions à ce casse tête. Brad Frost compile quelques techniques de navigation responsive sur son site [This Is Responsive](#).

La navigation qu'il appelle "Toggle" consiste à cacher la navigation pour les petits appareils sous un lien "menu". Quand l'utilisateur clique sur ce lien, le reste de la navigation apparaît sous forme de blocs juste en-dessous, poussant le contenu plus bas dans la page.

Une variante de cette technique a été inspirée par les navigations d'applications natives comme celle de Facebook et est appelée [off-canvas](#). Encore une fois on cache la navigation sous une icône "menu". Quand l'utilisateur clique sur cette icône, la navigation sort d'un panneau à gauche ou à droite du site, poussant le contenu de l'autre côté.

Le "problème" de ces techniques est que la navigation reste en haut de l'écran. Dans son article "[Responsive Navigation: Optimizing for Touch Across Devices](#)," Luke Wroblewski illustre de manière visuelle les zones faciles d'accès sur différents appareils. La partie en haut à gauche est la plus compliquée à atteindre sur mobile.

En se basant sur ces recherches, Jason Weaver créa quelques démonstrations [de navigations avec les boutons en bas de l'écran](#). Une des solutions proposées est d'avoir la [navigation en pied de page](#), et d'y renvoyer l'utilisateur lorsqu'il clique sur le bouton "menu" à l'aide d'un système d'ancres HTML.

[Bien d'autres solutions](#) ont été proposées pour tenter de régler le problème de la navigation responsive. Comme vous pouvez le voir, encore une fois, il n'existe pas de solution idéale et cela va là encore dépendre non seulement de votre projet, mais également du niveau de profondeur de votre navigation. Heureusement qu'il existe des gens comme ceux cités plus haut qui expérimentent et partagent les expériences avec la communauté.

Il nous faut également encore trouver quelles icônes choisir pour montrer à l'utilisateur de manière pertinente qu'il y a un menu, et qu'il se cache des éléments supplémentaires en dessous. Certains sites utilisent le symbole (+), d'autres une grille, d'autres encore utilisent un symbole de liste ordonnée, ou encore le symbole des 3 lignes (ou *burger icon*).

Pour voir ces icônes en utilisation je vous conseille l'excellent article "[We Need a Standard 'Show Navigation' Icon for Responsive Web Design](#)."

Reste à trouver laquelle de ces icônes est la plus facilement identifiable par un utilisateur comme étant un menu. Si tout le monde s'accordait sur l'utilisation d'une d'entre elles, les utilisateurs (de tous pays) seraient-ils capables de la comprendre ? Et laquelle choisir ? Je suis très curieuse de savoir quelle icône vous utilisez sur vos sites, donc n'hésitez pas à partager votre expérience dans les commentaires ci-dessous.

## **Les spécificités du mobile : "Mon utilisateur est-il sur un appareil mobile? Et que peut-il y faire?"**

Le monde des mobiles et tablettes est tout nouveau pour nous, loin des navigateurs de bureaux, et avec ses propres règles, comportements et capacités. Il va donc nous falloir penser à adapter nos design à toutes ces nouvelles possibilités.

## **Détecter la possibilité d'évènement "touch" en JavaScript natif**

Je pense que si vous demandez dans la rue à quelqu'un de vous donner la différence entre un navigateur de bureau et mobile, en dehors de la taille, il y a fort à parier que cette personne va vous répondre l'utilisation avec les doigts. Il n'y a pas de souris sur les mobiles (ha bon ? ;) ) et en dehors de quelques rares appareils hybrides et ajouts en bluetooth sur des tablettes, il n'est pas possible d'avoir des événements de souris sur un mobile ou une tablette. Cela implique qu'en fonction du navigateur, la pseudo-classe `:hover` fonctionnera différemment. Certains navigateurs sont sympatiques et "émulent" cette fonctionnalité, traduisant un effet au survol par un effet au touch. Hélas tous ne sont pas aussi flexibles. Créer un design dont une partie des éléments ne sont révélés qu'au survol de la souris peut

s'avérer compliqué sur mobile.

Récupérer les évènements au touch est une solution envisageable. Le W3C a un groupe de travail sur le projet de spécifications [touch event](#). Dans le futur, nous serons donc capables de détecter directement en JavaScript et sans librairie tierce comme [Hammer.js](#) ou [jGestures](#) des évènements comme [touchstart](#), [touchmove](#) ou encore [touchend](#).

JavaScript est une chose, mais qu'en est-il du CSS ?

