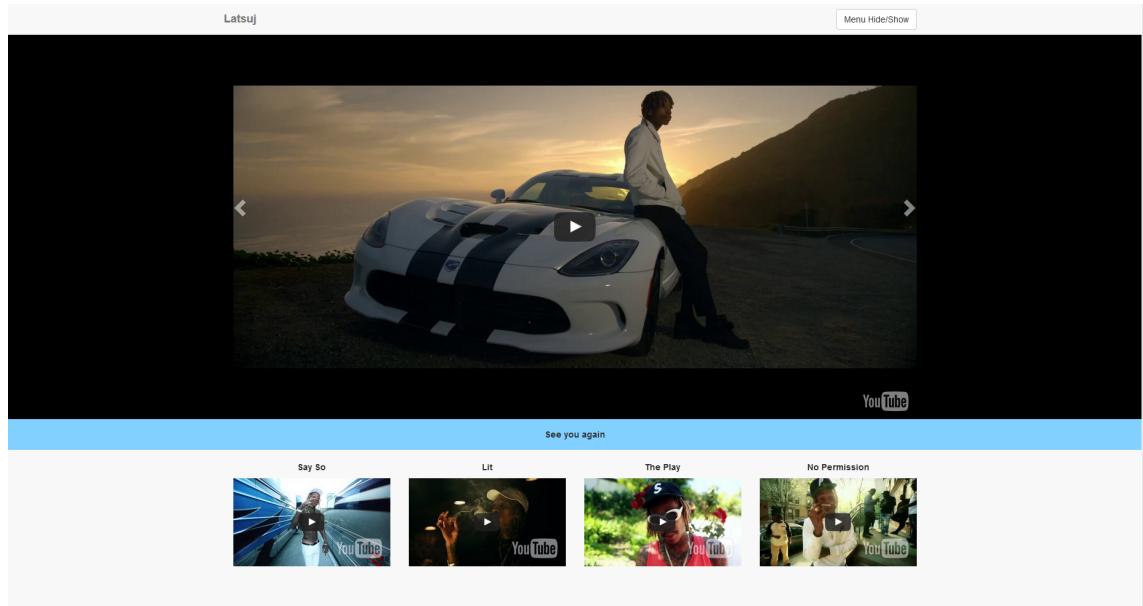


Bootstrap et Polymer

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 Techniques de conceptions adaptatives utilisés	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 L'exemple de A à Z	12
2.1 Installation	12
2.2 Creation des containers	13
2.3 Barre de menu	17
2.4 Le carousel	21
2.5 Le bloc d'informations	24
2.6 Clips vidéos supplémentaires	31
2.7 Vidéo principale	35
2.8 Le menu	36
3 Outils de développement et test	38
3.1 Outils de développement	38
3.2 Outils de test	39
4 Documentations, Outils, liens utiles	40

1 Techniques de conceptions adaptatives utilisés

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.

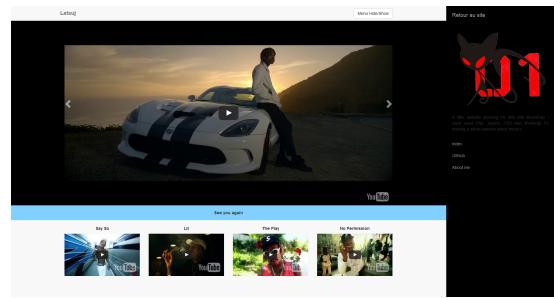


FIGURE 1 – Le site pour PC

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

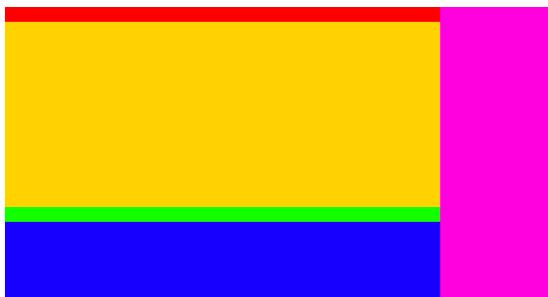
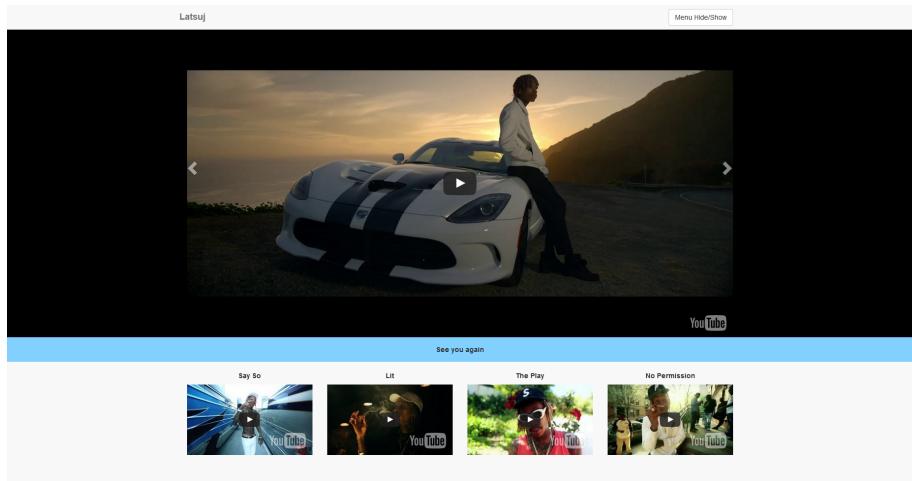


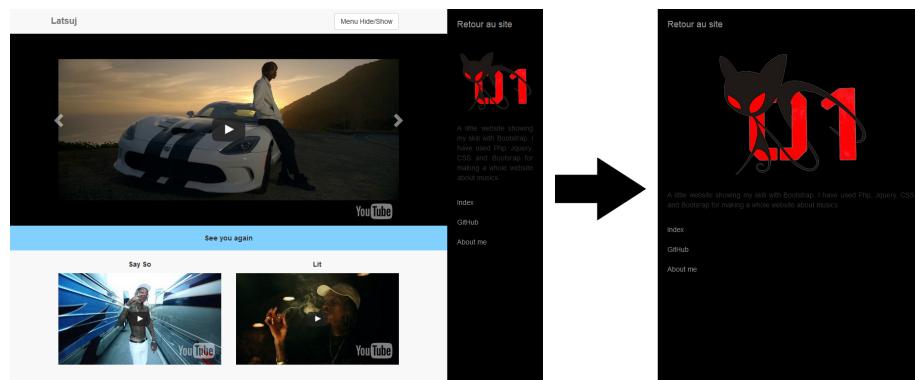
FIGURE 2 – Structure du site

que l'on peut ou non afficher grâce à un bouton qui se trouve 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 :



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 intéresser au menu (zone violette). 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 spécifier de prendre toute la longueur disponible dans le bloc où elle se trouve et de prendre automatiquement une hauteur 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 peut plus simple, une classe existe nommée « `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 chapitre.

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 claire, nous allons montrer leur application sur le site.

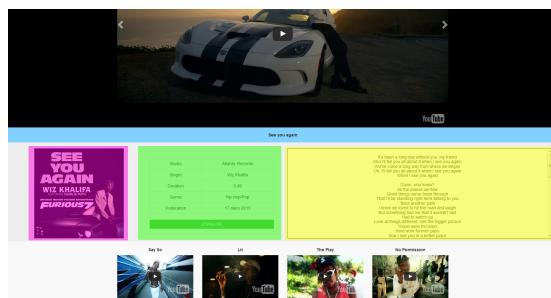
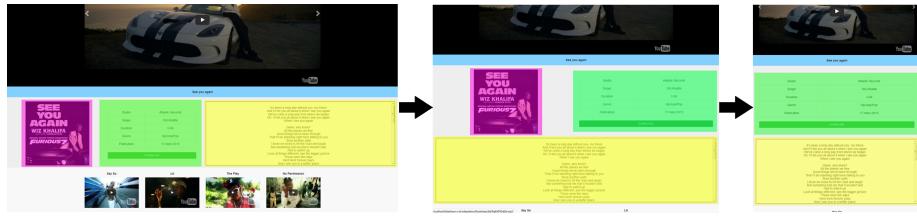


FIGURE 3 – Les blocs

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 quant à lui prend 100% de la largeur 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 disparaîtra.



