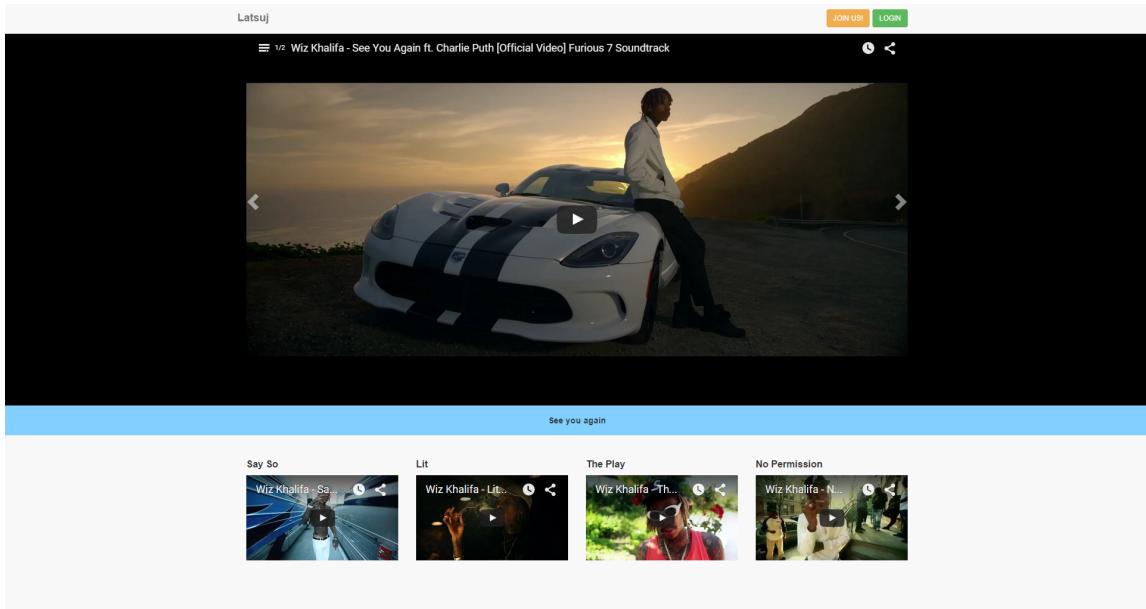


Bootstrap vs Polymer

DOSSIER D'ANALYSE DES DIFFÉRENCES



JUSTAL KEVIN
2015

Justal Kevin - justal@polytech.unice.fr - SI5 - IHM

Enseignant :
Anne Marie Dery - dery@polytech.unice.fr

Table des matières

1	Comment avez vous réalisé l'exemple ?	3
1.1	Structure du site	3
1.2	Images flexibles	4
1.3	Media queries ou Points de rupture	4
1.4	Grille de positionnement	9
2	Bootstrap ou Polymer	11
2.1	Compatibilité	11
2.2	Documentation	11
2.3	Vitesse de chargement	11
3	L'exemple de A à Z	12
4	Comment teste-t-on les capacités d'adaptations ?	13
4.1	Outils de test	13
4.2	Éléments adaptatifs	13
5	Documentations, Outils, liens utiles	14
6	Difficultés rencontrés	19
7	Simple trouvaille d'optimisation	19

1 Comment avez vous réalisé l'exemple ?

Pour réaliser mon exemple, je me suis dans un premier temps intéressé à la définition même d'un site web adaptatif. Le principe du RWD (*responsive web design*) ou site web adaptatif dans la langue de molière consiste à s'appuyer sur l'usage des *Media queries* que l'on peut aussi appeler « points de rupture », de grilles de positionnement et d'images flexibles pour rendre un site adaptable à son support.

Pour développer, je suis parti de la version ordinateur à une taille relativement imposante, puis j'ai remis en forme les éléments à mesure que la largeur de l'écran diminuait voire je les supprimais totalement. J'ai essayé autant que faire se peut d'ajouter un maximum d'éléments différents. On retrouve ainsi des vidéos, un menu, des tableaux, des images, du texte, des éléments interactifs comme des boutons... Il y a donc l'ensemble des éléments que l'on peut retrouver sur n'importe quelles sites lambda.

1.1 Structure du site

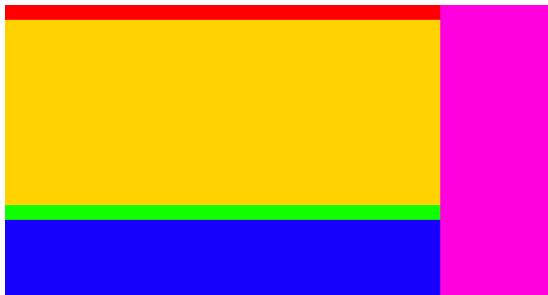


FIGURE 2 – Structure du site

que l'on peut ou non affiché grâce à un bouton se trouvant dans la zone rouge. Si le menu est rentré, la zone violet n'existe plus et on se retrouve avec seulement quatre blocs qui prendront la totalité de la fenêtre comme on peut l'observer sur l'image suivante :

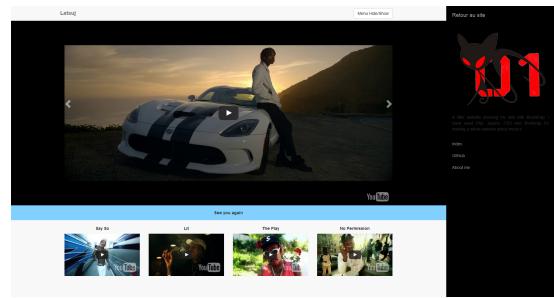
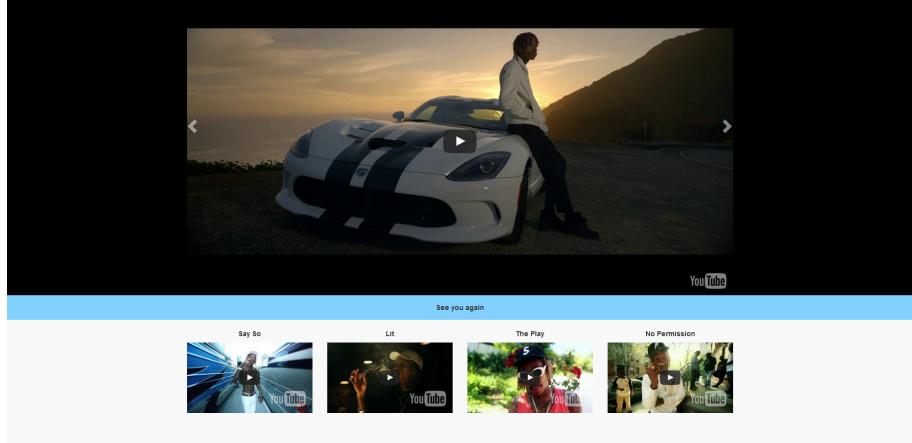