## CSS Level 4 "pointer media query"

Le CSS Level 4 propose une nouvelle Media Query appelée "[pointer](#)". Elle peut être utilisée pour détecter la présence d'un dispositif de pointage comme une souris, et prend 3 valeurs possibles :

- [none](#) : l'appareil n'a pas de dispositif de pointage
- [coarse](#) : l'appareil a un dispositif de pointage avec une précision limitée (un mobile ou une tablette où le dispositif de pointage est un doigt)
- [fine](#) : l'appareil dispose d'un dispositif de pointage précis comme une souris, ou encore un track pad ou le stylet d'une tablette graphique

En utilisant cette requête, on peut imaginer ce genre de code pour élargir la taille des boutons pour des appareils "touch" :

```
@media (pointer:coarse)
{
  input[type="submit"],
  a.button {
    min-width: 30px;
    min-height: 40px;
    background: transparent;
  }
}
```

Cette Media Query n'est pour le moment supportée nulle part et est à l'état de proposition. Le potentiel n'en reste pas moins très intéressant puisqu'elle permettrait de détecter des appareils supportant le touch sans devoir passer par une librairie comme [Modernizr](#).

## La Media Query CSS Level 4 "hover"

Les spécifications CSS Level 4 proposent également une nouvelle Media Query appelée [hover](#), qui est capable de détecter si le dispositif de pointage de l'appareil peut supporter les effets au survol. Elle renvoie un booléen 1 si l'appareil supporte le survol, 0 si ce n'est pas le cas. Notez ici que cela n'a RIEN à voir avec la pseudo-classe [:hover](#).

En utilisant cette syntaxe, nous pourrions par exemple améliorer les interfaces et ne masquer des fonctionnalités que pour les appareils qui supportent le survol d'éléments. Le code pourrait ressembler à ça :

```
@media (hover) {  
  .hovercontent { display: none; } /* cacher le contenu uniquement pour les  
appareils qui supportent le survol */  
  .hovercontent:hover { display: block; }  
}
```

On pourrait imaginer l'utiliser pour créer des menus déroulants au survol uniquement pour les appareils le supportant, et proposer une alternative plus facilement pour les autres sans avoir besoin de détection des capacités de l'appareil.

## La Media Query CSS Level 4 "luminosity"

Les appareils mobiles sont aujourd'hui équipés de détecteurs leur permettant de mesurer la luminosité ambiante. CSS Level 4 propose une [media-query "luminosity"](#) qui aura accès aux données de ces capteurs. Voici comment la décrivent les spécifications du W3C :

" La Media Query luminosité est utilisée pour récupérer des données sur la luminosité ambiante dans laquelle l'appareil se trouve et permet à l'auteur d'ajuster le style du document en fonction. "

Dans le futur nous serons donc capables de créer des sites qui vont **réagir à la luminosité ambiante**. Nous allons pouvoir grandement améliorer l'expérience utilisateur en leur proposant par exemple plus de contrastes dans un environnement très lumineux avec la valeur "washed". La valeur "dim" est utilisée pour les environnements sombres, et la valeur "normal" si le niveau de luminosité ne requiert pas d'ajustement.

Voici à quoi pourrait ressembler le code :

```
@media (luminosity: washed) {  
  p { background: white; color: black; font-size: 2em;  
  }  
}
```

Si vous êtes curieux et souhaitez en savoir plus sur les nouveautés CSS level 4 pas forcément liées au responsive, vous pouvez lire l'article [Sneak Peek Into the Future: Selectors, Level 4](#).

## Plus de capacité mobiles détectables en utilisant les APIs et JavaScript

Beaucoup d'autres fonctionnalités peuvent être détectées pour améliorer l'expérience utilisateur et proposer un site responsive digne de ce nom. Par exemple il serait possible d'accéder nativement au gyroscope, à la boussole ou à l'accéléromètre pour détecter l'orientation de l'appareil en utilisant [DeviceOrientationEvent](#). [Le support de cette API](#) s'améliore pour Android et iOS mais elle reste pour le moment à l'état de brouillon. Les possibilités pour les jeux HTML5 qui exploiteraient l'orientation directement dans le navigateur n'en restent pas moins prometteuses.

Une autre API qui commence à être beaucoup utilisée et particulièrement utile est la [Géolocalisation](#). Cette API est d'ailleurs [très bien supportée](#). Elle permet de positionner l'utilisateur en utilisant son GPS, l'adresse IP ou encore les réseaux Wi-Fi ou la triangulation d'antennes relais. On peut imaginer utiliser la géolocalisation sur un site responsive pour proposer des informations contextuelles à l'utilisateur. Une grosse chaîne de restaurants pourrait pas exemple améliorer l'expérience en montrant à l'utilisateur géolocalisé les restaurants de la chaîne qui se trouvent autour de lui.