Sur Bootstrap, il est assez simple de réaliser ceci. Cette technologie inclue un système de grille relativement solide et pratique. Il faut dans un premier temps, définir notre zone où l'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. Je n'entrerai dans les détails qu'au chapitre suivant, pour l'instant il faut imaginer cela comme une subdivision en 12 parties de la zone rouge sur la figure 4. Comme je l'ai déjà expliqué précédemment, l'image dans la zone rouge ne représente que 25% de cette zone lorsque l'écran est supérieur à 1500px (valeur choisie par mes soins). Ensuite, pour spécifier 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>
```

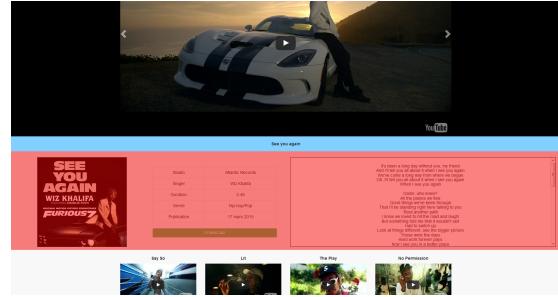


FIGURE 4 – Zone du container

Nous allons maintenant entrer dans le vif du sujet, comme dit plus haut, le container est divisé en 12 parties que j'appellerai dans la suite colonne. On peut fixer le nombre de colonnes que peut prendre un élément. Reprenons et analysons précisément 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. Au niveau du 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 effet. Tandis que le X dans ces classes représente tout simplement le nombre de colonnes que prendra l'élément utilisant ces classes. Je reviendrai 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 expliquer 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 qui 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

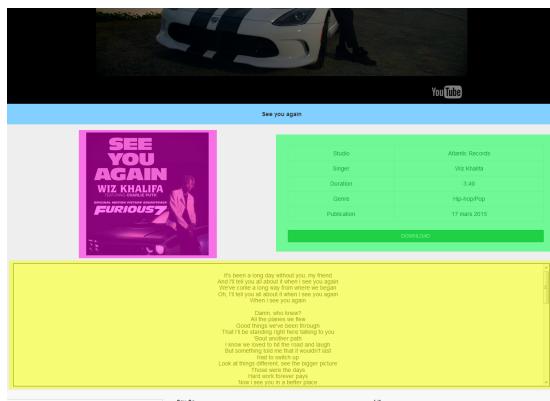


FIGURE 6 – 50% de la fenêtre

Dans notre cas, si la fenêtre est inférieur à 1500px alors les classes lg ne sont plus utilisées. Prenons une fenêtre de 1300px, si l'on regarde le tableau de la figure 5, on remarque que pour toutes les dimensions situées 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.

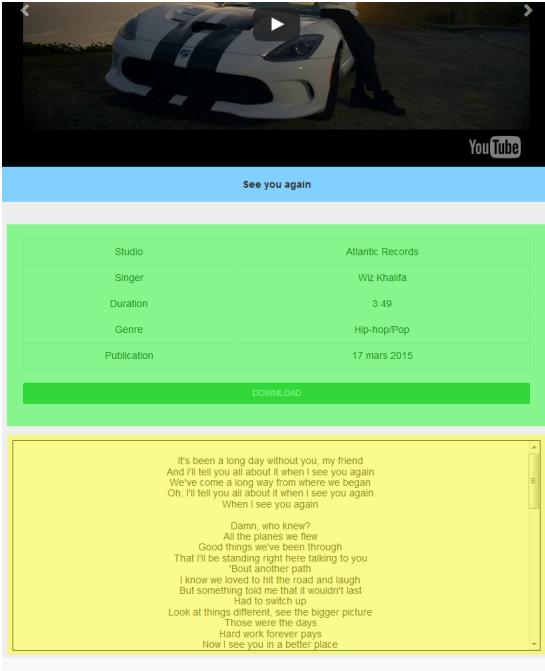


FIGURE 7 – L'image a disparu

Si nous continuons de réduire la fenêtre et atteignons une taille inférieure 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 colonnes pris par cet élément sera réduit à zéro. Cela se traduit par une disparition complète de l'élément comme on peut 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 règles CSS pour réaliser ces modifications. Plus précisément, Bootstrap utilise l'héritage et le cascading de CSS pour pouvoir réécrire des règles sur des règles. Si on ne définit 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'aurai pu omettre le « col-md-6 » dans mon texte bleu et cela donnerai 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 dans le DOM nommé 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, tout 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 largeur 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>
    @media (max-width :900px) {
      :host {
        width :100%;
      }
    }
    </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 ». Cette dernière permet de faire la division et le reposicionnement 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 apparait 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 largeur maximale 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

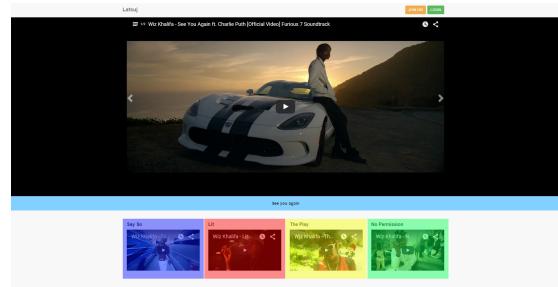
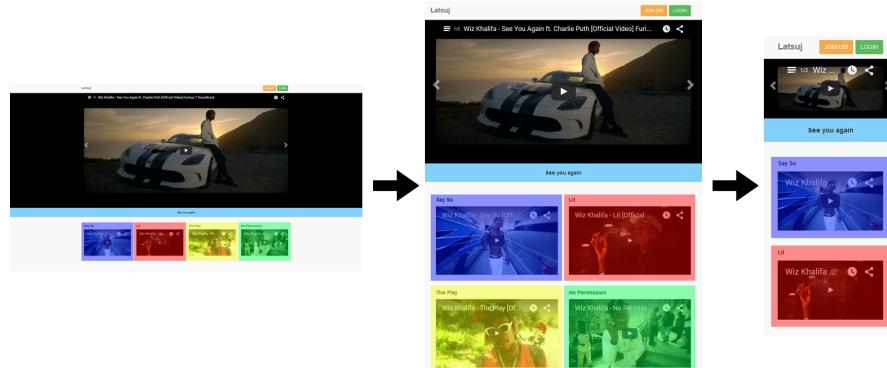


FIGURE 8 – Les 4 blocs

rupture avec les modules précédents :



À travers ces différents exemples, nous avons pu voir comment faire des points de rupture sous les deux technologies différentes. Sur Bootstrap, il est bien plus simple d'utiliser ces points de rupture. L'utilisation des classes xs, sm, md et lg rendent le principe facile à comprendre. Cependant, nous sommes limité par Bootstrap à suivre ces quatres points de rupture. Si le site contient de nombreux autres points de rupture, Bootstrap perd de sa superbe car cela reviendrait à faire du CSS classique. Sur Polymer, il faut bien comprendre les média queries (points de rupture) pour pouvoir les utiliser. Comme Polymer fractionne notre code, on se retrouve avec du CSS et donc des points de rupture 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 positionnement consistent 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).

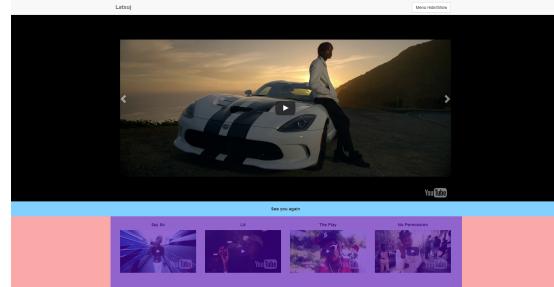


FIGURE 9 – Le container du bas de page

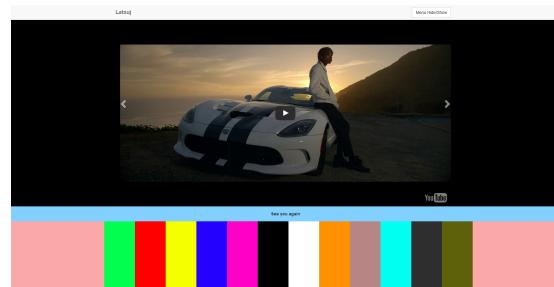


FIGURE 10 – Les 12 divisions d'un container

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 grille 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 exéderont le nombre de colonnes disponibles sur la ligne, ils se placeront automatiquement sur la ligne en dessous.

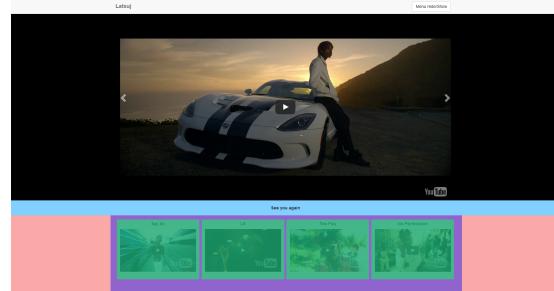
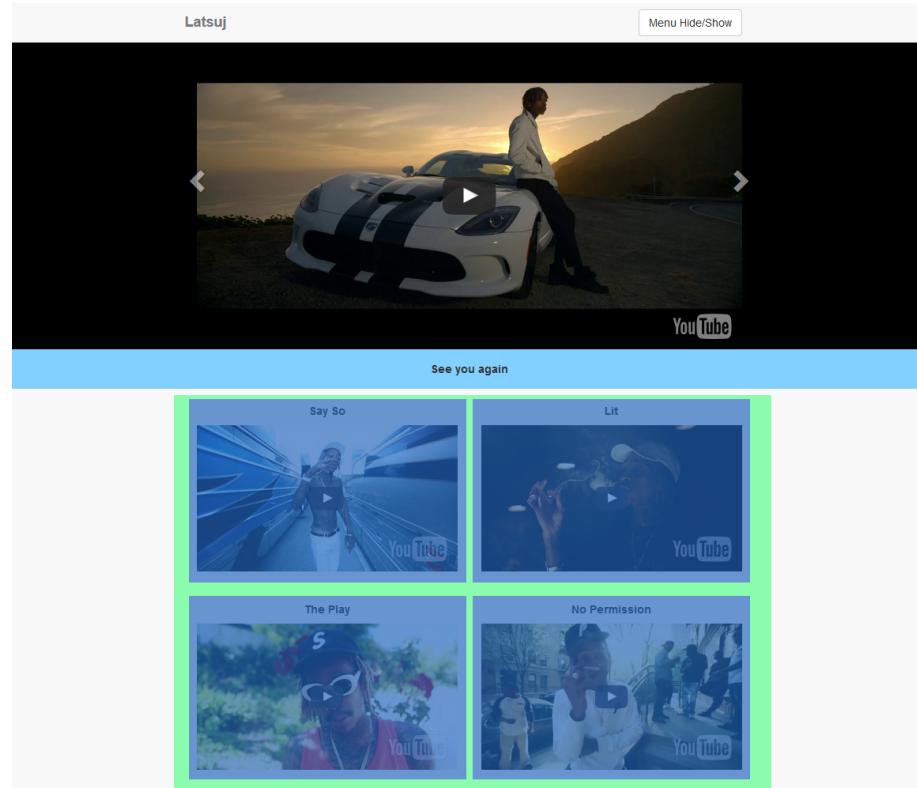


FIGURE 11 – Les 12 divisions d'un container



Comme vu dans les exemples précédent, les clips musicaux du bas de la page étaient affichés 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 colonnes qu'on a permis à l'élément d'utiliser. Ici, j'ai utilisé la classe « col-XX-6 » sur Bootstrap. Pour indiquer que chaque vidéos prendrait 6 colonnes. Comme il y a 4 clips, les deux premières vidéos ont rempli la première ligne puis la troisième vidéo est automatiquement passée à la ligne en dessous pour compléter le container.

Ainsi, nous avons une idée générale des moyens que j'ai utilisé pour faire mon site ainsi qu'une compréhension poussée des techniques que je vais employé pour comprendre comment j'ai formé mon site.

2 L'exemple de A à Z

2.1 Installation

Avant même de commencer à développer, il faut installer les outils nécessaires pour déployer et exécuter les exemples expliqués ci-dessous. Dans un premier temps, il faut savoir que j'ai utilisé un peu de PHP dans mon exemple. Il faut donc commencer par installer un interpréteur. Dans mon cas, j'ai choisi WAMP Serveur. L'installation est relativement aisée et bien expliquée sur le site officiel du logiciel. La seule difficulté possible que vous rencontrerez sera lié au port 80. Certaines applications comme Skype utilisent ce port par défaut, il faut donc le libérer pour que Apache (un logiciel dans wamp) puisse s'en servir.

Une fois le logiciel installé et fonctionnant sans problème (icone verte), il faut remplir un peu la base de donnée avec quelques informations pour que mon exemple ci-dessous puisse fonctionner sans problème. Ci-dessous figure la structure globale de la base de donnée (PhpMyAdmin dans wamp) :



FIGURE 12 – WAMP

Pour plus de facilité lors de la création de la base de donnée, il existe une fonction à exécuter dans le fichier sql.php qui se trouve dans inc qui se nomme : setup(). Cette fonction va automatiquement créer la base de donnée suivant le schéma que j'utiliserai ensuite. Il suffit alors de remplir la base « latsuj ».

La table « movie » permet de donner l'ensemble des informations sur une vidéo. « TITLE » représente le titre de la musique. « CODE » représente le code youtube du clip. « STUDIO » représente le studio de développement de la musique. « SINGER » représente le nom de l'artiste. « DURATION » représente la durée exacte de la musique. « GENRE » représente le genre de la musique. « PUBLICATION » représente la date de la sortie de la musique. « IMG » représente le lien de la jaquette de la musique. Et enfin « LYRIC » représente les paroles de la musique.

La table « pubs » permet de spécifier les vidéos qui se trouvent en bas de page. Chaque colonne associe un lien avec un titre. Ainsi « LINK1 » représente le lien youtube de la vidéo et « TITLE1 » représente le titre de la vidéo. Et chacune des quatres vidéos suivent exactement le même principe. Enfin la dernière table n'est là que pour réaliser une association ONE-ONE d'une ID de la table « movie » vers la table « pubs ». Dans notre cas, il n'y a pas grand intérêt ici car chaque id de "movie" sera attribué à un id de "pub" qui sera au final identique. 1 va pointé vers 1, 2 pointera vers 2...

Si vous souhaitez simplement lancer le site ou que vous disposez des fichiers, il faudra placer l'ensemble des fichiers du site dans le répertoire "www" de wamp puis lancer un navigateur internet sur l'adresse suivante pour Boostrap par exemple : <http://localhost/Bootstrap/index.php>.

Si vous souhaitez simplement suivre le tutoriel, il faudra dans chaque technologie effectuée une installation supplémentaire dans un fichier index.html à placer dans le repertoire "www" de WAMP. Pour Bootstrap, il faudra simplement rajouter les CDN dans le fichier, c'est à dire les fichiers CSS et JavaScript :

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap-theme.min.css">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
```

