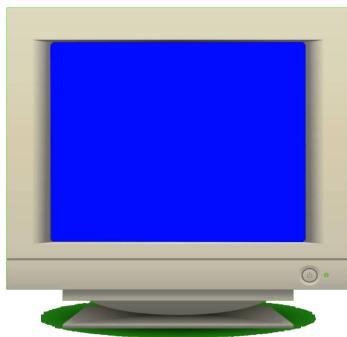


Page WEB dynamique



6. CSS Responsive Web Design

Avant 2007 : un univers (relativement) simple



Dans les années 80 : densité de pixels fixe (72 ppi¹), définie par Apple dont les imprimantes avaient une résolution de 144 dpi (pile le double).

L'émergence d'écrans de tailles variables a anéanti ce "standard" assez rapidement.

¹ : ppi (pixels per inch) : nombre de pixels sur 2,54 cm d'écran
on utilise parfois dpi (dots per inch) mais un "dot" (point) n'est pas un pixel : à éviter pour les écrans

Depuis 2007 : la situation devient (très) complexe

Depuis l'avènement des SmartPhones (en 2007), et plus généralement des dispositifs mobiles capables d'accéder à Internet, il a fallu prendre en compte l'affichage des pages web sur de petits écrans. En effet, l'utilisation d'un seule et même configuration conduit à une mauvaise expérience utilisateur



Le site du Monde
le 18/10/2015
sur un écran de
1920 x 1080

3

Le site du Monde le 18/10/2015



sur un écran de 1280 x 1024

sur un écran de 320 x 480
→ beaucoup de scroll...



sur iPhone 6
→ lisibilité ?

Remarque : on affiche ici la même page que sur l'écran 1920*1080

4

La solution "classique"



Un site pour écran d'ordinateur



Un site pour mobile



Une app pour iOS



Une app pour Android

La solution classique : autant de développements que de (groupes cohérents de) dispositifs ciblés
→ Coût !

→ Maintenabilité ?

→ Cohérence entre les différentes versions ?

5

Mais...



Diversification

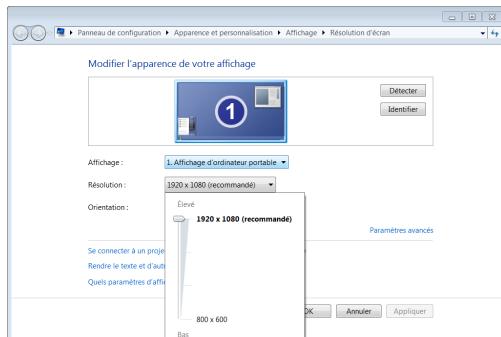
- des dispositifs (OS, navigateur),
- des surfaces d'affichages (tailles d'écrans)
- de la densité de pixels

Absence de standard de tailles d'écrans

Impossible de (re)développer le site pour chaque configuration.

6

Les pixels : pixels physiques, pixels OS

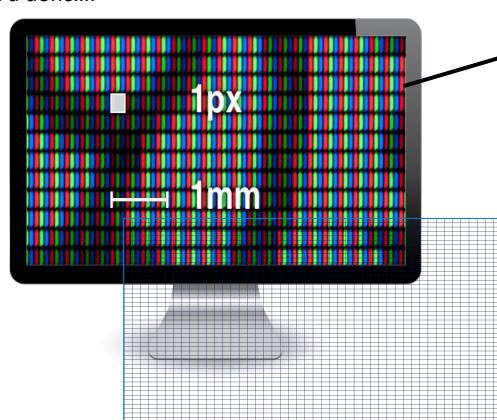


Depuis longtemps, on peut demander à l'OS de prendre en compte une résolution différente de la résolution physique, comme ici sur Windows.

7

Pixels physiques, pixels OS

On a donc....



La définition¹ physique, le nombre de pixels physiques qui constituent concrètement l'écran

La définition (nombre de pixels) considérée par l'OS, qui peut différer de la définition physique

En pratique : la définition physique n'a aucune importance, on ne la connaît pas et on ne sait pas la trouver.

¹ La "définition" d'un écran est le nombre de pixels qu'il contient (1920×1080), qu'il convient de distinguer de sa "résolution", qui est une densité de pixels (x pixels par cm ou plus classique par pouce).

En 2007 : arrivée de l'iPhone

Apple souhaite créer un dispositif capable d'accéder à Internet et (entre autres) d'afficher une page web.

Problème : comment afficher une page prévue pour des écran d'ordinateur avec des définitions importantes sur un écran de 320 pixels de large ?