Le W3C propose également un brouillon pour [l'API Vibration](#). Le navigateur peut alors proposer des retours tactiles sous forme de vibrations à l'utilisateur. Néanmoins on sort là quelque peu du domaine du responsive pour entrer un peu plus dans le domaine du jeu et de l'applicatif web.

Une autre API qui a fait couler beaucoup d'encre est [Network Information](#). La possibilité de pouvoir optimiser des images et un site en fonction de la bande passante de l'utilisateur a de quoi séduire plus d'un développeur. Nous serions en théorie alors capables de proposer des images en haute résolution pour les appareils avec une grosse bande passante, et vice et versa. Avec l'attribut `bandwidth` de l'API network, il serait possible d'estimer la bande passante d'un utilisateur en Mb/s. Le second attribut `metered` est un Boolean qui nous indique si l'utilisateur a une connexion mesurée, comme c'est le cas pour des mobicartes pré-payées par exemple. Ces deux attributs sont accessibles en JavaScript.

Hélas la mesure de la connexion d'un utilisateur est une chose techniquement complexe. En outre, la connexion peut changer à tout moment. Un utilisateur passant sous un tunnel peut perdre sa connexion, et la vitesse de téléchargement va subitement chuter. Il est donc totalement hypothétique d'imaginer à l'heure actuelle une Media Query "magique" qui permettrait de mesurer la bande passante. Yoav Weiss a écrit un excellent article sur le problème qu'une telle Media Query pourrait engendrer et sur la mesure de la bande passante en général intitulé [Bandwidth Media Queries? We Don't Need 'Em!](#).

Il existe encore beaucoup d'autres APIs liées aux fonctionnalités mobiles que je ne vais pas détailler ici. Si vous souhaitez en savoir plus, [Mozilla a une liste très détaillée](#). La plupart d'entre elles sont loin d'être standardisées, et beaucoup sont plus orientées vers de l'applicatif web que vers le site responsive. Il n'en reste pas moins passionnant de voir à quel point la mobilité va devenir complexe dans le futur et toutes les possibilités qui vont s'ouvrir à nous.

## **Repenser la façon dont nous, et nos clients/utilisateurs gérons le contenu.**

Il reste encore bons nombres de défis techniques à relever pour gérer du contenu de manière efficace sur mobile. La philosophie "mobile-first" commence à faire de plus en plus d'émules dans la communauté de designers et d'intégrateurs. Il serait par exemple techniquement possible d'envoyer à un mobile le minimum de HTML et données nécessaires, puis d'utiliser du JavaScript et des chargements en AJAX pour charger plus de contenu et d'images de manière conditionnelle pour les navigateurs de bureau, voire les tablettes. Mais pour faire cela, il nous faut d'abord **repenser la manière dont nous allons gérer notre contenu**. Il nous faut être capable de prioriser et hiérarchiser les contenus pour les générer de la manière la plus flexible et adaptive possible. Un bon exemple ici serait la carte responsive décrite plus haut : nous chargeons pour le mobile une simple image, et améliorons progressivement l'expérience avec une "vraie" carte pour les utilisateurs de navigateur sur bureau. Plus nous allons vouloir rendre un site responsive, et plus la gestion du contenu va se complexifier.

Du code plus flexible peut nous aider à formater un contenu qui s'adapte. Certaines personnes ont suggéré de créer des phrases responsive en morcelant les éléments de balises HTML et de classes pour pouvoir n'en afficher que certaines parties en fonction de la taille de l'écran. L'idée est de masquer certaines parties pour les petits appareils en utilisant des Media Queries. Vous pouvez voir cette technique en action sur le blog de 37signal's [Signal vs. Noise](#) ainsi que dans l'article de Frankie Roberto intitulé [Responsive Text](#). Même si cette technique peut être utilisée pour améliorer l'affichage de petites parties du site comme un slogan dans un pied de page, il est totalement impensable de l'appliquer à grande échelle à l'ensemble d'un site web.

On commence ici à mettre le doigt sur un problème qui va prendre de plus en plus d'importance dans le futur : la nécessité de parser son contenu textuel de métadonnées pour lui donner une plus grande structure sémantique. Rappelons-nous, le contenu de notre site ne vient plus uniquement de nos rédacteurs de contenus en interne. Si nous souhaitons à terme, être capable de réutiliser le contenu

d'autres site sur le notre, il va falloir préparer la structure de ce contenu pour qu'il soit facilement réutilisable. Les nouveaux éléments HTML5 comme [article](#), [section](#) sont un bon point de départ pour faire gagner de la valeur sémantique à nos contenus. Dans le futur, il ne faudra plus penser et structurer le contenu comme une seule entité (une page web) mais le penser comme l'assemblage de multiples blocs réutilisables que l'on va pouvoir afficher dans différents formats sur différents appareils.