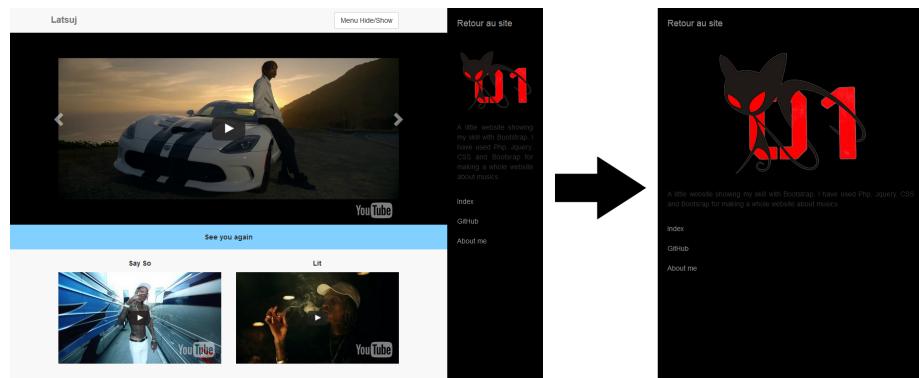
FIGURE 1 – Le site pour PC

Les deux technologies que j'ai choisies partagent la même structure de base. C'est à dire un découpage en cinq grands blocs. Sur la figure 2, le bloc en rouge représente la barre principale. En jaune, on retrouve la zone de lecture principale des clips musicaux. En vert, il s'agit d'un bouton permettant d'ouvrir un bloc d'information concernant le clip de la zone jaune. En bleu, il s'agit simplement d'autres clips musicaux en rapport avec le chanteur chantant dans le clip de la zone jaune. En rose, on retrouve un menu



1.2 Images flexibles

Pour observer précisément les différents principes de RWD sur notre site, nous allons nous intéresser à certaines zones du site. Pour commencer, nous allons d'abord nous intéressé au menu (zone violet). Cette dernière illustre à la perfection le principe d'image flexible que l'on attend d'un site adaptatif. Qu'est ce qu'une image flexible ? Il s'agit d'une image qui se redimensionnera en fonction de la taille de la fenêtre. Pour réaliser cela, il faut simplement donner à une image quelques attributs CSS. Il faut lui spécifié de prendre toute la longueur disponible dans le bloc où elle se trouve et de prendre une hauteur automatiquement en fonction de son rapport. En CSS, cela se traduit par : « `width:100%;height:auto;` ». Avec Polymer, j'ai créé un object image qui ajoutait automatiquement des images responsives. Sous Bootstrap, c'est on ne peux plus simple, une classe existe nommé « `img-responsive` » et il suffit de la rajouter sur une balise IMG. Cela permet d'obtenir le résultat suivant lorsque l'on réduit la taille de la fenêtre :



Dans mon exemple ci-dessus, on remarque bien que l'image s'adapte à la taille de la fenêtre. On remarque de plus que le site est totalement différent, il s'agit là de l'application des points de rupture ou Media queries dont je parlais précédemment et que nous allons voir au prochain point.

1.3 Media queries ou Points de rupture

Les **media queries** ou points de rupture sont des règles CSS appliquées en fonction de la largeur de l'écran. Ce sont ces différentes largeurs qui sont appelées « **points de rupture** », elles correspondent à un besoin de modifier la page à partir d'un seuil critique pour la facilitation de la navigation et de la lecture du contenu. Pour que cette définition soit clair, nous allons montrer leur application sur le site.

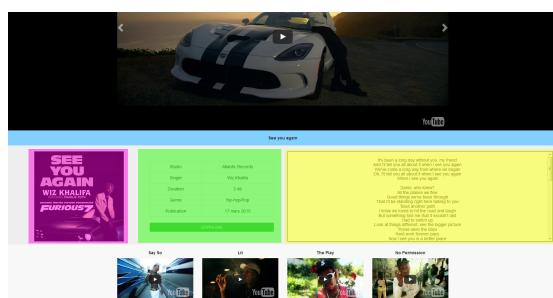


FIGURE 3 – Les blocs

Sur la figure 3, j'ai colorié les différents blocs chacun d'une couleur différente. Ces blocs sont simplement représentés par des balises DIV avec des règles CSS fonction de la taille de l'écran. Prenons l'exemple du bloc violet, ce bloc fait exactement 25% du conteneur qui lui prend 100% de la longueur de l'écran. Lorsque l'on réduit la fenêtre jusqu'à un certain point, ce bloc prendra alors 50% sur ces 100%. Puis si l'on continue à réduire la fenêtre, ce bloc disparaît.



Sur Bootstrap, il est assez simple de réaliser ceci. Cet technologie inclue un système de grille relativement bien ficellé. Il faut dans un premier temps, définir notre zone où on souhaite réaliser le découpage en utilisant la classe « container » dans une balise DIV :

```
<div class="container">
</div>
```

On a ainsi spécifié où se trouvait notre zone par dessus laquelle nous allons poser une grille. J'entrerais dans les détails au chapitre suivant, pour l'instant il faut voir cela comme une subdivision en 12 partie de la partie rouge sur la figure 4. Comme je l'ai déjà spécifié précédemment, l'image dans la partie rouge ne représente que 25% de cette zone lorsque l'écran est supérieur à 1500px (valeur choisie par mes soins). Pour ensuite, spécifié à Bootstrap que nous allons couper notre container en plusieurs parties, il faut utiliser la classe « row » :

```
<div class="container">
  <div class="row">
    ...
  </div>
</div>
```