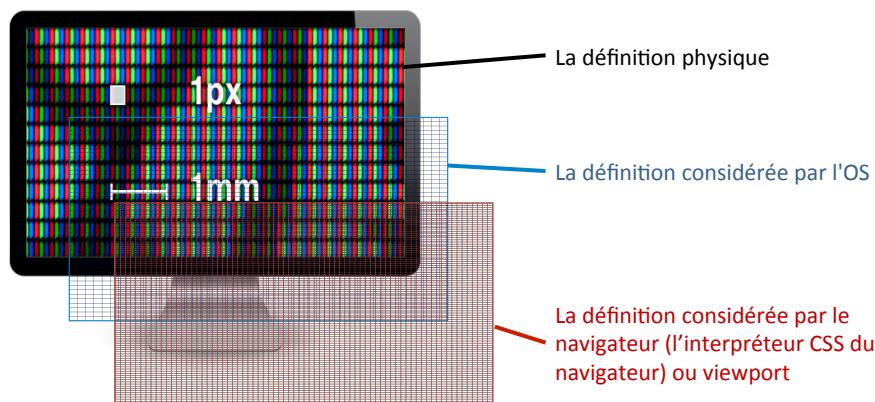
Apple crée alors la notion de viewport : son navigateur (Safari) "croit" qu'il a beaucoup plus de pixels qu'il n'en a réellement et des fonctions mathématiques recalculent la présentation pour la rendre affichable sur les pixels disponibles.



9

Pixels physiques, pixels OS, pixels CSS

Avec cette nouveauté, on se trouve avec 3 définitions indépendantes des pixels de l'écran :



Bibliographie : M. Lechat, *Pixel CSS et pixel physique*, juillet 2015 ; M. Area, *Pixel de référence en CSS*, décembre 2014

10

Pixels : exemple de l'iPhone 6

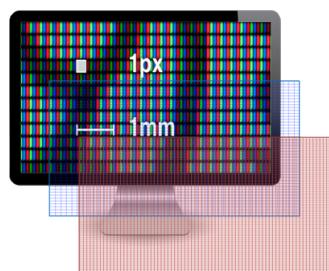


Pixels physiques : 750 x 1334
Pixels OS : 750 x 1334
Pixels CSS : 980 x 1461 sur Safari
980 x 1545 sur Chrome

11

Le viewport

Définition : nombre de pixels dont le navigateur croit disposer, et qui entraîne une interpolation calculée.



AVANTAGES

- Permet la navigation sur le web sans modification de la page visitée
- Indépendance des pixels OS

INCONVENIENTS

- Adaptation calculée automatiquement (pas toujours optimale)
- Page web dégradée
- Utilisation / lecture plus difficile
- Résultat différent selon le navigateur (nombre de pixels défini par le navigateur)

12

Les pages pour mobiles

Face aux inconvénients du viewport, la première réaction fut de créer des pages dédiées aux smartphones (mobile.lemonde.fr) et de rediriger sur ces pages les requêtes qui provenaient d'un dispositif mobile.

Mais cette solution a elle-même des limites : la diversification croissante des tailles d'écrans fait qu'il n'y a plus UNE bonne solution pour TOUS les dispositifs mobiles.



Bibliographie : Y. Poiron, *Une visualisation parfaite de la fragmentation d'Android*, août 2013, d'après une enquête de OpenSignal

13

Le Responsive Web Design (RWD)

Depuis les premières propositions en 2009, le Responsive Web Design a été défini de plusieurs manières. Toutes ont un objectif : proposer *une expérience visuelle élégante quelle que soit la taille de l'écran de l'utilisateur* (Jeffrey Zeldman).

Dans la définition originelle (Ethan Marcotte, 2011) du Responsive Web Design, il s'agit de

- mettre en œuvre des **media queries** CSS pour proposer des rendus graphiques différents **selon la largeur** de l'écran
- utiliser des "**grilles fluides**" (dimensionnement des différents blocs de la page en unités relatives comme les pourcentages ou les em)
- utiliser des "**images fluides**" (dimensionnement aussi en unités relatives, voire utilisation d'images avec différentes résolutions pour couvrir les différentes densités de pixels).

Bibliographie : H. Giraudel & R. Goetter, *CSS3, Pratique du design Web*, Eyrolles, 2015

14

Le Responsive Web Design (RWD)

Plus récemment, le Responsive Web Design est défini comme l'ensemble des techniques permettant de **maintenir au mieux l'expérience utilisateur dans différentes configurations**. Ainsi, les adaptations programmées en JavaScript sont donc aujourd'hui considérées comme des éléments du Responsive Web Design.

L'idée centrale est de considérer qu'on a d'une part un conteneur (l'écran) et d'autre part le contenu (la page) à afficher au mieux dans le conteneur, avec toute la flexibilité qu'a un liquide à s'adapter au récipient dans lequel on le verse.



Stéphanie Walter, 1980

15

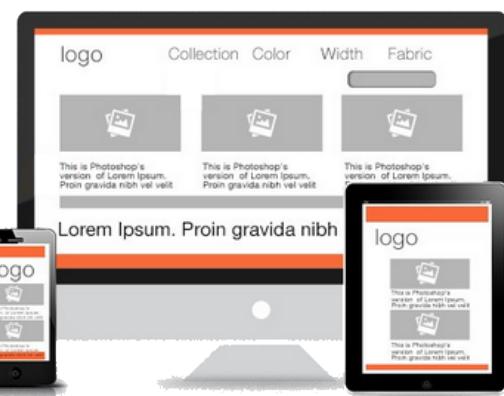
RWD : adaptation ou dégradation ?

L'objectif est donc de maintenir **au mieux** l'expérience utilisateur, c'est-à-dire (en simplifiant) l'utilisabilité du site, le plaisir d'utilisation et les fonctionnalités offertes.

L'exemple ci-contre présente une adaptation.