Le plus grand défi sera de **faire en sorte que la structuration en métadonnées soit facile d'accès et compréhensible** par toutes les personnes qui font partie de la chaîne de création d'un site web. Nous allons devoir les éduquer, et leur expliquer comment ces données peuvent être utilisées pour prioriser le contenu et assembler de manière pragmatique différents éléments tout en restant indépendant de la plateforme utilisée. Le grand challenge sera de les aider à penser en terme de blocs réutilisables plutôt qu'en terme de "grosse zone de texte que l'on peut copier/coller depuis Word vers l'éditeur WYSIWYG du CMS". Notre travail sera de les aider à comprendre que le contenu et sa mise en forme sont deux choses séparées et indépendantes, tout comme nous en tant que designers et intégrateurs avons compris que le contenu (HTML) et sa présentation (CSS) sont plus simples à gérer une fois séparés.

**Écrire un contenu orienté vers une seule et unique plateforme n'est plus une option** . Qui sait sur quels appareils votre contenu sera affiché demain ? Dans 6 mois ? Dans un an ? Il nous faut [préparer ce contenu pour l'inattendu](#). Mais pour arriver à cela, il va nous falloir également créer de meilleurs outils de publication. Karen McGrane a donné une conférence intitulée [Adapting Ourselves to Adaptive Content](#). Elle y parle d'exemples concrets tirés de l'industrie de la publication. Elle détaille le processus pour créer des contenus réutilisables et introduit l'idée de COPE (*create once and publish everywhere* - en français "créer une fois, publier partout"). Il va nous falloir construire de meilleurs CMSs qui seront capables de générer les métadonnées nécessaires à cette priorisation des contenus. Nous allons également devoir expliquer à nos collègues et rédacteurs de contenu comment ces CMSs fonctionnent et comment penser en terme de contenus réutilisables et non pas de pages WYSIWYG.

Karen McGrane écrit :

*"Vous serez peut-être amenés à écrire trois versions différentes de votre titre; deux versions du sommaire, et ajouter plusieurs images dans différentes tailles, différentes coupes, et vous ne serez même pas la personne qui décidera quelle image ou quel titre sera affiché sur telle ou telle plateforme. Cette décision sera faite par une métadonnée. Elle sera faite par les règles de l'industrie [...] **les métadonnées sont la nouvelle direction artistique.**"*

Masquer des contenus pour des appareils n'est plus une stratégie de gestion de contenu pérenne. Il nous faut des CMS qui puissent nous fournir la structure pour créer du contenu réutilisable. Il nous faut également des meilleurs workflows de publication dans ces CMSs. Les interfaces hasardeuses et complexes font peur aux utilisateurs, et la plupart des personnes qui vont générer les contenus ne sont pas particulièrement à l'aise avec des outils trop compliqués. Il va nous falloir leur fournir des outils qui sont faciles d'accès et de compréhension pour les aider à publier du contenu propre, sémantique et indépendant de la présentation finale.

## Conclusion

Cet article peut vous paraître long, **mais il n'est que le sommet de l'iceberg** . Aujourd'hui notre profession commence à comprendre que le responsive webdesign c'est bien plus que balancer quelques Media Queries dans une feuille de style, choisir des points de rupture et doubler les dimensions des



images pour ces nouveaux jouets haute résolution qui nous servent de mobile. Vous l'avez lu, vu, le chemin est long, et nous sommes loin d'être arrivés à destination. Il y a encore tellement de choses à faire et de problèmes à résoudre que la solution responsive magique et miraculeuse n'existe pas encore aujourd'hui.

Nous avons découvert des solutions techniques, d'autres viendront dans le futur en utilisant des nouvelles technologies présentées dans cet article avec l'aide d'organisations comme le [W3C](#), le [WHATWG](#) ou encore le [Filament Group](#).

Mais gardons une chose importante à l'esprit : c'est également à nous, designers, nous intégrateurs, nous développeurs d'aider à trouver de meilleures solutions. Des gens géniaux comme [Luke Wroblewski](#), [Brad Frost](#), toutes les femmes et les hommes cités dans cet article et ceux que j'ai oublié œuvrent au quotidien, testent et expérimentent différentes techniques et solutions. Succès ou échecs, **la chose la plus importante reste le partage** ce que nous, designers, développeurs, rédacteurs de contenus et membres de la communauté webdesign créons au quotidien pour aider à surmonter ces défis présentés par le concept de responsive webdesign.

Nous sommes tous dans le même bateau, autant faire ensemble du web un endroit meilleur, non ?

"