FIGURE 4 – Zone du container

Nous allons maintenant entrer dans le vif du sujet, comme dit plus haut, le container est divisé en 12 parties que j'appellerais dans la suite colonne. On peut fixer le nombre de colonne que peut prendre un élément. Reprenons et analysons précisement la figure 3. Sur cette dernière, la zone jaune prend 6 colonnes et les 6 colonnes restantes sont réparties équitablement entre la zone violette et verte. Niveau code, on spécifie cela en utilisant les classes « col-xs-X », « col-sm-X », « col-md-X » ou « col-lg-X ». Xs, sm, md et lg représentent la taille minimum de l'écran pour que les règles que ces classes contiennent prennent effets. Tandis que le X dans ces classes représente tout simplement le nombre de colonnes que prendra l'élément utilisant ces classes. Je reviendrais un peu plus loin sur ce point, pour l'instant, il est plus important de comprendre la différence entre les classes xs, sm, md et lg :

```
<div class="container">
  <div class="row">
    <div class="col-md-12 col-lg-6">
      <div class="row">
        <div class="hidden-xs hidden-sm col-md-6 col-lg-6">
          ...
        </div>
        ...
      </div>
    </div>
    ...
  </div>
</div>
```

Ce bout de code est suffisant pour comprendre et expliqué clairement les mouvements de l'image violette dans la figure 3. Le texte en rouge précise qu'à grande taille lg (supérieur à 1500px), la zone violette et verte ont 50% du container, soit 6 colonnes : « col-lg-6 ». Le texte en bleu est la

balise DIV directe qui contient l'image violette. Ces classes spécifient qu'à grande taille, ce bloc ne peut excéder 6 colonnes, soit 50%. Or, 50% de 50% fait bien 25% comme je l'avais dit plus haut. Cependant cela n'est valide que lorsque la fenêtre est supérieur à 1500px. En dessous, les autres classes prennent le relais.

Pour comprendre précisément quelles sont les classes qui sont actives à une grandeur de fenêtre donné, il faut simplement retenir les dimensions que représentent les sigles xs, sm, md, lg. Les dimensions qui sont affichées ne sont pas celles par défauts de BootStrap mais celles que j'ai utilisé pour mon site. J'ai changé les points de rupture pour que mon menu puisse s'insérer joliment sur mon site. Sans cela, il se pouvait que le bouton de la barre de menu chevauche le menu, ce qui donnait un résultat plutôt moche. La seule manière de régler le problème a été de construire une version personnalisé de Boostrap. Il est possible d'effectuer cette opération sur le site officiel de Bootstrap.

xs	600px
sm	900px
md	1200px
lg	1500px

FIGURE 5 – Points de rupture

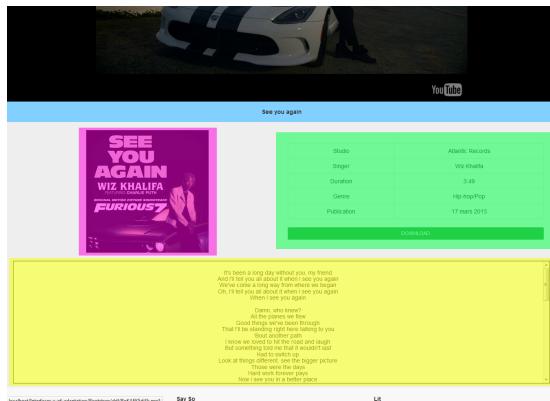


FIGURE 6 – 50% de la fenêtre

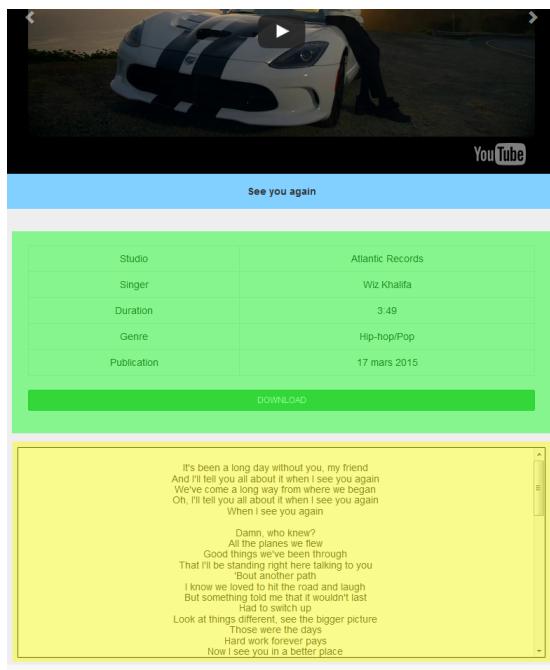


FIGURE 7 – L'image a disparu

Dans notre cas, si la fenêtre est inférieur à 1500px alors les classes lg ne sont plus utilisée. Prenons une fenêtre de 1300px, si l'on regarde le tableau de la figure 5, on remarque que pour toutes les dimensions situés entre 1200 px et 1500 px md est la classe qui est actif. On reprend notre bout de code précédent et on remarque que le texte en rouge mentionne « col-md-12 ». Cela signifie que cette div qui représente la zone violette et verte de la figure 3 prendra 100% du container. On regarde notre texte en bleu et on lit « col-md-6 ». L'image violette prendra donc 50% de 100%, elle prendra donc la moitié de la longueur de la fenêtre.

Maintenant, si nous continuons de réduire la fenêtre et atteignons une taille inférieur à 1200px, nous arrivons dans les classes sm. Si on regarde le texte en bleu dans le bout de code précédent, on voit qu'il est écrit « hidden-sm ». Cette classe signifie que le nombre de colonne prise par cet élément sera réduit à zéro. Cela se traduit par une disparition complète de l'élément comme on peux le voir sur la figure 7. Notons au passage que je n'ai pas précisé dans le texte en rouge le nombre de colonne pour les dimensions sm et xs. Je n'ai pas eu à le faire car Bootstrap utilise les règle CSS pour réaliser ces modifications. Plus précisement, Bootstrap utilise l'héritage et le cascading de CSS pour pouvoir réécrire des règles sur des règles. Si on ne défini pas de xs, alors les règles CSS seront celles de sm. Si elles aussi ne sont pas définis, alors ce seront celles de md qui seront prise en compte... Cela ne constitue pas une faute mais j'aurais pu omettre le « col-md-6 » dans mon texte bleu et cela donnerais toujours le même résultat.