Il y a cependant une petite dégradation : le menu présent sur le grand écran (facilement accessible) est invisible sur les deux autres écrans. L'auteur propose de le remplacer par un bouton qui le fait apparaître, ou de le placer en bas d'écran. L'utilisateur doit alors comprendre l'utilité du bouton (charge cognitive) ou se déplacer jusqu'en bas de l'écran (charge de travail).

En pratique, il est presque impossible d'éviter les dégradations. L'important est de bien les choisir et donc de les analyser.



Bibliographie : F. Grenet, RWD, décryptage des bonnes pratiques en eCommerce, startoppers.co, 2013

16

RWD en pratique : les points de rupture

L'objectif étant d'offrir différentes présentations pour différentes surfaces d'affichage à l'aide de *media queries*, il faut dans un premier temps définir les points de rupture, c'est-à-dire les plages de valeurs dans lesquelles chaque présentation doit être affichée.

En pratique, on ne prend en compte (le plus souvent) que la largeur de l'écran, le fait d'imposer un *scrolling* vertical étant considéré comme moins pénalisant.

Bibliographie : J. Beranger, *7 conseils pour réaliser un site en Responsive Web Design*, mars 2015
V. Pereira, *Les 10 bonnes pratiques du responsive design*, avril 2015

17

RWD en pratique : le point de départ

Le principe est d'offrir une utilisabilité forte dans un environnement et de dégrader cette utilisabilité pour les autres environnements... Mais alors quel est l'environnement favorisé ?

On lit souvent : **mobile first** = commencer par la version pour mobile.

Bénéfice attendu : favoriser la navigation la plus sensible / difficile
Inconvénient : incertitude sur l'optimisation globale de l'utilisabilité

Préconisation : *commencer par l'environnement le plus utilisé ou l'environnement ciblé prioritairement (et il y a de bonnes chances que ce soit le mobile).*

Bibliographie : J. Beranger, *7 conseils pour réaliser un site en Responsive Web Design*, mars 2015
V. Pereira, *Les 10 bonnes pratiques du responsive design*, avril 2015

18

RWD en pratique : les points sensibles

Question : quelles sont les différences impactantes entre un environnement de type PC / portable et un environnement mobile ?

- La taille d'écran (évidemment)
- La modalité d'interaction : clavier + souris vs tactile
- Variations de capacités :
 - sur mobile, pas de tabulation, pas de bouton Esc, pas de raccourcis claviers (ctrl/+ , alt/tab, ctrl/a...), pas de survol de zone (tooltips), une seule main disponible (pour la plupart des utilisateurs), pas de touche + et – dans les champs numériques, pas de répétition automatique par appui sur une touche, réseau fluctuant, bande passante réduite....
 - sur "PC" : pas (ou peu) de tactile, multi-fenêtrage,...

➔ CE N'EST PAS SEULEMENT UNE QUESTION DE PRÉSENTATION, IL FAUT REPENSER L'INTERACTION.
RWD apporte des briques techniques pour développer ce qui a été ainsi conçu.

19

Démarche de développement multi-supports

1. **Identifier le besoin** auquel on veut répondre
2. Faire une **étude de l'existant**
 - a. **Interne** : a-t-on déjà un site qui traite du besoin ? Quelles sont ses limites ? Quels sont ses points forts ?
 - b. **Externe** : comment font les autres dans le même domaine ou dans un domaine proche
Exemple : site de vente de tomates.
Regarder Amazon, Fnac, Ooshop, Booking.com, blablacar, ...
3. **Caractériser les supports** visés (points de rupture en taille d'écran + compétences interactionnelles)
4. **Identifier les interactions** correspondantes sur chacun des supports visés, avec leurs spécificités
5. **Vérifier si le Responsive Web Design suffit à couvrir toutes les contraintes**
6. Si oui,
 - a. Définir les points de rupture entre supports, i.e. les différentes tailles d'écrans
 - b. Coder la page pour le support principal
 - c. Etendre la page pour les autres supports

20

Les points de rupture

Le nombre de points de rupture pertinents relève de votre objectif graphique et de la situation courante : si vous devez prendre en compte...

- les (futures) montres connectées capables de surfer...
- Les téléviseurs à haute définition...
- Les écrans géants publicitaires...
- Les vidéoprojecteurs embarqués dans les téléphones...
- ...

Ou à l'inverse si votre site est destiné à une utilisation interne (intranet) sur ordinateurs uniquement...

... vous devrez sans doute augmenter (ou à l'inverse diminuer) le nombre de points de rupture que vous prenez en compte !