Pour Polymer, c'est un petit peu plus complexe, il faudra installé Node.js puis suivre à la lettre toutes les commandes indiquées sur la page officielle de Polymer pour créer un projet Polymer.

2.2 Creation des containers

Il est maintenant temps de commencer l'application de ce que nous avons vu précédent dans un exemple concret. Nous allons reproduire l'ensemble de mon site sous les deux technologies que sont Polymer et Bootstrap. Si vous avez compris ce que j'ai expliqué dans le chapitre précédent, il faut commencer par créer nos containers. C'est à dire nos éléments qui serviront de base à nos grilles de positionnement.

Sur mon exemple, il y a deux grands blocs principaux. Un container ou plutôt une barre principale qui se trouve en haut du site (zone rouge sur l'image à droite) et une partie contenue qui est en fait un grand carrousel (couleur violette sur l'image à droite). La partie violette sera une partie du site qui sera animée via un script JQuery mais suivra globalement la même structure.

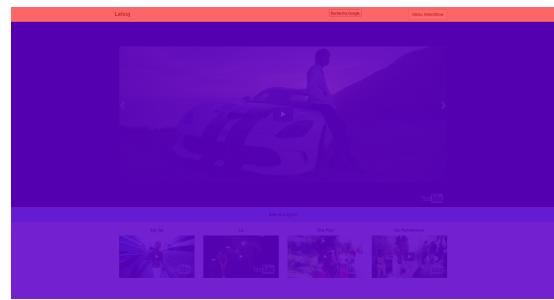


FIGURE 13 – Les 12 divisions d'un container

Sous Bootstrap, j'ai utilisé trois classes sur des balises HTML pour réaliser cela facilement : « container-fluid », « navbar » et « container ». La première permet de spécifier que nous utiliserons 100% de la largeur de la fenêtre. La seconde est la balise pour spécifier où se trouve notre menu de navigation. La dernière permet de prendre la dimension en pixel avant le point de rupture. Les autres classes qui se trouvent dans l'exemple ci-dessous ne sont que des classes améliorant le design et ne sont donc pas nécessaire pour reproduire mon exemple. Ainsi je n'expliquerai pas leurs utilité, la plupart étant expliquée dans la documentation de Bootstrap.

```
...
<body>
  <nav class="navbar navbar-default navbar-static-top">
    <div class="container-fluid black">
      </div>
  </nav>
  <div class="container-fluid black">
    </div>
</body>
...
```

Sous Polymer, nous allons simplement reproduire le comportement des classes précédentes et en faire de nouveaux éléments du DOM (Document Object Model). Pour faire cela proprement, j'ai utilisé Firebugs et analysé le CSS utilisé sur Bootstrap pour reproduire le style de ces classes. Le code sous Polymer dans la page index est le suivant :

```
...
<body>
  <bar-top>
    <my-container>
      </my-container>
  </bar-top>
  <my-container-full>
    </my-container-full>
</body>
...
```

Pour que ce code fonctionne correctement, il faut bien évidemment créer les modules. Un module est simplement une page HTML avec une certaine syntaxe que nous incorporons dans la page principale (index.html) avec le code suivant à placer entre les balises HEAD :

```
<link rel="import" href="bar-top.html">
<link rel="import" href="my-container.html">
<link rel="import" href="my-container-full.html">
```

Le fichier bar-top.html qui représentera notre nouvelle élément du DOM aura son style CSS fixé à lui. À noter que la syntaxe est extrêmement importante. Un module doit obligatoirement être nommé de la façon suivante (xxxx-xxxx). Le code ci-dessous est à ajouter au même niveau que index.html dans le fichier **bar-top.html** :

```

<dom-module id="bar-top">
  <template>
    <style>
      :host {
        background-color : #f8f8f8 ;
        border-color : #e7e7e7 ;
        z-index : 1000 ;
        border-width : 0 0 1px ;
        position : relative ;
        min-height : 50px ;
        display : block ;
        box-sizing : border-box ;
        border : none ;
      }
      @media (min-width : 768px) {
        :host {
          border-radius : 0 ;
        }
      }
    </style>
    <content></content>
  </template>
  <script>
    Polymer({
      is : "bar-top"
    });
  </script>
</dom-module>

```

Comme dit plus haut, il s'agit du module « bar-top », le lien avec ce code est réalisé avec l'id du dom-module ainsi que le script qui spécifie son nom. « :host » représente le module lui-même. Toutes les règles CSS inscrites dans ce tag prendra automatiquement effet pour tout élément <bar-top></bar-top>. Enfin la balise « content » est le point d'ancrage du nouvelle élément. C'est à dire que le code se trouvant entre les balises <bar-top> et </bar-top> se positionnera à l'intérieur des balises « content ». Pour les média queries, j'ai déjà expliqué leurs fonctionnements dans le chapitre précédent.