Nous avons vu comment mettre cela en place sur Bootstrap, on peut évidemment faire la même chose avec Polymer. Le résultat obtenu est le même mais la manière d'y arriver est différente. Cependant, notons que cela est totalement contraire à l'idée de Polymer. J'ai discuté de cela avec Monsieur Bidelman lui-même qui est un des fondateurs de Polymer et qui travaille actuellement chez Google. Le but est de créer des éléments réutilisables. Or à partir du moment où l'on utilise des médias queries, l'élément devient ancré au site qui les utilisent. Prenons l'exemple précédent, pour que l'image disparaît à un certain point de rupture, j'ai créé un élément qui change son attribut « display » lorsque l'on atteint ce fameux point.

```
<dom-module id="my-col-hidden-900">
  <template>
    <style>
      @media (max-width :900px) {
        :host {
          display : none ;
        }
      }
    </style>
    <content></content>
  </template>
  <script>
    Polymer(
      is : "my-col-hidden-900"
    );
  </script>
</dom-module>
```

On a donc créé une nouvelle node HTML nommée my-col-hidden-900. Le code en bleu est la partie la plus importante. Elle spécifie que le code se situant entre les balises « my-col-hidden-900 » aura l'attribut « display :none » si la taille de la fenêtre est inférieur à 900px. Autrement dit, tous le contenu à l'intérieur ne sera pas affiché.

Dans le même style, j'ai créé un élément du DOM dont la longueur s'adapte en fonction la largeur de la fenêtre. La longueur ainsi que le position naturel des balises DIV avec l'attribut « float :left » permet de réaliser un semblant de grille sous Polymer :

```
<dom-module id="my-col-25-100">
  <template>
    <style>
      :host {
        position : relative ;
        min-height : 1px ;
        padding-left : 15px ;
        padding-right : 15px ;
        float :left ;
        width :25% ;
        box-sizing : border-box ;
      }
      @media (max-width :1500px) {
        :host {
          width :50% ;
        }
      }
    </style>
  </template>
  <script>
    Polymer(
      is : "my-col-25-100"
    );
  </script>
</dom-module>
```

```

@media (max-width :900px) {
  :host {
    width :100%;
  }
}
</style>
<content></content>
</template>
<script>
Polymer({
  is : "my-col-25-100"
});
</script>
</dom-module>

```

Ce module permet d'ajouter une nouvelle node HTML nommé « my-col-25-100 ». Ce dernier permet de faire la division et le repositionnement des clips musicaux en bas de page. J'ai colorié chacun des éléments utilisant ce module sur la figure 8. Sur cette figure, la dimension de la fenêtre est supérieur à 1500px. Dans le code précédent, les règles en bleu sont celles actives. On remarque que la taille (« width ») est spécifié à 25% du container. Chacun des clips prend donc 25%, ce qui apparaît bien sur la figure à droite. Ensuite, si l'on réduit la fenêtre, nous arriverons au point d'ancrage à 1500px. Les règles appliquées seront donc celles écrites en bleu puis en rouge. Les règles rouges donneront de nouvelles valeurs aux règles en bleu, elles seront réécrites. Dans notre cas, la longueur maximal que pourra prendre un de nos clips musicaux sera de 50% du container. Ainsi, lorsque 100% du container est pris par 2 vidéos, les suivantes se placeront en dessous. Maintenant, imaginons que nous continuons à réduire la fenêtre jusqu'au point de rupture à 900px, chaque vidéos prendra la totalité de la largeur de la fenêtre. L'image ci-dessous montre le résultat obtenu via ces points de rupture avec les modules précédents :

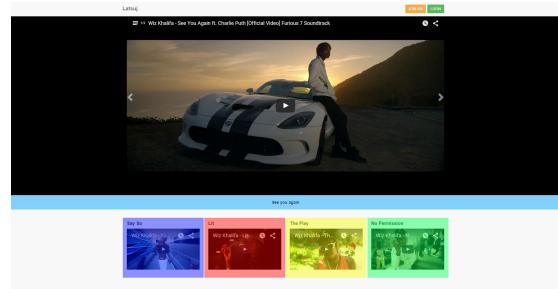
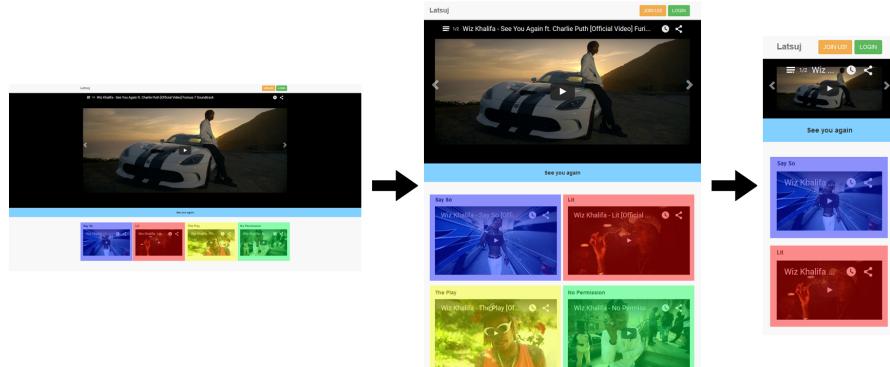


FIGURE 8 – Les 4 blocs



À travers ces différents exemples, nous avons pu voir comment faire des points de ruptures sous les deux technologies différentes. Sur Bootstrap, il est bien plus simple d'utiliser ces points de ruptures. L'utilisation des classes xs, sm, md et lg rendent le principe facile à comprendre. Cependant, nous sommes limités par Bootstrap à suivre ces quatre points de ruptures. Si le site contient de nombreux autres points de ruptures, Bootstrap perd de sa superbe car cela reviendrait à

faire du CSS classique. Sur Polymer, il faut bien comprendre les média queries (points de ruptures) pour pouvoir les utiliser. Comme Polymer fractionne notre code, on se retrouve avec du CSS et donc des points de ruptures qui se retrouvent éparpillé dans le code. En cas de changement de points de rupture, la maintenance peut vite devenir difficile.

1.4 Grille de positionnement

Les grilles de positionnements consiste en un dimensionnement relatif des différents blocs de la page. J'ai déjà partiellement évoqué ce point. Je vais essayer de me concentrer ici sur ce point en particulier. Comme dit dans le chapitre précédent, les grilles se placent dans des containers. Qu'est qu'un container ? Sur la figure 9, j'ai représenté en violet, le container du bas de la page. On peut aussi l'obtenir en analysant le code avec Firebug (F12). Cet espace violet est la place maximale que peut prendre le contenu ici.