21

Les points de rupture

Classiquement, on prend en compte 3 voire éventuellement 4 largeurs d'écran (w) :

- **w ≤ 479px :** Smartphones en portrait
- **480px ≤ w ≤ 959px :** Smartphones en paysage, tablettes en portrait, petites tablettes en paysage
- **960px ≤ w ≤ 1280px :** tablettes en paysage, petits écran d'ordinateur
- **[1281px ≤ w :** grands écrans d'ordinateurs ($\geq 21''$)]

On peut aussi vouloir prendre en compte des écrans plus grands mais avec une résolution moins importante (ex : téléviseur de 42" en Full HD, sur lequel les textes doivent être lisibles jusqu'à 5 mètres) :

- **[1920px ≤ w ET résolution ≤ 72 dpi]**

Ces points de rupture sont ensuite intégrés dans des media queries spécifiques pour chaque support :

```
@media screen (min-width:480px) and (max-width:959px) and (min-resolution:73dpi) { ... }
```

22

Prise en charge du viewport

Objectif : désactiver le fonctionnement habituel du viewport; indiquer au navigateur de ne plus chercher à redimensionner la page (puisque vous allez le gérer vous-même).

avec une balise meta dans la section <head>
de votre page :

```
<meta name="viewport" content="...">
```

avec la directive @viewport en CSS :

```
@viewport {  
    ...  
}
```

Remarques :

1. Cette balise, inventée par Apple, ne fait partie d'aucune norme, même si la plupart des navigateurs l'ont adoptée
2. Il n'est pas particulièrement logique que cette configuration du viewport soit faite sous la forme d'une balise HTML, puisque c'est le CSS qui doit la prendre en compte. Le W3C l'a donc normalisée sous la forme d'une directive CSS, qui n'est pour l'instant pas implémentée dans tous les navigateurs... **Cependant IE 10 supporte la directive CSS mais PAS la balise meta.** **Concrètement, il est donc pertinent d'utiliser les deux.**

23

La balise meta viewport

```
<meta name="viewport" content="...">
```

Dans l'attribut content, on peut indiquer (séparer les attributs par une virgule) :

- une largeur : **width=valeur**, où valeur peut être
 - une taille : **640px** (pas très intéressant)
 - **un alignement sur les pixels OS : width = device-width (recommandé)**
- une hauteur : **height=valeur**, où valeur peut être
 - une taille : **980px** (pas très intéressant)
 - un alignement sur les pixels OS : **height = device-height** (utilité limitée)
- **un facteur de zoom :**
 - **initial-scale = 1 (recommandé)**
- un facteur de zoom mini et/ou maxi :
 - **maximum-scale=2, minimum-scale=0.5**
- la possibilité pour l'utilisateur de zoomer (ou pas)
 - **user-scalable=yes** ou **user-scalable=no**

La balise recommandée est donc :

```
<meta name="viewport" content="width = device-width, initial-scale = 1 ">
```

Bibliographie : J. Patonnier & R. Rigot, *Projet Responsive Web Design*, Eyrolles, 2013

24

La directive @viewport

```
@viewport {  
    ...  
}
```

Tout d'abord, il est recommandé de définir le viewport dans une media query, en intégrant les directives spécifiques aux navigateurs :

```
@media screen and (max-width: 397px) {  
    @viewport, @-o-viewport, @-ms-viewport {  
        ...  
    }  
}
```

25

Les descripteurs de @viewport

- **min-width : valeur;** et **max-width : valeur;** ou **width : valeur ;** avec valeur
 - **640px** (pas très intéressant)
 - **device-width** : aligne la largeur sur les pixels OS (**recommandé**)
 - **auto** : la valeur sera calculée d'après les autres descripteurs (comme la hauteur spécifiée)
- **min-height : valeur;** et **max-height : valeur;** ou **height : valeur ;** avec valeur
 - **640px, auto,...** comme pour la largeur
 - **device-height** : aligne la hauteur sur les pixels OS
 - en général, on ne se sert pas de ce descripteur
- **min-zoom : valeur;** et **max-zoom : valeur;** ou **zoom : valeur ;** avec valeur
 - un nombre ou un pourcentage : **0.75, 1, 2, 75%, 100%, 200%**
 - **auto** : on laisse le navigateur décider (identique à ne pas spécifier de zoom ????)
- **user-zoom : valeur;** avec valeur
 - **zoom** : l'utilisateur est autorisé à zoomer / dézoomer sa page
 - **fixed** : l'utilisateur ne peut pas changer le facteur de zoom
- **orientation : valeur;** avec valeur
 - **auto** : l'orientation est celle du dispositif
 - **portrait** : quelle que soit l'orientation réelle du dispositif, l'affichage est en portrait
 - **landscape** : quelle que soit l'orientation réelle du dispositif, l'affichage est en paysage

Bibliographie : W3C, *CSS Device Adaptation Module Level 1*

26

Les polices responsive

Il n'y a pas UNE règle qui fasse consensus pour définir les tailles de police des différents éléments.

Certains auteurs préconisent des tailles en pixels en tablant sur la redéfinition du pixel par CSS.

D'autres préconisent l'utilisation d'unités relatives, comme em. Cependant, em impose une attention particulière car il repose sur la surcharge de la taille de l'élément parent.

Ainsi, si l'on définit li { font-size : 1.2em; }, tout ira bien pour une liste simple : les textes des li auront une taille égale à 1,2 fois (+20%) la taille du texte du parent.

Par contre, dans des listes imbriquées, les li imbriqués auront une taille 20% plus grande que les li de niveau supérieur.

Texte normal

- Niveau 1
- Niveau 1
 - Niveau 2
 - Niveau 2
 - Niveau 2

27

Police responsive : utilisation de rem

L'unité rem ne présente pas ce défaut : la taille est définie par rapport à la taille de l'élément racine (html).