Ainsi la création de module prendra toujours la même forme, il faut reproduire l'opération pour que les deux autres modules prennent eux aussi effet :

```

<dom-module id="my-container">
  <template>
    <style>
      :host {
        padding-right : 15px ;
        padding-left : 15px ;
        margin-right : auto ;
        margin-left : auto ;
        box-sizing : border-box ;
        display : block ;
        height : auto ;
      }
      @media (min-width :900px) {
        :host {
          width :700px ;
        }
      }
      @media (min-width :1200px) {
        :host {
          width :970px ;
        }
      }
      @media (min-width :1500px) {
        :host {
          width :1170px ;
        }
      }
    </style>
    <content></content>
  </template>
  <script>
    Polymer({
      is : "my-container"
    });
  </script>
</dom-module>

```

```

<dom-module id="my-container-full">
  <template>
    <style>
      :host {
        display : block ;
        position : absolute ;
        width : 100% ;
        height : auto ;
      }
    </style>
    <content></content>
  </template>
  <script>
    Polymer({
      is : "my-container-full"
    });
  </script>
</dom-module>

```

Ainsi nous obtenons le même résultat avec les deux technologies. Nos containers étant maintenant prêt, nous allons nous occuper de la barre du haut (zone rouge dans la figure 12).

2.3 Barre de menu



Le menu est composé de ma signature « Latsuj », tout simplement mon nom à l'envers ainsi qu'un bouton pour afficher ou cacher le menu. J'ai choisi de mettre ma signature à gauche à cause de la nature de l'utilisateur. Un humain lit toujours de mani'ere inconsciente en diagonale. C'est aussi ce que l'on nomme lecture-rapide. Le premier mot que l'utilisateur verra sera donc ma signature. Ensuite, j'ai mis le bouton à droite pour faire écho au texte à gauche, la nature préfère ce qui est symétrique, ainsi que pour utiliser toutes la place disponible. Sous Bootstrap, comme toujours nous avons des classes pour réaliser cela sans trop se fatiguer. Par rapport à ce dont j'ai parlé précédemment, il y aura 3 nouvelles classes ici que je n'ai pas encore évoqué : « pull-left », « pull-right », « btn ». Les deux classes « pull » permettent de pousser leurs contenus vers la gauche ou la droite. « Btn » permet quant à elle de mettre le style d'un bouton sur le composant où on l'utilise. On obtient alors notre menu assez facilement :

```
<nav class="navbar navbar-default navbar-static-top">
  <div class="container">
    <div class="row">
      <div class="col-sm-12 col-md-12 col-lg-12">
        <div class="pull-left">
          <strong class="navbar-brand">Latsuj</strong>
        </div>
        <div class="pull-right navbar-brand">
          <div class="menu-toggle btn btn-default">Menu Hide/Show</div>
        </div>
      </div>
    </div>
  </div>
</nav>
```

A noter ici que nous aurions pu spécifier que notre signature prenne 2 colonnes, que le bouton prenne aussi 2 colonnes et mettre un offset entre les deux de 8 colonnes. Cependant, j'ai préféré cette méthode car cela me fait moins de ligne de code donc un code plus clair. Les classes dont je n'explique pas le rôle ne sont là que pour rendre les choses plus jolie et ne sont pas nécessaire pour reproduire l'architecture de mon site/exemple. Avec le simple exemple ci-dessus, nous obtenons sous Bootstrap notre menu. Sous Polymer, il va falloir comme précédemment reproduire le comportement des blocs de Bootstrap en créant de nouvelles nodes. Dans le code qui va suivre, il n'y a rien de nouveau, ni de compliqué, ce n'est que du CSS :

```

<dom-module id="my-row">
  <template>
    <style>
      :host {
        display : block ;
        position : relative ;
        margin-right : -15px ;
        margin-left : -15px ;
        box-sizing : border-box ;
      }
    </style>
    <content></content>
  </template>
  <script>
    Polymer({
      is : "my-row"
    });
  </script>
</dom-module>

```

Il n'y a rien de particulier sur ce module « my-row » qui reproduit le comportement de la classe « row » de Boostrap. Notons cependant que dans Polymer, les nouveaux modules sont toujours spécifié comme « display :inline », il faut donc très souvent les passer en « display :bloc ». Notons par ailleurs l'utilisation de box-sizing qui n'est peut-être pas nécessaire ici. Ayant voulu créer un site fonctionnant sur tous les navigateurs, cet attribut est essentiel. Cependant, Polymer étant très mal supporté pour l'instant, cet attribut perd de son intérêt au jour d'aujourd'hui. Si les navigateurs s'efforçaient de faire le nécessaire pour supporter Polymer, il n'y aurait pas besoin de modifier mon exemple car j'aurais déjà prévu cela.

```

<dom-module id="col-top">
  <template>
    <style>
      :host {
        display : block ;
        position : relative ;
        padding-right : 15px ;
        padding-left : 15px ;
        box-sizing : border-box ;
        min-height : 50px ;
        float : left ;
        width : 100% ;
      }
    </style>
    <content></content>
  </template>
  <script>
    Polymer({
      is : "col-top"
    });
  </script>
</dom-module>

```

Ce module « col-top » reprend la définition de la classe « col » de Bootstrap. Il n'y a rien de spécial dans ce module, l'ensemble de ces attributs on déjà été commenté.

<pre> <dom-module id="pull-left"> <template> <style> :host { display : block ; float : left !important ; box-sizing : border-box ; margin-left : -15px ; min-height : 50px ; padding : 15px 15px ; font-size : 18px ; line-height : 40px ; margin-top : -15px ; } </style> <content></content> </template> <script> Polymer({ is : "pull-left" }); </script> </dom-module></pre>	<pre> <dom-module id="pull-right"> <template> <style> :host { display : block ; float : right !important ; box-sizing : border-box ; margin-left : -15px ; min-height : 50px ; padding : 15px 15px ; font-size : 18px ; line-height : 40px ; margin-top : -15px ; } </style> <content></content> </template> <script> Polymer({ is : "pull-right" }); </script> </dom-module></pre>
--	---

Ces deux modules servent à aligner leurs contenus à droite ou à gauche dans leurs containers. Pour des raisons graphiques, j'ai ajouté un attribut « margin-top ». La seule différence entre ces deux blocs réside dans leur float. On aurait pu créer un seul élément et spécifier ensuite dans le fichier index.html un style particulier vers la gauche ou la droite. Cependant, j'ai trouvé plus intéressant de séparer tout l'aspect graphique de l'index et des modules. Polymer proscrit l'emploi de feuille de style séparé pour la page HTML qui sera affiché. Les règles CSS se trouve soit dans le module ou alors dans une balise style défini « is="custom-style" ».

```

<dom-module id="title-top">
  <template>
    <style>
      :host {
        margin-left :-15px ;
        line-height : 40px ;
        float : left ;
        height : 50px ;
        padding : 15px 15px ;
        font-size : 18px ;
        margin-top : -12px ;
        color :#777 ;
      }
      b,strong {
        font-weight :700 ;
      }
    </style>
    <strong>Latsuj</strong>
  </template>
  <script>
    Polymer({
      is : "title-top"
    });
  </script>
</dom-module>

```

Le module « title-top » est le module qui permet d'afficher le titre. J'ai préféré là aussi en faire un module car cela permet de bien découper mon code. Je sais où se trouve toutes les règles CSS en rapport avec ce dernier.

```

<dom-module id="my-button">
  <template>
    <style>
      :host {
        color : #333 ;
        background-color : #FFF ;
        display : inline-block ;
        margin-bottom : 0px ;
        font-weight : normal ;
        text-align : center ;
        vertical-align : middle ;
        cursor : pointer ;
        background-image : none ;
        border : 1px solid #CCC ;
        white-space : nowrap ;
        padding : 6px 12px ;
        font-size : 14px ;
        line-height : 1.42857 ;
        border-radius : 4px ;
      }
    </style>
  </template>
  <script>
    Polymer({
      is : "my-button"
    });
  </script>
</dom-module>

```

```

        -moz-user-select : none ;
        box-sizing : border-box ;
    }
    :host( :focus){
        color : #333 ;
        background-color : #E6E6E6 ;
        border-color : #8C8C8C ;
    }
    :host( :active) {
        color : #333 ;
        background-color : #D4D4D4 ;
        border-color : #8C8C8C ;
        outline : 0px none ;
        background-image : none ;
        box-shadow : 0px 3px 5px rgba(0, 0, 0, 0.125) inset ;
    }
</style>
<div>Menu Hide/Show</div>
</template>
<script>
Polymer({
    is : "my-button"
});
</script>
</dom-module>

```

Ce gros module est celui qui va représenter notre bouton en haut à droite. Je vais m'attarder sur une petite particularité. Dans le code, on retrouve deux blocs de règles qui sont lié à des évènements. Le premier « :focus » permet de définir des règles CSS lorsque le bouton aura le focus dans la fenêtre. Le deuxième est un évènement lorsque le bouton a été cliqué. On aurait pu aussi définir ces changements de règles avec JavaScript mais j'ai choisi d'accomplir cela comme il est indiqué dans la documentation de Polymer.