Lorsque l'on utilise la classe « row » dans Bootstrap, on divise ce container en 12 parties égales. Avec Polymer, le principe que j'ai développé est globalement le même cependant j'ai dû reconstruire ces divisions de A à Z via les média queries et les attributs CSS (en particulier : width). Sur la figure 10, j'ai représenté chacune des divisions d'une couleur différente. Ensuite, on peut définir combien de colonnes ou divisions pourra prendre un élément. (voir les explications du chapitre précédent).

Pour les clips musicaux en bas de page, j'ai défini pour chacun d'entre eux 3 divisions. Sur Bootstrap, cela se traduit par « col-XX-3 » où XX représente tout simplement les points de rupture que j'ai évoqué dans le chapitre précédent. Avec Polymer, pour réaliser cela, il faut imbriqué deux modules ou balises l'une dans l'autre. La première servira de container pour réaliser la même chose que sur la figure 9 tandis que la deuxième servira à réaliser les divisions comme sur l'image de droite.

Le point important est que les éléments des grilles sont définis comme « float :left » que ce soit sur Bootstrap ou sur Polymer. Ce qui veut dire qu'un élément, ce positionnera toujours le plus à gauche possible sur la même ligne si il y a la place disponible, sinon l'élément se positionnera sur une nouvelle ligne. De même, il y a toujours 12 colonnes sur une ligne peu importe la taille de la fenêtre. Si jamais les éléments dépassent le nombre de colonnes disponibles sur la ligne, il se place automatiquement sur la ligne en dessous.

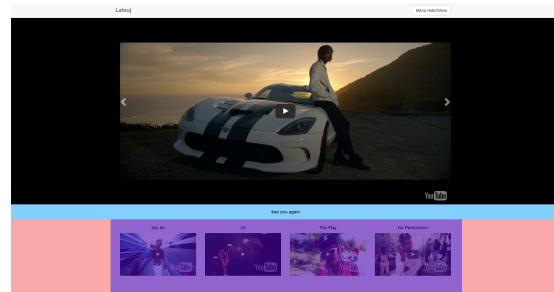


FIGURE 9 – Le container du bas de page



FIGURE 10 – Les 12 divisions d'un container

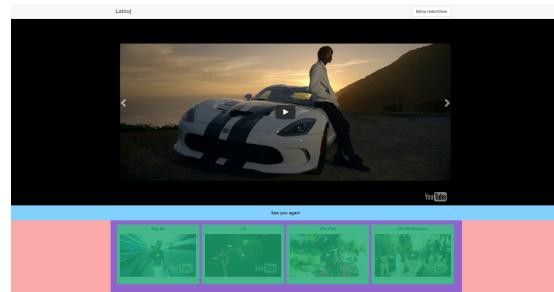
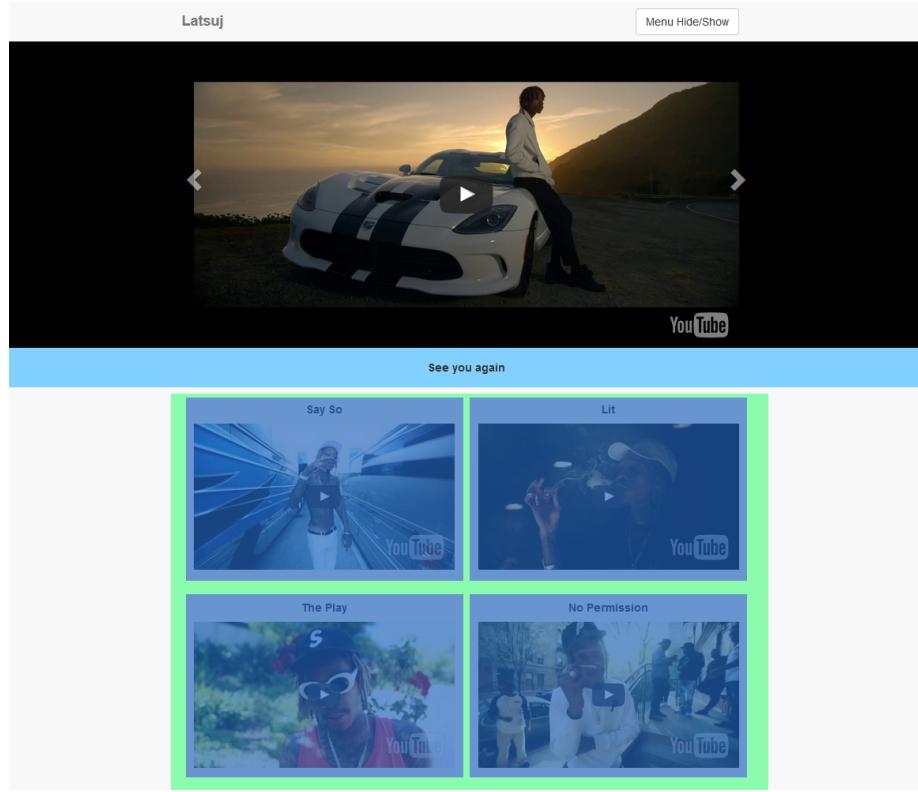


FIGURE 11 – Les 12 divisions d'un container



Comme vue dans les exemples précédent, les clips musicaux du bas de la page étaient affiché sur 1 ligne. Sur Bootstrap, cela équivaut à utiliser la classe « col-XX-3 ». Il y a 4 vidéos qui utilisent chacune 3 divisions. Sur Polymer, on précisera que le bloc utilise 25% du container en CSS avec l'attribut « width ». Maintenant sur l'image ci-dessus, il y a deux lignes. Chacune utilise un système de 12 colonnes. La différence réside simplement dans le nombre de colonne qu'on a permis à l'élément d'utiliser. Ici, j'ai utilisé la classe « col-XX-6 » sur Bootstrap. Pour indiquer que chaque vidéo prendrait 6 colonnes. Comme il y a 4 clips, les deux premières vidéos ont rempli la première ligne puis la troisième vidéo est automatiquement passé à la ligne en dessous pour compléter le container.

Ainsi, nous avons un très gros aperçu des moyens et technologies utilisé dans mes sites pour en faire un site suivant les notions de web adaptatif. Les deux technologies, Bootstrap et Polymer permettent de faire la même chose. Mais quelle technologie est la plus simple ? Quelle technologie permet de réaliser le plus rapidement un site responsive ? Quelles sont les contraintes de chacunes des technologies ? Quels sont leurs avantages respectifs ? C'est ce dont je discuterai dans le chapitre suivant.