Il est donc recommandé de définir une taille de police pour l'élément racine de la page (html) et d'utiliser l'unité rem pour tous les éléments inclus dans la page.

En remplaçant les em par des rem dans l'exemple précédent, tous les li, quel que soit leur niveau d'imbrication, ont la même taille :

Texte normal

- Niveau 1
- Niveau 1
 - Niveau 2
 - Niveau 2
 - Niveau 2

28

Police responsive : taille de police root

La question devient alors la taille de police à définir pour l'élément racine.

On trouve les recommandations suivantes :

- Une taille de police de 20px par défaut, ramenée à 16px sur les dispositifs de moins de 980px et à 18px pour les dispositifs de moins de 1580px (<http://codeitdown.com/responsive-font-size-css/>)
- Une taille de police de 1em, en se reposant sur le navigateur pour adapter cette taille au dispositif (<http://typecast.com/blog/a-more-modern-scale-for-web-typography>)
- Une taille de police de 16px (ce qui revient à 1em !), pour <https://developers.google.com/speed/docs/insights/UseLegibleFontSizes?hl=fr>
- ...

29

Police responsive : bonnes pratiques

D'autres bonnes pratiques doivent être respectées :

- **Respecter l'interligne** du navigateur (en général 1.2em)
- **Eviter de multiplier les polices** sur une même page (leurs différentes variations selon les dispositifs et les navigateurs peuvent conduire à des rendus incompréhensibles pour l'utilisateur – critère d'homogénéité de Bastien et Scapin)
- **Créer un "rythme vertical"**, c'est-à-dire des variations de tailles régulières entre les différents titres et contenus (éviter d'avoir un h1 en 12rem, le h2 en 4rem et le h3 en 3.9rem !), préférer par exemple h1 en 3.2rem¹, h2 en 2.8rem, h3 en 2.4rem etc.
- **Prévoir une solution de repli** pour les anciens navigateurs qui ne supportent pas l'unité rem (IE < 9). Par exemple, vous pouvez définir une première propriété avec une taille de police en px suivie immédiatement d'une propriété avec une taille de police en rem, qui surchargera la première si le navigateur supporte les rem.
- Surveiller les nouveautés CSS, comme *font-size-adjust*, qui assure une bonne lisibilité lorsque les polices de secours sont utilisées (et si le navigateur la supporte) et permet de spécifier la hauteur des lettres minuscules pour la standardiser entre les différentes polices (pour plus de détails, voir par exemple <http://babylon-design.com/mais-c-est-quoi-au-juste-font-size-adjust/>)

30

Le modèle de boites flexibles

Le modèle de boites flexibles offre une alternative aux positionnements classiques (inline, block,...).

Il permet en particulier de :

- distribuer les éléments en ligne ou en blocs
- contrôler la gestion des espaces disponibles
- contrôler les alignements horizontaux et verticaux
- agencer les éléments sans tenir compte du DOM

Le tout en allégeant les media queries requises pour ces adaptations, qui sont automatiques en fonction de la surface d'affichage disponible

Bibliographie : H. Giraudel & R. Goetter, *CSS3 Pratique du design web*, Eyrolles, 2011;
C. Coyier, *A complete guide to Flexbox*, sept.2015

31

Flexbox : illustration

L'exemple qui illustre les propriétés décrites dans la suite est une `<div id="monConteneur">` qui fait 30% de la taille de l'écran, contenant une phrase, chaque mot de la phrase étant encapsulé dans une `<div>` (un enfant sur deux a `color:blue`). `monConteneur` a une bordure rouge.

Affichage par défaut :

```
Il
était
une
fois
une
petite
fille
qu'on
appelait
le
petit
chaperon
rouge
et
qui
allait
voir
sa
grand-mère
en
passant
par
les
bois.
```

32

Flexbox : le conteneur

L'organisation d'une Flexbox repose sur le changement de la propriété display de l'élément conteneur :

```
#monConteneur {  
    display : flex ;  
}
```

Appliquer un *display : flex* à un conteneur...

- le transforme en *flex container* (conteneur flexible)
- transforme **TOUS** les enfants directs en éléments flexibles (*flex items*), même du texte sans balise
- ne nécessite pas de définir le display des enfants **directs**

L'affichage devient alors :

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

33

propriétés CSS d'un *flex-container* : flex-direction

Un conteneur flexible n'a pas une "direction" similaire aux display classique : là où *display : block* a une "direction verticale" et *display : inline* une "direction horizontale", un conteneur flexible a par défaut une direction "row" : ses enfants sont affichés en ligne (de gauche à droite ou de droite à gauche selon le sens de lecture par défaut).

On peut cependant spécifier *flex-direction = valeur* ; avec valeur :

- *row* : affichage dans le sens de lecture par défaut
Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.
- *row-reverse* : affichage dans le sens inverse du sens de lecture par défaut
et rouge chaperon petit le appelait qu'on fille petite une fois une était Il
- *column* : affichage de haut en bas
- *column-reverse* : affichage de bas en haut



34

propriétés CSS d'un *flex-container* : flex-wrap

Par défaut, les descendants directs d'un conteneur flexibles tentent de s'afficher sur une ligne (ou une colonne). Lorsque ce n'est pas possible, flex-wrap permet de spécifier comment le débordement est géré.

flex-wrap = valeur ; avec valeur :

- *wrap* : multi-lignes (multi-colonne) en conservant la direction

Il était une fois une petite fille qu'on appelait le petit chaperon rouge
et qui allait voir sa grand-mère en passant par les bois.