Nos modules étant créé, nous pouvons maintenant nous en servir pour reproduire le menu comme sur Bootstrap. Le code obtenu après tous ces efforts est simple et clair :

```

<bar-top>
    <my-container>
        <my-row>
            <col-top>
                <pull-left>
                    <title-top></title-top>
                </pull-left>
                <pull-right>
                    <my-button></my-button>
                </pull-right>
            </col-top>
        </my-row>
    </my-container>
</bar-top>

```

2.4 Le carousel

Le carousel est la partie qui m'a demandé le plus de temps sous Polymer car rien n'a été mis en place dans ce framework (structure logicielle). D'un autre coté, Bootstrap a mis en place plusieurs

classes pour avoir du contenu animé facile à implémenté. Le carousel est une de ces animations. Sous Bootstrap, 3 classes sont nécessaire pour faire un carousel digne de ce nom : « carousel », « carousel-inner » et « item ». La première classe permet d'initialiser le carousel et permet si il y a plusieurs carousels de faire la distinction entre ces derniers en spécifiant un ID. La deuxième classe définit la zone pour la page actuelle du carousel. Enfin, la dernière classe permet de définir une slide du carousel. Pourquoi avoir choisi le carousel ? Tout simplement car il s'agit d'une méthode d'animation simple et facile à comprendre. Elle permet d'avoir tout le contenu du site sur une page. Cependant, cela peut amener un problème qui d'ailleurs se trouve dans mon exemple. Lors du chargement de la première page, on remarque que le site est relativement long à se charger. C'est tout simplement car mon site charge en fait l'ensemble des slides au chargement de la première slide. On pourrait faire comme sur facebook et ne charger qu'une partie du contenu. Cependant, cela veut aussi dire que l'utilisateur devra charger toutes les X slides, les prochaines slides. Il y a donc un compromis à prendre. J'ai préféré sur mon exemple, au vu du nombre faible de slide sur le site, faire que l'utilisateur charge l'ensemble des slides. Ainsi il ne souffre d'aucun ralentissement pendant l'utilisation.

```
<div class="container-fluid black">
    <div id="myCarousel" class="carousel slide" data-ride="carousel" data-interval="false">
        <div class="carousel-inner" role="listbox">
            <div class="item active" >
                ...
            </div>
            <div class="item" >
                ...
            </div>
        </div>
    </div>
</div>
```

L'attribut « data-ride » permet de marquer le carousel comme animé au chargement de la page. L'attribut « data-interval » quant à lui renseigne sur le temps avant qu'une slide change à la suivante automatiquement. Enfin la classe « active » indique simplement que le contenu de la div sera la première slide du carousel.

Sous Polymer, c'est bien plus compliqué pour fabriquer notre carousel. Pour arriver à faire cela, je me suis tout simplement basé sur du JavaScript. Le fonctionnement est assez simple, toutes les slides sont positionné à gauche de la fenêtre sauf une qui sera affiché sur l'écran de l'utilisateur. Lorsque l'on clique sur une flèche vers la gauche ou la droite, la slide suivante se placera respectivement à droite ou à gauche de la slide courante. Enfin, on déplace vers la gauche ou la droite, la slide courante et la slide suivante pour remplacé la slide actuelle par la suivante. Il n'y avait aucun moyen fournis dans Polymer pour effectuer cela autrement qu'en utilisant JavaScript.

Dans un premier temps, il faut créer les nouvelles nodes qui nous permettrons de faire le changement de slide. C'est à dire les flèches sur la gauche et la droite de la vidéo principale.

<pre> <dom-module id="my-right"> <template> <style> img { position :absolute ; right :1% ; top :0 ; bottom :0 ; margin :auto ; z-index :1000 ; opacity : 0.4 ; transform :scale(1.1,1.1) ; cursor :pointer ; } img :hover { opacity : 1 ; } </style> </template> <script> Polymer({ is : "my-right" }); </script> </dom-module> </pre>	<pre> <dom-module id="my-left"> <template> <style> img { position :absolute ; left :1% ; top :0 ; bottom :0 ; margin :auto ; z-index :1000 ; opacity : 0.4 ; transform :scale(1.1,1.1) ; cursor :pointer ; } img :hover { opacity : 1 ; } </style> </template> <script> Polymer({ is : "my-left" }); </script> </dom-module> </pre>
---	--

On mettra ces éléments dans chacune des slides. Le code global de la page devient alors celui là :

```

...
<body>
  <bar-top>
    <my-container>
      </my-container>
  </bar-top>
  <my-container-full id="XX">
    <my-container>
      <my-row>
        <col-mid>
          <my-left></my-left>
          <my-right></my-right>
        </col-mid>
      </my-row>
    </my-container>
  </my-container-full>
</body>
...

```

Le « container-full » a comme on peut le voir un ID. Ce module représente une slide. Avec un peu de code PHP (car j'aime pas la duplication de code), je reproduis ce morceau de code plusieurs fois en incrémentant l'ID de 1. Cela permet d'identifier les slides facilement dans le script JavaScript qui est dessous :

```

var slide=1;
var SLIDE_MAX=4;
var SLIDE_MIN=1;
$("#my-left").click(function() {
    $("#"+slide).css("right","0px");
    $("#"+slide).animate({right : -$ (window).width()+"px"},500);
    slide++;
    nextslide();
    $("#"+slide).css("left","auto");
    $("#"+slide).css("right",$ (window).width()+"px");
    $("#"+slide).animate({right : $ ("my-menu").width()+"px"},350,function() {
        $("#"+slide).css("right","auto");
    });
});
$("#my-right").click(function() {
    $("#"+slide).css("left","0px");
    $("#"+slide).animate({left : -$ (window).width()+"px"},500);
    slide--;
    nextslide();
    $("#"+slide).css("right","auto");
    $("#"+slide).css("left",$ (window).width()+"px");
    $("#"+slide).animate({left : $ ("my-menu").width()+"px"},350,function() {
        $("#"+slide).css("left","auto");
    });
});
function nextslide() {
    if(slide<1) {
        slide=SLIDE_MAX;
    }
    if(slide>4) {
        slide=SLIDE_MIN;
    }
}

```

Comme dit plus haut, sous Polymer, il n'y a aucune autre manière d'effectuer cela. Certains développeurs du projet Polymer sont actuellement entrain d'implémenter un module pour pouvoir en quelques minutes avoir un carousel fonctionnel. Pour l'instant, le carousel ne fonctionne qu'avec des images, ce qui n'était pas suffisant pour effectuer ce que je voulais faire.

2.5 Le bloc d'informations

Le bloc d'informations est la petite partie caché du site qui regroupe toutes les informations sur un morceau de musique. J'ai volontairement caché l'intérieur de cette partie car je souhaitais un site rapide à parcourir. L'oeil humain fonctionne par étape : progression, retour à la ligne et régression. Mon but était que l'utilisateur effectue en environs 1 seconde, la lecture entière d'une page du site. Il fallait donc que je limite le nombre de points de fixation (un point d'arrêt dans la lecture). L'oeil s'arrêtera entre 10 à 30 ms sur ces points. Mon site comporte 4 points d'arrêts comme on peut le voir sur la figure 13. Si j'avais laissé mon bloc d'informations visible, il y aurait 7 points d'arrêts.

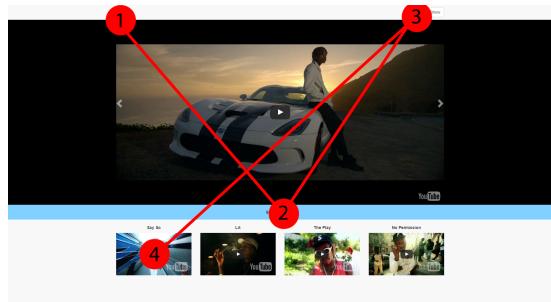


FIGURE 14 – Lecture du site

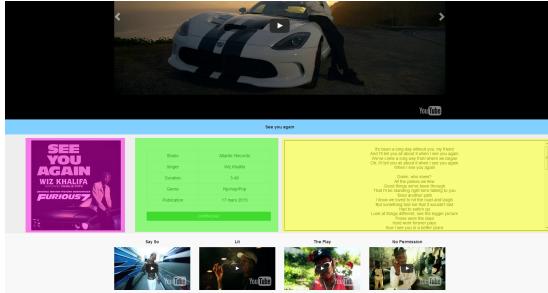


FIGURE 15 – Le bloc d’informations

tains points de rupture (faible résolutions). J’ai effectué ce choix car je pense que l’utilisateur sur son téléphone ne dispose pas forcément d’une connexion phénoménale comme on pourrait en avoir sur un ordinateur connecté en Ethernet. L’image n’étant pas une donnée essentielle et prenant une part importante de ressources, j’ai jugé bon de la supprimé à basse résolution pour le confort de l’utilisateur.

Sous Bootstrap, il n’y a rien de nouveau sur ce que j’ai cité plus haut. Il s’agit maintenant seulement de placement des éléments dans une grille de positionnement. Le code ci-dessous réalise ce que j’ai fait. Si il y a un point incompris, vous avez donc raté un point dans mes explications précédentes :

```
<div class="title text-center">
    <strong>< ?php echo $movie[0]['TITLE'] ;?></strong>
</div>
```

Ce petit bout de code permet d’afficher la barre de bleu avec le titre de la musique en son centre. Notons le petit emploi de PHP pour ne pas faire de travail trop répétitive entre mes pages.

```
<div class="hidden-xs hidden-sm col-md-6 col-lg-6 text-center pagination-centered">
    <img src=< ?php echo $movie[0]['IMG'] ;?> class="img-responsive" alt="Responsive image">
</div>
```

Ce code est celui qui servira pour afficher l’image. Comme on peut le voir l’image est "responsive" et sera cachée lorsque la largeur de la fenêtre sera inférieur à « sm ».

```

<div class="col-md-6 col-lg-6 text-center pagination-centered center-block" style="display : flex ;">
    <div style="align-self : center ;width :100% ;">
        <table class="table table-bordered">
            <tbody>
                <tr>
                    <td>Studio</td>
                    <td>< ?php echo $movie[0]['STUDIO'] ;?></td>
                </tr>
                <tr>
                    <td>Singer</td>
                    <td>< ?php echo $movie[0]['SINGER'] ;?></td>
                </tr>
                <tr>
                    <td>Duration</td>
                    <td>< ?php echo $movie[0]['DURATION'] ;?></td>
                </tr>
                <tr>
                    <td>Genre</td>
                    <td>< ?php echo $movie[0]['GENRE'] ;?></td>
                </tr>
                <tr>
                    <td>Publication</td>
                    <td>< ?php echo $movie[0]['PUBLICATION'] ;?></td>
                </tr>
            </tbody>
        </table>
        <a href="ddl/< ?php echo $movie[0]['CODE'] ;?>.mp3" class="btn btn-success btn-sm full"
download>DOWNLOAD</a>
    </div>
</div>