2 Bootstrap ou Polymer

2.1 Compatibilité

Il s'agit sans doute de la première chose à comparer. Bootstrap est supporté par presque tout les navigateurs, Bootstrap n'est pas compatible avec Opéra sur téléphone et Safari sur Windows. Tandis que Polymer n'est compatible totalement qu'avec Google Chrome. Pour les autres navigateurs, Polymer est partiellement supporté ou encore n'est pas du tout supporté comme sur Internet explorer. En s'intéressant seulement à nos utilisateurs, on peut dès le départ éliminer une des technologies. Polymer n'est clairement pas destiné à viser un large public. Notons de plus que les développeurs de Mozilla Firefox ont d'ores et déjà annoncé qu'il ne ferait aucun effort pour que leur navigateur supporte Polymer.

2.2 Documentation

La documentation est aussi un point fondamental d'une technologie. Elle permet de faire la liaison entre les utilisateurs et les nouvelles méthodes de programmation. Sur Bootstrap, la documentation est claire et concise avec à chaque fois un exemple claire et simple à comprendre. Sur Polymer, ce point est mauvais. J'ai passé de long moment à arpenter la documentation pour trouver certaines informations essentielles, ce qui représente une perte de temps non négligeable. Par exemple, le point d'ancrage du nouveau composant est indiqué par l'utilisation de la balise `<content>` dans le nouveau module. Cette information fondamentale ne se retrouve pas sur les exemples de base sur l'API (Application Program Interface) de Polymer.

2.3 Vitesse de chargement

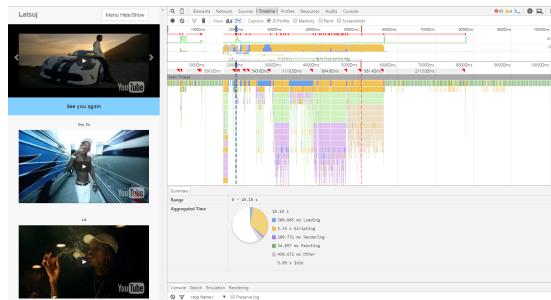


FIGURE 13 – Compatibilité des technologies

En terme de performance, j'ai utilisé la fenêtre de débogage de Google Chrome pour savoir la vitesse d'exécution de nos deux technologies. Les deux sites donnant le même résultat, il serait normal de croire que le temps nécessaire pour que le site soit prêt pour l'utilisateur soit identique. Malheureusement, ce n'est pas le cas. Pour Bootstrap, leur script est chargé en 3,33 secondes en moyenne tandis que celui de Polymer met 6,2 en moyenne. Ensuite, celui de Polymer doit effectuer ses traitements pour transformer les éléments. Tout ceci a évidemment un coût en terme de performance. Avec Polymer, le site met en moyenne 400 ms pour apparaître tandis qu'avec Bootstrap, cela tombe à 200 ms. Là encore, Polymer est encore en dessous de Bootstrap.

BOOTSTRAP						POLYMER					
	Chrome	Firefox	Internet Explorer	Opera	Safari		Graded	Partial	It's fine	Stable	Chromium (Windows)
Android	✓ Supported	✗ Unsupported	✗ Not Supported	N/A	N/A	Templates	✓	✓	✓	✓	✓
iOS	✓ Supported	N/A	N/A	N/A	N/A	Shadow DOM	✓	✗ dev flag	✗	✗	✗
Mac OS X	✓ Supported	✗ Unsupported	✗ Unsupported	✓ Supported	✓ Supported	System Elements	✓	✗ dev flag	✗	✗	✗
Windows	✓ Supported	✗ Unsupported	✗ Unsupported	✓ Supported	✗ Not Supported	Shadow DOM	✓	✗ dev flag	✗	✗	✗

FIGURE 12 – Compatibilité des technologies

3 L'exemple de A à Z

4 Comment teste-t-on les capacités d'adaptations ?

4.1 Outils de test

Pour tester l'affichage et l'adaptation de nos éléments à la fenêtre, il existe plusieurs voies envisageables. J'ai utilisé plusieurs d'entre elles pour effectuer mes tests. La première méthode a été de modifier la taille de la fenêtre sous windows.

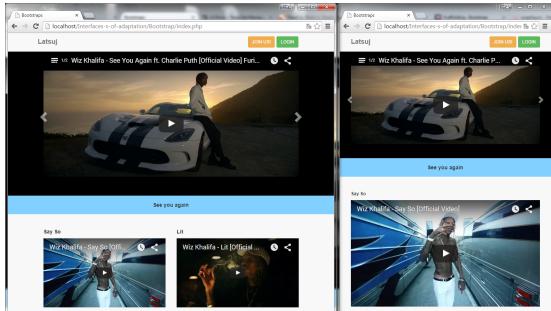


FIGURE 14 – Redimensionnement de la fenêtre sous windows

Polymer, on ne peut testé qu'avec quelques navigateurs dû au problème de compatibilité que j'ai évoqué précédemment.

Pour tester l'adaptation sur des appareils en particuliers, Google Chrome dispose d'un outil redimensionnant la fenêtre à la dimension exacte d'un appareil en particulier. J'ai utilisé cet outil pour voir le résultat sous différentes tablettes et téléphones actuellement sur le marché. Pour activer cet option sous le navigateur, il suffit d'appuyer sur F12 puis le bouton représentant un téléphone nommé « toggle device mode ».