- *nowrap* : pas de retour à la ligne / à la colonne possible (déborde du conteneur si besoin)

- *wrap-reverse* : multi-lignes (multi-colonnes) en inversant la direction

et qui allait voir sa grand-mère en passant par les bois.
Il était une fois une petite fille qu'on appelait le petit chaperon rouge

35

propriétés CSS d'un *flex-container* : justify-content

justify-content permet d'indiquer comment le contenu du conteneur flexible doit être aligné horizontalement (**ou plus précisément dans la direction par défaut**).

justify-content = valeur ; avec valeur :

- *flex-start* : les items flexibles sont "tassés" vers le début de la ligne (dans le sens de la lecture)

Il était une fois une

- *flex-end* : les items flexibles sont "tassés" vers la fin de la ligne (toujours dans le sens de lecture)

Il était une fois une

- *center* : le groupe d'items est centré sur la ligne

Il était une fois une

- *space-between* : les items sont distribués horizontalement sur la ligne (verticalement dans la colonne)

Il était une fois une fois une

- *space-around* : l'espace restant est distribué autour des items de façon égale

Il était une fois une fois une

36

propriétés CSS d'un *flex-container* : align-items

align-items permet d'indiquer comment le contenu du conteneur flexible doit être aligné verticalement (**ou plus précisément dans la direction orthogonale à la direction par défaut**)

align-items = valeur ; avec valeur :

- *flex-start* : le "point de départ" des items (voir plus loin) est fixé au point de départ de la ligne
- *flex-end* : le "point d'arrivée" des items est fixé au point d'arrivée de la ligne
- *center* : le groupe d'items est centré sur l'axe perpendiculaire à la direction
- *baseline* : les items sont alignés pour que leurs baselines soient alignées
- *stretch* : les items sont agrandis pour remplir le conteneur (sauf si max-width spécifié)

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

37

propriétés CSS d'un *flex-container* : align-content

align-content détermine comment le contenu (pris dans son ensemble) du conteneur flexible doit être aligné verticalement

align-content = valeur ; avec valeur :

- *flex-start* : le groupe d'items est "tassé sur le départ du conteneur"
- *flex-end* : le groupe d'items est "tassé sur la fin du conteneur"
- *center* : le groupe d'items est centré sur l'axe perpendiculaire à la direction
- *space-around* : les lignes sont espacées régulièrement
- *space-between* : L'espace résiduel est réparti entre les lignes
- *stretch* : la hauteur des lignes est agrandie pour occuper tout l'espace résiduel

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

38

propriétés CSS d'un *flex-item* : order

Par défaut, les descendants directs d'un conteneur flexibles sont considérés dans l'ordre du code source. La propriété order permet de définir l'ordre dans lequel ils doivent être affichés.

sans order

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

avec order : 1
sur les éléments
en texte bleu et
order : 2 sur les
autres

Il une une fille appelait petit rouge qui voir grand-mère passant les
était fois petite qu'on le chaperon et allait sa en par bois.

39

propriétés CSS d'un *flex-item* : flex-basis

flex-basis permet de spécifier la longueur initiale d'un item.

flex-basis = valeur ; avec valeur :

- *un nombre* : définit la longueur de l'élément
ex : 10px [déconseillé], 5em,...
- *auto* : la longueur de l'item est calculé selon son contenu
- *initial* : la longueur de l'élément est fixée à sa valeur par défaut (permet d'annuler une longueur fixée auparavant)
- *inherit* : longueur héritée du parent (ex : quand le parent est à la fois un conteneur et un item du grand-parent)

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

40

propriétés CSS d'un *flex-item* : flex-grow

flex-grow permet d'indiquer si un élément peut grossir si besoin (s'il reste de l'espace à remplir). La valeur est un entier positif indiquant la proportion maximale par rapport aux autres éléments.

sans flex-grow

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

avec flex-grow : 2
sur les éléments
en texte bleu

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

41

propriétés CSS d'un *flex-item* : flex-shrink

flex-shrink permet d'indiquer si un élément peut rétrécir si besoin (s'il manque de l'espace). La valeur est un entier positif indiquant le rétrécissement maximal par rapport aux autres éléments.

(pas d'exemple, car pas beaucoup de sens avec du texte...)

42

propriétés CSS d'un *flex-item* : align-self

align-self permet de surcharger l'alignement défini au niveau du conteneur (align-items) pour les éléments concernés.

Aucun élément n'a de propriété align-self :

align-self = *valeur* ; avec valeur :

- *flex-start* : le "point de départ" de l'item ("grand-mère") est fixé au point de départ de la ligne
- *flex-end* : le "point d'arrivée" de l'item est fixé au point d'arrivée de la ligne
- *center* : l'item est centré sur la ligne
- *baseline* : l'item est aligné sur la baseline du conteneur
- *stretch* : l'item est positionné et redimensionné pour remplir la ligne du conteneur

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

Il était une fois une petite fille qu'on appelait le petit chaperon rouge et qui allait voir sa grand-mère en passant par les bois.