```

Le tableau est évidemment lui aussi composé d'un peu de PHP. Il n'y a rien d'exceptionnel, il s'agit seulement d'une balise TABLE utilisant le style de Bootstrap avec la classe « table ». A noter ici, la petite astuce qui permet au tableau de prendre 100% de l'espace lorsque l'image disparaît. Je n'ai pas spécifié volontairement le fait que le tableau prenne 12 colonnes, je n'ai rien mis pour les petites résolutions car par défaut Bootstrap force les éléments à prendre le plus d'espace possible. Je n'ai donc pas eu à le rajouter.

```

<div class="col-md-12 col-lg-6">
    <div class="text-center write">
        <h2>< ?php echo $movie[0]['LYRIC'] ;?></h2>
    </div>
</div>

```

Ce bloc fait référence au parol de la chanson. J'ai utilisé la balise H2 pour faire du « responsive 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 incommodé à basse résolution. Pour résoudre ce problème, une proportion a été spécifié 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.

```

It's been a long day without you, my friend
And I'll tell you all about it when I see you again
We've come a long way from where we began
Oh, I'll tell you all about it when I see you again
When I see you again

Damn, who knew?
All the planes we flew
Good things we've been through
That I'll be standing right here talking to you
'Bout another path
I know we loved to hit the road and laugh
But something told me that it wouldn't last
Had to switch up
Look at things different, see the bigger picture
Those were the days
Hard work forever pays
Now I see you in a better place

How can we not talk about family when family's all that we got?
Everything I went through you were standing there by my side

```

```

It's been a long day without you, my friend
And I'll tell you all about it when I see you again
We've come a long way from where we began
Oh, I'll tell you all about it when I see you again
When I see you again

Damn, who knew?
All the planes we flew
Good things we've been through
That I'll be standing right here talking to you
'Bout another path
I know we loved to hit the road and laugh
But something told me that it wouldn't last
Had to switch up
Look at things different, see the bigger picture
Those were the days

```

Avec l'ensemble de ces parties de code, on arrive à reproduire notre bloc d'informations sous Boots-trap, il s'agissait bien comme je le disais en début de partie d'une affaire de positionnement. Sous Polymer, le code est moins long et moins complexe à mes yeux. Cependant, il faut là encore créer un gros nombre de nouveaux modules, ce qui représente un travail fastidieux et long. Le code dans l'index se résume à cela :

```

<bloc-information>
  <my-col-50-100>
    <my-col-50-50>
      <my-col-hidden-1200>
        <my-img-responsive link=< ?php echo $movie[0]['IMG'] ; ?>></my-img-responsive>
      </my-col-hidden-1200>
    </my-col-50-50>
    <my-col-50-50>
      <my-informations studio=< ?php echo $movie[0]['STUDIO'] ; ?>" singer=< ?php
echo $movie[0]['SINGER'] ; ?>" duration=< ?php echo $movie[0]['DURATION'] ; ?>" genre=< ?php echo $movie[0]['GENRE'] ; ?>" publication=< ?php echo $movie[0]['PUBLICATION'] ; ?>" download="ddl/< ?php echo $movie[0]['CODE'] ; ?>.mp3"></my-informations>
    </my-col-50-50>
  </my-col-50-100>
  <my-col-50-100>
    <my-lyric>
      < ?php echo $movie[0]['LYRIC'] ; ?>
    </my-lyric>
  </my-col-50-100>
</bloc-information>

```

Je vais évidemment expliquer l'ensemble de mes modules ci-dessous. Je tiens avant d'indiquer que l'utilisation de PHP est proscrite dans les modules que l'on créé, il faut donc faire en sorte que le module puisse avoir des attributs. Ce qui explique pourquoi ma balise MY-INFORMATION prend beaucoup d'informations.

```

<dom-module id="my-information">
  <template>
    <style>
      ... (se reporter à mon GIT ou voir sur Bootstrap pour avoir le style exact)
    </style>
    <div style="display : flex ;">
      <div style="align-self : center ;width :100% ;">
        <table class="table table-bordered">
          <tbody>
            <tr>
              <td>Studio</td>
              <td>{{studio}}</td>
            </tr>
            <tr>
              <td>Singer</td>
              <td>{{singer}}</td>
            </tr>
            <tr>
              <td>Duration</td>
              <td>{{duration}}</td>
            </tr>
            <tr>
              <td>Genre</td>
              <td>{{genre}}</td>
            </tr>
            <tr>
              <td>Publication</td>
              <td>{{publication}}</td>
            </tr>
          </tbody>
        </table>
        <a href="{{download}}" class="btn btn-success btn-sm full" download>DOWNLOAD</a>
      </div>
    </div>
  </template>
  <script>
    Polymer({
      is : "my-information",
      properties : {
        link : String,
        studio : String,
        singer : String,
        duration : String,
        genre : String,
        publication : String,
        download : String
      }
    });
  </script>
</dom-module>

```

Ce module représente le tableau avec toutes ces informations. Ici, j'ai utilisé comme dit plus haut des propriétés pour cet élément. En rouge, on trouve les propriétés du bloc, j'ai défini ces dernières comme des strings. Pour ensuite, les utiliser dans le bloc, il suffit d'écrire {{ NOM DE LA PROPRIETE }}.

```

<dom-module id="my-col-50-50">
  <template>
    <style>
      :host {
        position : relative;
        min-height : 1px;
        padding-left : 15px;
        padding-right : 15px;
        float :left ;
        width :50% ;
        box-sizing : border-box ;
        vertical-align :middle ;
      }
      @media (max-width :1500px) {
        :host {
          margin-bottom : 20px;
        }
      }
      @media (max-width :1200px) {
        :host {
          width :100% ;
          margin-bottom : 0px;
        }
      }
    </style>
    <content></content>
  </template>
  <script>
    Polymer({
      is : "my-col-50-50"
    });
  </script>
</dom-module>

```

```

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

```

Comme vu précédemment, ce sont ces modules qui joueront le rôle de positionnement sous Polymer. L'utilisation des média queries permet de fixer la taille de la zone dont dispose l'élément en fonction de la largeur de l'écran.

```

<dom-module id="my-img-responsive">
  <template>
    <style>
      :host {
        width :100% ;
        height : auto ;
      }
    </style>
    </img>
  </template>
  <script>
    Polymer({
      is : "my-img-responsive",
      properties : link : String
    });
  </script>
</dom-module>

```

Ce module permet de faire des images responsives. Comme dit au tout départ de mon document. Une image responsive n'est qu'une image avec un largeur de 100% et une hauteur fonction de son ratio. Dans ce module, j'ai aussi ajouté une propriété qui est bien évidemment l'adresse de l'image en question.

```
<dom-module id="my-lyric">
  <template>
    <style>
      :host {
        display : block ;
        overflow-y : scroll ;
        max-height : 300px ;
        background : #F8F8F8 none repeat scroll 0% 0% ;
        border : 1px solid #000 ;
        text-align : center ;
        box-sizing : border-box ;
        padding-top : 20px ;
      }
      h2 {
        font-family : inherit ;
        font-weight : 500 ;
        line-height : 1.1 ;
        color : inherit ;
        font-size : 14px ;
      }
      @media(max-width :600px){
        h2{font-size : 10px ;}
      }
      @media(min-width :601px){
        h2{font-size : 14px ;}
      }
      @media(min-width :1200px){
        h2{font-size : 14px ;}
      }
      @media(min-width :1500px){
        h2{font-size : 14px ;}
      }
    </style>
    <div><h2><content></content></h2></div>
  </template>
  <script>
    Polymer({
      is : "my-lyric",
      properties : {
        link : String
      }
    });
  </script>
</dom-module>
```

Le module « my-lyric » représente le petit bloc avec les paroles de la chanson. Comme dit précédemment, je réalise ici un changement de police suivant la largeur de la fenêtre. J'ai déjà expliqué pourquoi j'ai fait ce choix mais je n'avais pas encore montré la manière de procédé. Il s'agit comme on peut le voir d'un simple changement de la taille de la police.

```

<dom-module id="my-col-hidden-1200">
  <template>
    <style>
      @media (max-width :1200px) {
        :host {
          display : none;
        }
      }
    </style>
    <content></content>
  </template>
  <script>
    Polymer({
      is : "my-col-hidden-1200"
    });
  </script>
</dom-module>

```

Ce dernier module est ici pour reproduire le comportement de la classe « col-hidden-sm ». C'est à dire que ce module permet de supprimer un élément à partir d'une certaine largeur de fenêtre (1200px dans ce cas).