L'exemple de la figure 14 montre le site à deux dimensions différentes. Comme on peut le voir, le site s'adapte bien à la largeur de la fenêtre. Sur la gauche, le site est plus grand. La taille plus grande permet de mettre deux clips vidéos l'un à côté de l'autre. Sur la droite, le site est plus petit et ne permet de n'afficher qu'un clip vidéo par ligne. Les autres éléments quant à eux se redimensionne pour s'adapter à la page. J'ai réalisé cette opération avec chacun des navigateurs avec Bootstrap. Le résultat obtenue est le même sur chacun des navigateurs. Avec

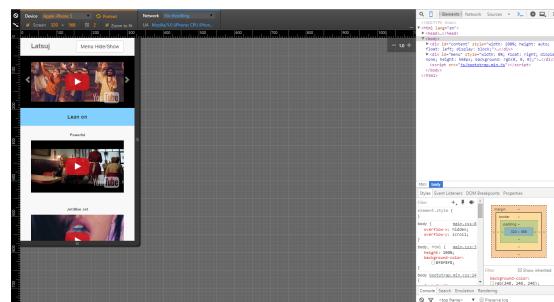


FIGURE 15 – Test au format iphone 5

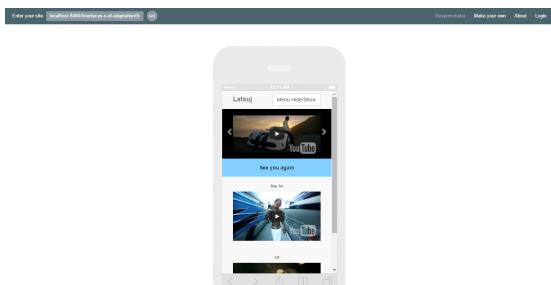


FIGURE 16 – Responsinator.com

4.2 Éléments adaptatifs

L'ensemble du site contient des éléments qui s'adaptent à la largeur de la fenêtre. Que ce soit le menu, la vidéo principale, les images, le texte, chaque élément à sa propre façon de se positionner, de se redimensionner suivant la largeur de la fenêtre.

Il est aussi possible de passer par des sites qui permettent de tester l'adaptation de nos sites comme responsinator.com. J'ai aussi testé mon site sous cet élément. On voit sur ce site un grand ensemble d'appareil avec notre site à l'intérieur. Le seul problème de cette méthode est qu'il faut charger X fois le même site, ce qui peut s'avérer plutôt long si le site en question est relativement lourd.

5 Documentations, Outils, liens utiles

Wikipédia

Le site d'où j'ai démarré mes recherches, il contient une bonne définition des sites RWD.
https://fr.wikipedia.org/wiki/Site_web_adaptatif

What is a responsible web design ?

Les liens suivant sont les articles ou vidéos que j'ai analysé pour écrire ce rapport.
<https://www.youtube.com/watch?t=133&v=iSY38POjLYc>

Ethan Marcotte

Le site du créateur du RWD qui montre la différence entre un site adaptable et un site responsive.
<http://alistapart.com/d/responsive-web-design/ex/ex-site-flexible.html>
<http://alistapart.com/d/responsive-web-design/ex/ex-site-linearize.html>

Responsible typesetting

Un article qui traite du responsible typesetting.
<http://blog.line0.eu/responsible-typesetting/>

Fitt's law

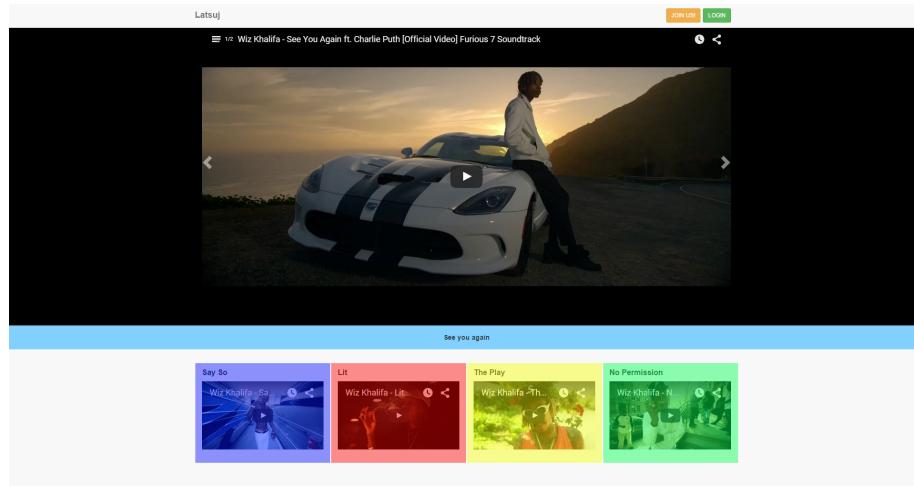
La description de la loi de Fitt
https://en.wikipedia.org/wiki/Fitts'_law
<http://webdesign.tutsplus.com/articles/applying-fitts-law-to-mobile-interface-design-webdesign-6919>

Media queries

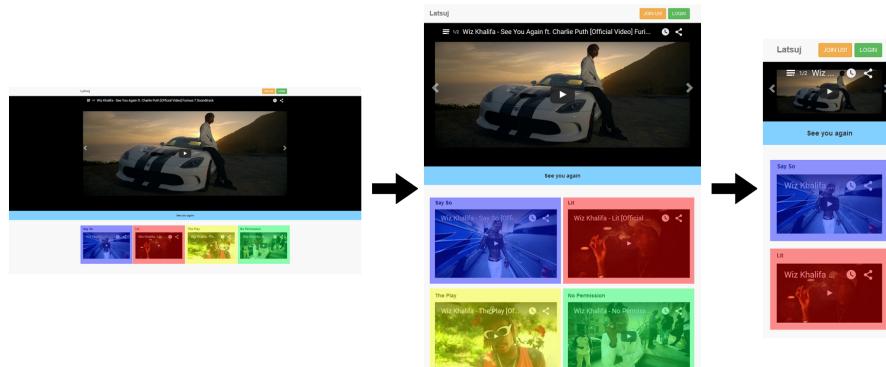
Les media queries sont intéressants mais limités. Le futur serait plutôt du côté des éléments queries.
<http://ianstormtaylor.com/media-queries-are-a-hack/>
<http://www.smashingmagazine.com/2013/06/media-queries-are-not-the-answer-element-query-polyfill/>

Mais cela n'est qu'un hack qui permet de faire ce que je désire. On peut le faire mais on ne devrait pas le faire. Le gros problème ici est que ce code est non maintenable. Si jamais je changeais de point de rupture par exemple, je devrais changer la valeur de ce média query. Si il n'y a que celui-là, ce n'est trop long et fatigant. Mais si le projet a quelques milliers de modules, cela risque d'être très long.

Ne pas oublier de detailler le changement de point de rupture.
Colorions chaque divisions de cette partie du site d'une couleur unique.