43

Exemple d'utilisation de Flexbox

En spécifiant un article comme un conteneur flexible, son contenu (un header, un footer, un contenu principal et deux "aside") peut facilement s'adapter à la taille de l'écran.

Ici, à droite la version pour smartphone et en dessous, la version pour PC.

Le code dédié à cette adaptation est présenté sur la diapositive suivante, l'exemple complet est sur Chamilo.

The diagram illustrates the use of Flexbox. It shows two views of a page layout: a smartphone view on the left and a desktop view on the right. In both views, there is a header (orange), a main content area (blue) containing text, and two aside sections (yellow and pink). Below the main content is a footer (green). In the smartphone view, the aside sections are positioned to the sides of the main content. In the desktop view, the aside sections are stacked vertically below the main content, demonstrating how Flexbox adapts to different screen sizes.

44

Profiter de la tolérance aux erreurs

Le langage CSS est tolérant aux erreurs : lorsqu'il n'arrive pas à interpréter une directive, il l'ignore et applique les autres.

Exemple : border-radius, arrondit les angles d'un élément sur un navigateur récent, ignorée par les navigateurs anciens

Bonne pratique :

- définir une directive "simple" pour les navigateurs anciens
- surcharger par une directive plus élaborée destinée uniquement aux navigateurs récents.

Exemple :

```
#maSection {  
    background-color : #EEEEEE ;  
    background: linear-gradient(180deg, red, blue);  
}
```

Bibliographie : J. Patonnier & R. Rigot, *Projet Responsive Web Design*, Eyrolles, 2013

45

Gérer la haute résolution

Les écrans de type Retina ont une densité de pixels deux fois supérieure à la normale. Ceci ne pose aucun problème en CSS, cette différence étant gérée par les "pixels OS".

Le seul problème se situe sur certaines images. En effet, les images sont (souvent) dimensionnées pour avoir une résolution proche de celle des écrans ciblés (de façon à minimiser leur taille et accélérer leur chargement). Mais dans ce cas, les écrans à haute résolution doivent interpoler les pixels de l'image pour générer un rendu avec une résolution suffisante, ce qui conduit à ajouter du "flou" (comme ci-dessous) => il faut prévoir différentes images si c'est un problème !



En résolution simple

En haute densité de pixels

46

Le problème des écrans à haute résolution (2/2)

La solution consiste alors à utiliser deux résolutions pour la même image : l'une pour les écrans "normaux" et l'autre avec une résolution double pour les écrans à haute résolution.

```
.exemple {  
    background-image: url(monFond.png);  
    height: 300px;  
    width: 200px;  
}  
  
@media screen and (-webkit-min-device-pixel-ratio: 1.5),  
      screen and (-moz-min-device-pixel-ratio: 1.5),  
      screen and (-o-min-device-pixel-ratio: 3/2),  
      screen and (min-device-pixel-ratio: 1.5) {  
  
    .exemple {  
        background-image: url(monFond2x.png);  
    }  
}
```

Bibliographie : J. Patonnier & R. Rigot, *Projet Responsive Web Design*, Eyrolles, 2013

47

Les bonnes pratiques (1)

Du côté du style :

- Mettre le **CSS dans un fichier à part** ;
- Définir **un style par défaut** et le surcharger pour une configuration donnée ;
- Les styles communs (polices, couleurs, background... tout ce qui n'est pas positionnement) est **factorisé** dans une feuille appelée sans condition (éviter les redondances !)
- Sur petit écran : **supprimer les blocs qui ne sont pas indispensables** (par exemple, conserver le aside de définition du vocabulaire, supprimer celui de l'histoire de la notion présentée)
- **Eviter les positionnements absolus** (ex : une div est positionnée en left : 900px, que se passe-t-il si l'écran a moins de 900px ?) ;
- Conserver une **cohérence entre les différentes présentations** (l'utilisateur ne doit pas être perdu quand il vient sur le site avec différents dispositifs)
- **préférer width:auto; à width:100%**; (dans ce dernier cas, la bordure et la marge de l'élément "dépassent" du conteneur)

Bibliographie : J. Beranger, *7 conseils pour réaliser un site en Responsive Web Design*, mars 2015

48

Les bonnes pratiques (2)

Du côté de la structure et du contenu :

- **Eviter trop d'imbriques structurelles**, afin de faciliter l'application des media queries (même si personne ne semble à même d'expliquer ce que signifie "trop") ;
- **Adapter les fonctionnalités** du site aux capacités du dispositif. Par exemple, un numéro de téléphone sera affiché comme un simple texte sur un ordinateur, mais comme un bouton cliquable sur un téléphone (pour permettre de passer l'appel d'un simple clic) ;
- **Réserver les tableaux** à leur utilisation normale (structurer des données et non faire de la mise en page : ils sont peu "responsive" et perturberont l'adaptation)
- **Eviter les éléments flottants** (difficiles à configurer correctement pour les différentes présentations)
- **Préférer une image recadrée** à une image redimensionnée, afin de montrer ce qui est important dans l'image sans trop réduire cette partie
- **Faire des tests** avec des utilisateurs différents, pour vérifier que votre design fonctionne (en particulier l'iconographie, parfois incompréhensible pour des utilisateurs novices)

Bibliographie : J. Beranger, *7 conseils pour réaliser un site en Responsive Web Design*, mars 2015

49

Les bonnes pratiques (3)

Du côté de la portabilité de votre site :

- **Valider son site** avec le W3C (cela augmente les chances que votre code soit compris par les navigateurs) ;

Bibliographie : J. Beranger, *7 conseils pour réaliser un site en Responsive Web Design*, mars 2015

50

L'amélioration progressive

L'amélioration progressive consiste à enrichir le site en détectant proactivement les fonctionnalités acceptées par le navigateur.