Avec toutes ces informations, nous sommes maintenant capable de reproduire ce que j'ai réalisé. Il ne reste plus qu'à reproduire le comportement pour cacher ou afficher ces informations. Ce mécanisme est produit via JQuery avec les méthodes « slide-toggle » et « clic » :

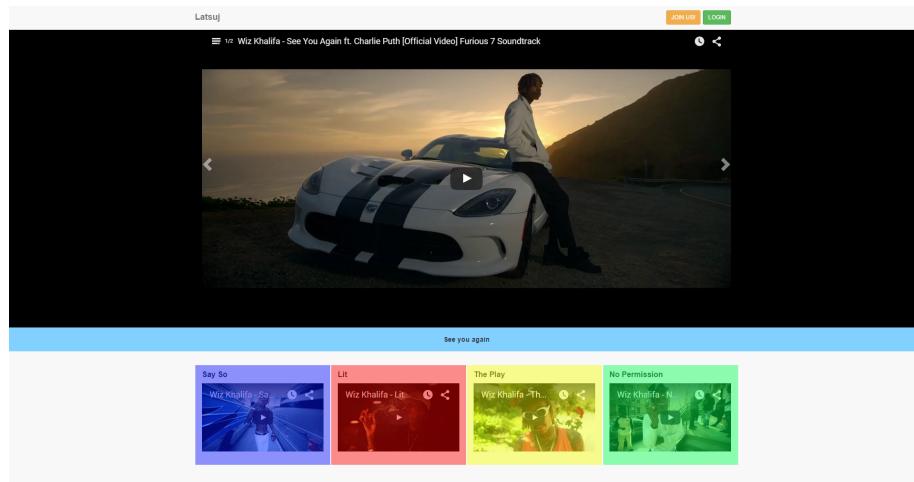
```

$( "#my-bar-title" ).click(function() {
  $( "#bloc-information" ).slideToggle( "fast" );
});

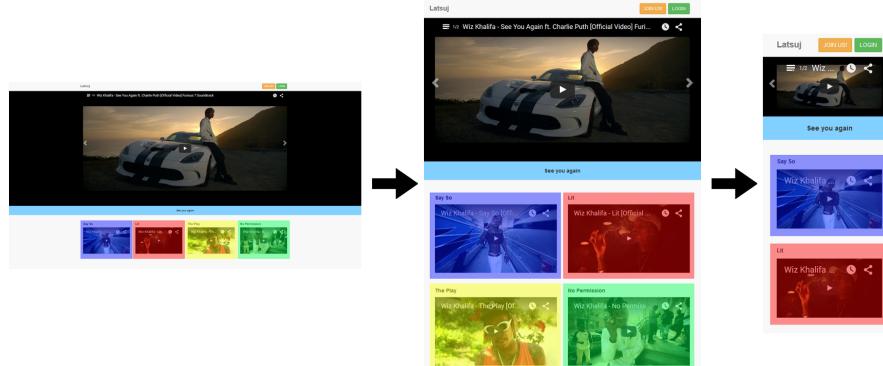
```

2.6 Clips vidéos supplémentaires

Les clips vidéos supplémentaires sont les vidéos qui se trouvent en bas de la page. Ces vidéos sont au nombre de 4 à grandes résolutions et au nombres de 2 à basses résolutions. Pour bien comprendre pourquoi j'ai fait ce choix, 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.



Pourquoi ai-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 calcule 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.

Que cela soit sur Bootstrap ou Polymer, nous savons normalement comment faire cela à partir des informations et techniques que nous avons déjà évoqué jusqu'ici. Cette partie reprend le positionnement et l'affichage ou le non affichage de certaines informations en fonction de la largeur de la fenêtre.

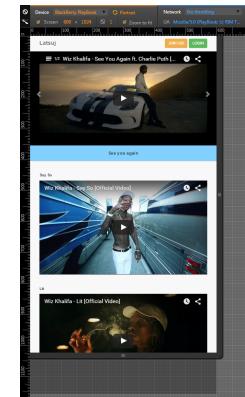


FIGURE 16 – Affichage du site sous BlackBerry (via Google Chrome).

```

<nav class="navbar navbar-default navbar-static-bottom">
    <div class="container">
        <div class="row">
            <div class="col-xs-push col-sm-6 col-md-6 col-lg-3 text-center">
                <h2><strong>< ?php echo $pubs[0]['TITLE1'] ; ?></strong></h2>
                <div class="embed-responsive embed-responsive-16by9">
                    <iframe class="embed-responsive-item" src="https://www.youtube.com/embed/
< ?php echo $pubs[0]['LINK1'] ; ?> ?showinfo=0&controls=0"></iframe>
                </div>
            </div>
            <div class="col-xs-push col-sm-6 col-md-6 col-lg-3 text-center">
                <h2><strong>< ?php echo $pubs[0]['TITLE2'] ; ?></strong></h2>
                <div class="embed-responsive embed-responsive-16by9">
                    <iframe class="embed-responsive-item" src="https://www.youtube.com/embed/
< ?php echo $pubs[0]['LINK2'] ; ?> ?showinfo=0&controls=0"></iframe>
                </div>
            </div>
            <div class="hidden-xs col-sm-6 col-md-6 col-lg-3 text-center">
                <h2><strong>< ?php echo $pubs[0]['TITLE3'] ; ?></strong></h2>
                <div class="embed-responsive embed-responsive-16by9">
                    <iframe class="embed-responsive-item" src="https://www.youtube.com/embed/
< ?php echo $pubs[0]['LINK3'] ; ?> ?showinfo=0&controls=0"></iframe>
                </div>
            </div>
            <div class="hidden-xs col-sm-6 col-md-6 col-lg-3 text-center">
                <h2><strong>< ?php echo $pubs[0]['TITLE4'] ; ?></strong></h2>
                <div class="embed-responsive embed-responsive-16by9">
                    <iframe class="embed-responsive-item" src="https://www.youtube.com/embed/
< ?php echo $pubs[0]['LINK4'] ; ?> ?showinfo=0&controls=0"></iframe>
                </div>
            </div>
        </div>
    </div>
</nav>

```

Sous Bootstrap, il faut donc simplement définir le nombre de colonne utilisé par chaque petit bloc. La moitié seront défini comme « hidden » comme on peut l'observer sur le code. Sous Polymer, cela sera beaucoup plus rapide car cela reprend une grosse partie des modules qui ont été défini précédemment, ainsi le code devient plus facile à implémenter car il n'y a presque pas de nouveaux modules :

```

<my-bar-pubs>
  <my-container>
    <my-col-25-100>
      <resp-div name=<?php echo $pubs[0]['TITLE1'];?>" link="https ://www.youtube.com/embed/<?php echo $pubs[0]['LINK1'];?> ?showinfo=0&controls=0"></resp-div>
    </my-col-25-100>
    <my-col-25-100>
      <resp-div name=<?php echo $pubs[0]['TITLE2'];?>" link="https ://www.youtube.com/embed/<?php echo $pubs[0]['LINK2'];?> ?showinfo=0&controls=0"></resp-div>
    </my-col-25-100>
    <my-col-hidden-900>
      <my-col-25-100>
        <resp-div name=<?php echo $pubs[0]['TITLE3'];?>" link="https ://www.youtube.com/embed/<?php echo $pubs[0]['LINK3'];?> ?showinfo=0&controls=0"></resp-div>
      </my-col-25-100>
    </my-col-hidden-900>
    <my-col-hidden-900>
      <my-col-25-100>
        <resp-div name=<?php echo $pubs[0]['TITLE4'];?>" link="https ://www.youtube.com/embed/<?php echo $pubs[0]['LINK4'];?> ?showinfo=0&controls=0"></resp-div>
      </my-col-25-100>
    </my-col-hidden-900>
  </my-container>
</my-bar-pubs>

```

Ce code reproduit donc le comportement précédent sur Polymer. Il n'y a rien normalement qui ne vous soit inconnu ici à part le « my-bar-pubs » mais il s'agit simplement d'une barre avec une couleur en fond comme on peut le voir ci-dessous :

```

<dom-module id="my-bar-pubs">
  <template>
    <style>
      :host {
        background-color : #f8f8f8 ;
        text-align : center ;
        float : left ;
        width : 100% ;
        box-sizing : border-box ;
      }
    </style>
    <content></content>
  </template>
  <script>
    Polymer({
      is : "my-bar-pubs"
    });
  </script>
</dom-module>

```

Petit à petit au travers de nos exemples, on peut remarquer que le développement sous Polymer

devient de plus en plus rapide car nous pouvons réutiliser nos éléments. Le code se réduit alors à du simple HTML.

2.7 Vidéo principale

Il s'agit du clip musicale principale, celui en plein centre de l'application. Pour afficher la vidéo, j'ai utilisé la balise IFRAME. Cependant, il faut que cette balise s'adapte elle aussi à la largeur de la fenêtre. Pour cela, on l'incorpore dans une balise DIV qui se redimensionne. Sous Bootstrap, j'ai utilisé les classes fournies : « embed-responsive » et « embed-responsive-16by9 ». La première classe permet de réaliser le redimensionnement et l'adaptation de la vidéo à la largeur de la fenêtre. La deuxième classe est seulement pour un aspect design, elle permet de forcer la vidéo à prendre un ratio 16/9.

```
<div id="aze" class="embed-responsive embed-responsive-16by9">
<iframe class="movie embed-responsive-item" src="https://www.youtube.com/embed/< ?php
echo $movie[0]['CODE'] ; ?> ?rel=0&loop=1&hd=1&vq=hd1080&playlist=< ?php echo $mo-
vie[0]['CODE'] ; ?>&modestbranding=1&showinfo=0"></iframe>
</div>
```

Le lien est relativement long car j'ai utilisé l'API de Youtube pour réaliser quelques modifications de comportement sur la vidéo. Par exemple, étant grand fan de musique, il m'arrive d'écouter la même musique en boucle. Cette fonction est absente de youtube mais peut-être ajouté sur d'autre site via leur API. En ajoutant « loop=1 », on peux alors jouer la vidéo en boucle. De même, je trouve que sur youtube le nombre d'informations affichés sur la vidéo est trop importante et embrouille l'utilisateur plus qu'autre chose. Comme dit un peu plus haut, je souhaitais un site simple d'utilisation. J'ai donc effacé les informations superflues via la commande « showinfo=0 ».