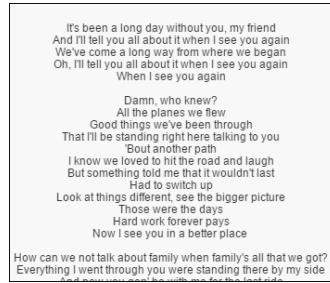
Si nous réduisons la largeur de la fenêtre, le contenu s'adaptera. Dans un premier temps, il n'y aura plus que deux blocs par ligne. Puis, si nous continuons de réduire la fenêtre, il n'y aura plus qu'un seul bloc par ligne et les deux derniers auront été caché.



Le site s'adapte donc aux dimensions de notre appareil ou fenêtre. À quoi cela peut-il bien servir ? Il serait plus adapté de se demander quels sont problèmes que cela résout-il ? Comme on le voit autour de nous, les ordinateurs ne sont plus les seuls éléments ou gadgets nous entourant, ils existent maintenant une innombrable quantité d'appareil informatique de tous types et de toutes dimensions. Il est important qu'un site internet ne laisse aucun utilisateur sur le bas coté. Comme il est impensable de concevoir une application ou un site internet pour chaque appareil, il faut donc faire un site qui puisse s'adapter suivant les dimensions de l'appareil.

Mais ce n'est pas tout, l'adaptation seul ne permet pas d'établir ce que l'on peut appeler un site web adaptatif. Le créateur de cette vision, Mr. Ethan Marcotte, a implementé un site (<http://alistapart.com/d/responsive-web-design/ex/ex-site-flexible.html>) qui s'adapte à la largeur de l'écran. Cependant, il pointe du doigt certains détails. Par exemple, lorsque l'on redimensionne la page, les éléments vont bel et bien se redimensionner mais les images et le texte à très basse résolution deviendront illisible. Il faut donc que les éléments se repositionnent dans la page afin que le contenu soit lisible et agréable à arpenter pour l'utilisateur. C'est pourquoi dans l'exemple ci-dessus, le nombre de bloc par ligne décroît au fur et à mesure que l'on réduit la fenêtre.

Le site est aussi *responsible typesetting*. La taille en pixel du texte varie suivant la taille de la fenêtre. Pour l'utilisateur, il est sans aucun doute plus agréable de pouvoir lire les paroles d'une chanson phrase par phrase. Or, si la taille du texte restait la même pour toutes dimensions de fenêtre, soit le texte serait illisible à une grande résolution, soit le site serait incommoder à basse résolution. Pour résoudre ce problème, une proportions a été spécifier pour l'ensemble des textes suivant la largeur de la fenêtre ou de l'appareil. Sur le site, on retrouve cette particularité sur plusieurs titres et sur les paroles comme nous pouvons le constater ci-dessous. À gauche, on retrouve les paroles des chansons sur téléphone portable tandis que à droite, on retrouve les mêmes paroles écrite avec une plus grande police sur tablette.



Pourquoi ais-je supprimé les deux derniers blocs sur la navigation à basse résolution ? Ce n'est pas une décision anodine. En supprimant ces deux blocs, j'améliore l'expérience de l'utilisateur sur deux aspects.

L'un est purement lié à la technologie, il est rare d'avoir un téléphone branché en Ethernet. Ceci implique que le débit moyen d'un utilisateur sur téléphone est souvent inférieur à celui d'un utilisateur sur ordinateur. J'évite ainsi à l'utilisateur sur téléphone de charger trop d'informations qui n'appartiennent pas au contenu principal de la page. Ce ne sont que des publicités pour les autres musiques du même chanteur. Deuxièmement, et c'est sans doute le point le plus important, suivant la **loi de Fitt**, que je suis sur téléphone ou ordinateur l'indice de difficulté doit rester le même. La loi calcul donne un indice de difficulté par rapport au temps requis pour aller rapidement d'une position de départ à une zone finale de destination. L'utilisateur doit arriver avec la même rapidité et la même facilité aux différentes parties du site. Si l'on regarde l'image sur la droite qui représente le site sur un téléphone BlackBerry, on remarque que le temps pour parvenir à la dernière vidéo en rapport avec le chanteur est aussi rapide sur le téléphone que sur ordinateur. Certes ce n'est pas la même, mais cela reste la dernière vidéo. Si j'avais juste réarrangé les choses, il aurait d'abord fallu descendre pour arriver à la dernière vidéo. Cela ne paraît pas beaucoup plus compliqué mais sans cela, le site se retrouverait complexifié d'après la loi de Fitts.

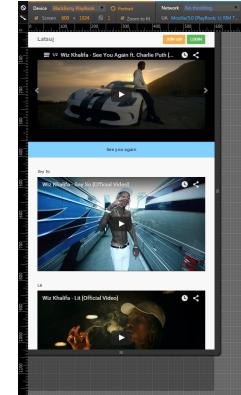


FIGURE 17 – Affichage du site sous BlackBerry (via Google Chrome).

6 Difficultés rencontrés

Le premier problème rencontré fut lorsque que j'essaya de coder une balise div de telle manière que celle-ci remplisse entièrement l'espace de l'application. Cette chose extrèmement simple n'est pourtant pas implémenté dans Bootstrap 3.0 et les versions supérieur alors que cela se trouvait dans les versions antérieur avec la class span. Après de longue recherches, il apparait donc impossible en pur Bootstrap de remplir un div à cent pour cent de la balise parent. De ce fait, j'ai du modifié le CSS pour réaliser le remplissage de la page. Pourquoi un tel choix des développeur de bootstrap ?

margin-bottom : Seriously ?

Compatibilite : WTF polymer !

min-height ? WTF do not work !OK parce que tous ces putain d'elements sont en inline et non en block...Ok l'erreur

Suivre un ordre pour appeler les modules au depart, les enfant en premier.

encapsulation des elements ? content :X Merci la doc....pourrie.

Le carousel une horreur sur Polymer....

Bootstrap, position fixed et col does not work.

J'ai du recompiler le bootstrap.

7 Simple trouvaille d'optimisation

En farfouillant sur les documentations de Bootstrap, je suis tombé sur une optimisation qui a retenu mon attention. Une chose simple et pourtant efficace que je ne faisait pas moi non plus. Les developpeurs de Bootstrap mettent toujours les scripts javascript en fin de page afin d'accelerer le chargement de la page. Cela peut paraître stupide comme remarque mais je tiens à m'en souvenir, j'en fait donc par dans mon document.

Petite astuce, enlever les ; sur le dernier elements de css pour gagner un caractere de lecture.