On peut citer aussi un cas qui devient de plus en plus pertinent aujourd'hui : un comportement différent si le terminal s'utilise avec un écran tactile. En effet, bien qu'on trouve chez certains développeurs une volonté de considérer les écrans tactiles comme « mobiles », cette approche est de moins en moins satisfaisante, surtout depuis que les tablettes font, en mode paysage, une largeur équivalente à certains écrans d'ordinateur ;

E.g. pas de survol sur écran tactile ==> quid des tooltips ?

La solution consiste donc à prévoir un comportement à manipuler avec la souris dans un premier temps (qui puisse être géré par les navigateurs anciens), puis de l'améliorer lorsque la capacité à interagir avec un écran tactile est détectée, en remplaçant le comportement par un autre ou en l'enrichissant.

Bibliographie : J. Patonnier & R. Rigot, *Projet Responsive Web Design*, Eyrolles, 2013

51

Les limites du Responsive Web Design

Le Responsive Web Design permet de maintenir « au mieux » l'expérience utilisateur.

Cependant...

- Attention à la bande passante pour les dispositifs mobiles (en attendant la généralisation de la 4G puis de la 5G !) : les éléments masqués par le CSS sont pourtant téléchargés ; les images redimensionnées sont aussi chargées à leur taille d'origine ; l'ensemble des calculs nécessaires pour l'adaptation coûte cher en CPU et en temps de réponse. Il est alors parfois plus pertinent de développer une application ou un site dédié pour les dispositifs mobiles (ou à l'inverse de ne charger qu'un contenu minimal et de charger le reste en JavaScript si c'est utile) : une seconde de temps de chargement en plus, ce sont des utilisateurs qui s'en vont sans attendre.
- Le Responsive Web Design ne permet QUE l'adaptation à la taille d'écran. Il ne permet pas d'autres adaptations qui pourraient être pertinentes (écran tactile [par de survol avec la souris...], luminosité ambiante, puissance de calcul du dispositif, problèmes de vision de l'utilisateur,...).

Bibliographie : G. Deruette, *Les grands principes du Responsive Web Design*, Programmez n°162, avril 2013

52

RWD demain : l'*Adaptive Design*

Alors que le responsive consiste à gérer uniquement les différentes tailles d'écran, l'Adaptive Design s'applique plutôt à une catégorie de sites dont le design (fonctionnel ou graphique) change en fonction des capacités du navigateur et cherche à prendre en compte certaines capacités du dispositif ou du système d'exploitation :

- Est-il capable d'exécuter du JavaScript ?
- Est-il capable de stocker des données côté client ?
- Est-il capable de représenter des blocs avec des dégradés en CSS ?
- ...

Les capacités du site Web sont alors modifiées en fonction de ce qu'il est possible de faire (utilisation en ligne et/ou hors ligne, diminution / augmentation des fonctionnalités proposées [ex : un tableau de bord synthétisant une activité est plus lisible sur grand écran et ne sera pas proposé sur un Smartphone]).

Bibliographie : J. Patonnier & R. Rigot, *Projet Responsive Web Design*, Eyrolles, 2013

53

RWD après-demain : les IHM plastiques

L'adaptation aux capacités du dispositif n'est suffisante pour une utilisation vraiment satisfaisante : à (long ?) terme, il sera aussi nécessaire de prendre en compte

- **les autres dispositifs de l'utilisateur**
 - et permettre d'avoir une partie de l'interface sur Smartphone et l'autre sur le téléviseur (2 modes graphiques) ou le Smartphone et la chaîne Hi-Fi (1 graphique, 1 vocal),...
- **l'environnement** pour adapter l'interface selon...
 - la luminosité (différencier le contraste en plein soleil et la nuit dans le lit),
 - le bruit ambiant (utiliser le vibreur plutôt qu'un son lorsqu'il y a du bruit),
 - le contexte social (éviter les sonneries lors d'une réunion)
 - ...
- **l'utilisateur** et ses préférences, habitudes et capacités
 - présenter en premier la page la plus utilisée du site,
 - éviter l'information au centre de l'écran si l'utilisateur ne le voit pas (tâche maculaire),
 - utiliser les commandes vocales pour un utilisateur paralysé,
 - choisir un style graphique plus apprécié ou plus facile à lire,
 - ...

Pour parvenir à prendre en compte toutes ces contraintes, l'IHM sera sans doute générée à la volée en fonction de la connaissance du contexte d'usage (dispositif(s) + environnement + utilisateur)

Bibliographie : E. Céret et al, *Plasticité des IHM par l'Ingénierie Dirigée par les Modèles*, Génie logiciel n° 105, p. 45-51.

54