Sous Polymer, le principe est le même sauf que nous allons créer un module pour afficher la vidéo. Le code dans l'index se réduit une fois le module implémenté à une simple ligne de code :

```
<my-movie      link="https://www.youtube.com/embed/< ?php      echo      $mo-
vie[0]['CODE'] ; ?> ?rel=0&loop=1&hd=1&vq=hd1080&playlist=< ?php      echo      $mo-
vie[0]['CODE'] ; ?>&modestbranding=1&showinfo=0"></my-movie>
```

```

<dom-module id="my-movie">
  <template>
    <style>
      :host {
        position :relative ;
        display :block ;
        height :0 ;
        padding :0 ;
        overflow :hidden ;
        padding-bottom :56.25
      }
      iframe {
        position : absolute ;
        top : 0 ;
        left : 0 ;
        width : 100% ;
        height : 100% ;
        padding-bottom : 56.25% ;
      }
    </style>
    <iframe src="{{link}}" frameBorder="0"></iframe>
  </template>
  <script>
    Polymer({
      is : "my-movie",
      properties : {
        link : String
      }
    });
  </script>
</dom-module>

```

2.8 Le menu

Le menu est la partie à droite du site qu'il est possible d'afficher ou non. Par défaut, cette partie est caché pour les même raison que le bloc d'informations (nombre de points de fixation). Le menu prend aussi une taille différente en fonction du premier points de rupture que l'on va rencontré lorsque l'on réduit la fenêtre. Lorsque nous sommes à la plus basse résolution le menu prend 100% de la largeur de la fenêtre, j'ai trouvé que cela était ce qu'il y avait de mieux au niveau utilisation sans abîmer la simplicité du site et l'ergonomie.

Sur les deux technologies, j'ai utilisé la même technique. L'ensemble du site est englobé dans une balise DIV dont la largeur peut varier via un script JavaScript. De l'autre, le menu est aussi compris dans une balise DIV qui là aussi à sa largeur qui pourra varier entre 0 et un nombre dépendant de la largeur pris par le contenu du site.

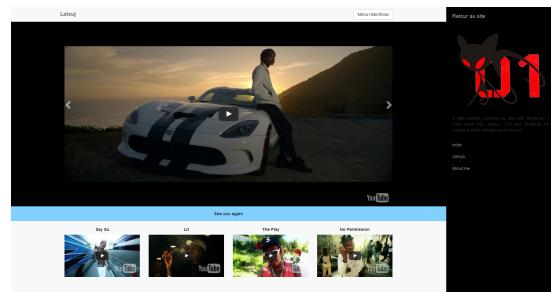


FIGURE 17 – Le menu à droite sur la version PC

```

...
<body>
  <div id="content" style="width :100%; height :auto; float : left;">
    ... (contenu du site)
  </div>
  <div id="menu" style="width :0%; background :#000000; float : right;">
    ... (menu)  </div>
</body>
...

```

```

...
<body>
  <my-content>
    ... (contenu du site)
  </my-content>
  <my-menu>
    ... (menu)  </my-menu>
</body>
...

```

Ce code ci-dessus montre la manière dont j'ai positionné les balises pour réaliser ce menu. Sous Bootstrap, il n'y avait rien pour réaliser cela de manière "Bootstrapsienne". J'aurai pu mettre le code dans un document style mais j'ai préféré éviter cela pour bien montrer que Bootstrap ne permet pas d'effectuer cela de manière simple. Notons par ailleurs qu'ici, j'ai découvert aussi un problème ennuyant. Il n'est pas possible d'utiliser le système de grille de Bootstrap lorsque l'on définit un élément comme « fixed ». Ce que je voulais faire au départ, j'ai donc dû me raviser.

Si vous avez suivi l'intégralité des explications ci-dessus, vous êtes maintenant apte à créer l'application de vos rêves en utilisant Bootstrap ou Polymer. Cependant pour décider quelles applications est la plus adapté à vos besoins, il faudra réfléchir en fonction des utilisateurs que vous viser. Les deux technologies n'étant pas supporté de la même manière.

3 Outils de développement et test

3.1 Outils de développement

Pour réaliser mon site adaptatif, j'ai eu besoin de plusieurs outils. Chacun ayant une part plus ou moins importante, je vais partir du plus utile dans mon exemple au moins utile mais qui a tout de même eu un rôle à jouer un moment ou à un autre.

L'outil que j'ai le plus utilisé, Eclipse, mon environnement de développement intégré préféré. Un IDE (Integrated Development Environment) est un outil qui permet d'augmenter la productivité d'un développeur. Tous les boutons, toutes les fonctions qui se trouvent à l'intérieur de ce logiciel ont été pensé pour le développeur. Ayant travaillé de nombreuses fois et pour différents projets avec ce logiciel, j'ai mes propres repères et cela me permet d'être très rapide lors de la phase d'implémentation.

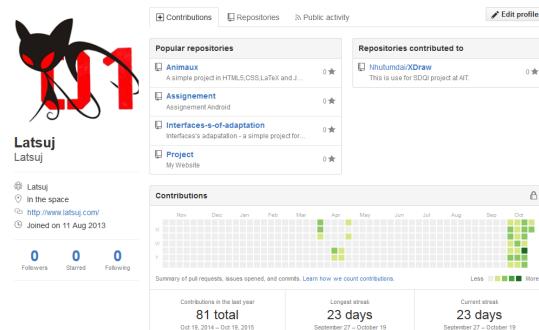


FIGURE 19 – Le logo d'éclipse

Les API (Application Programming Interface) de Bootstrap ainsi que Polymer ont été d'une aide précieuse. Une API est un service web qui donne une description plus ou moins poussé sur un programme ou langage. Elles sont souvent accompagnées de tutoriels et exemples qui permettent de comprendre rapidement comment fonctionne la technologie. Grâce à ces API, j'ai pu développer sans trop de difficulté mes deux exemples.

Enfin, j'ai utilisé à moindre échelle d'autre outils pour finaliser certains détails. J'ai donc utilisé **JQuery** pour effectuer mon carousel sous Polymer. Bootstrap utilise aussi cette bibliothèque pour effectuer leur propre carousel. J'ai aussi utilisé **PhpMyAdmin** et **Wamp**. Comme certaines pages étaient identiques en tout points et seul le contenu différait, j'ai jugé bon d'automatiser certaines tâches en utilisant un peu de PHP.



FIGURE 18 – Le logo d'éclipse

Depuis ce qui s'est passé lors de mon rattrapage en SI3, je ne me sépare plus de github lorsque je travaille sur un projet. Si jamais mon PC tombe en panne un jour avant la date limite de rendue (ce qui m'est arrivé), j'aurais toujours quelques choses à montrer. Cela me permet aussi de voir mes progrès sur mon projet. De revenir à une version antérieure si jamais une erreur inconnue s'est glissé dans le code... Ce petit logiciel anodin insoupçonné peut devenir un vrai sauveur de temps si un problème survient. Je regrette une seule chose : ne pas l'avoir utilisé plus tôt !



FIGURE 20 – Les APIs

3.2 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.

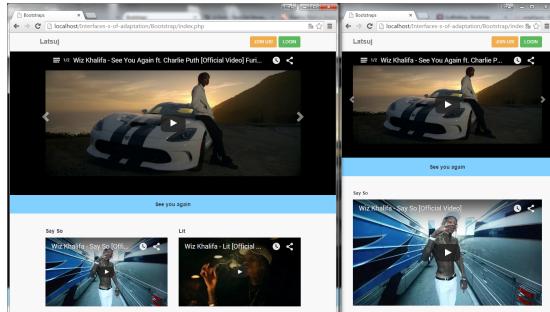


FIGURE 21 – Redimensionnement de la fenêtre sous windows

Polymer, on ne peut tester 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 coté 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

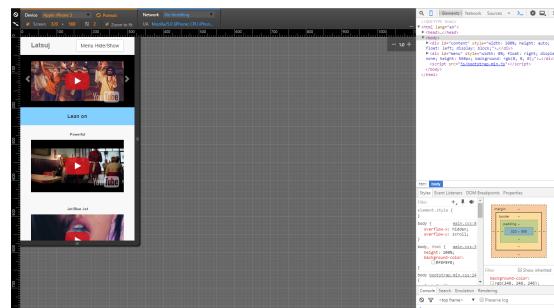


FIGURE 22 – Test au format iphone 5

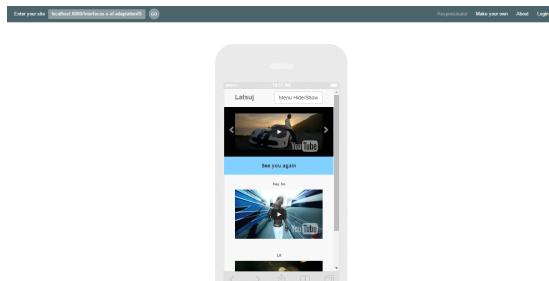


FIGURE 23 – Responsinator.com

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 sur ce site. 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.

4 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 vrai 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's_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 (media queries sur éléments et non par rapport au viewport).

<http://ianstormtaylor.com/media-queries-are-a-hack/>
<http://www.smashingmagazine.com/2013/06/media-queries-are-not-the-answer-element-query-polyfill/>

Le champs visuels

Le champs visuels est une aide précieuse pour concevoir une interface.

<http://sophie.raufaste.free.fr/Saccades%20fixations%20et%20champs%20visuel.htm>