

Adaptation des Interfaces

Rapport final

Projet Burger Express

Membres du groupe :

- DIB Dominique , dominique.dib@gmail.com , Bootstrap RWD
- GIANGRASSO Rémy , giangrasso.remy@gmail.com , Ionic CrossPlatform
- GRILLO Thomas , thomas.a.grillo@gmail.com, Angular 2, Web Components
- ELTAHER Randa , r.nasreldin11@gmail.com , Pure CSS RWD

Introduction	3
Introduction au RWD	4
Pure.CSS	4
I. Installation	5
II. Les méthodes de la page web	6
III. Test de l'adaptation	13
Bootstrap	14
I.Installation et mise en place:	14
II.Outil de développement et de tests:	15
III.Exemples et adaptations:	15
IV. Tests des capacités d'adaptation	23
Ionic	25
I. Installation et déploiement de l'exemple	25
II. Exécution de l'exemple	26
III Outil de développement et de tests	26
IV Réalisation de l'exemple	27
Angular2	37
I. Tâche implémentée	37
II. Installation de l'environnement de développement	37
III. Création de l'application	38
Création initiale avec angular-cli	38
Structure	39
Components	39
Services	43
Routing	44
Angular Material	44
Google Maps API	44
Déploiement	45
Avantages et inconvénients	46
Conclusion	48

Introduction

Dans le cadre de ce projet nous avons choisi de faire un site vitrine d'un restaurant fictif *Burger Express*.

L'idée s'inspire d'un site proposant le même service. Il s'agit de représenter les composants d'un site vitrine de burger tout en ayant la visualisation la plus ludique et simple possible.

Chaque technologie (Ionic, Angular2, Bootstrap, PureCSS) doit répondre à une problématique posée par le contexte, Ionic pour l'interface mobile par exemple.

Introduction au RWD

Responsive Web Design est essentiel pour bien représenter une page web sur tous les dispositifs quel que soit la taille de l'écran, en utilisant HTML et CSS. Normalement les pages web peuvent être visualisées par des différents dispositifs (desktops, tablettes, smartphones), donc, une page web doit être bien présentée, évidente, et facile à utiliser quel que soit le dispositif .

Donc, le contenu doit bien s'adapter sur les appareils ayant d'écran plus petit. De même, on doit prendre en considération la notion de View Port. C'est la région visible d'un page web pour l'utilisateur. Donc, il varie d'un dispositif à un autre. Avant les smartphones et les tablettes, les pages web sont conçues pour l'écran d'un ordinateur, donc qui ont eu une taille fixe et une conception statique. Pour un développeur doit donc, dans un premier temps, prendre le contrôle de ce View Port. Il doit intégrer le <meta> tag spécifique dans les pages web.

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

De plus, il doit prendre en considération que l'utilisateur va rouler sur un site verticalement et pas horizontalement. Toujours, il s'agit d'une conception nommée « Mobile First », c'est le fait de concevoir ou faire le design pour un téléphone mobile avant de le faire pour un desktop. Ce qui va permettre l'affichage des pages web plus rapide sur les téléphones mobiles. Ce qui signifie qu'on va faire des changements importants dans notre CSS, ce qui va nous introduire la notion de CSS FRAMEWORKS.

Plusieurs proposent Responsive Design. Ils sont gratuits et faciles à utiliser et ont tous des principes en commun. Ils sont basés sur la notion de Grid View. Ce qui signifie que la page est divisée en colonnes. Il y a toujours 12 colonnes, ayant de 100% largeur. Parmi les différents Framework utilisables, chacun a ses avantages et ses inconvénients.

Pure.CSS

Mise en place par YAHOO, on parle de modules CSS responsives, ayant ainsi de fichiers de taille néanmoins petite (4KB), ce qu'on peut considérer comme un avantage. Il est ainsi basé sur Normalize.css, mettant en place des différents « layout » et des patterns pour servir au Styling des éléments des applications natives. Les styles sont minimaux pour que chacun peut créer leurs styles en implémentant leurs applications. En fait, le choix de Pure CSS est basé sur l'idée que ce Framework utilise généralement HTML et CSS comme des langages suffisants pour créer des classes et des méthodes qui servent à créer à la fin des applications ayant des interfaces intéressantes, mettant en place par exemple des animations, des transformations, et de navigation. Donc, il ne consiste pas une connaissance importante par exemple de JQuery ou du JavaScript. C'est pourquoi il y a des limites, et des fonctionnalités ou on peut trouver Pure CSS insuffisant. Par ailleurs, on peut utiliser certains fichiers js pour le faire fonctionner sur les anciens browsers.

En outre, on doit tenir en compte que Pure CSS s'adapte mieux que certains autres Frameworks sur les écrans de très grandes tailles. Au moins, la majorité de ses classes n'ont pas besoin des méthodes supplémentaires pour s'adapter aux grands écrans.

I. Installation

Tout d'abord, on doit ajouter « Pure » dans notre page à travers « Yahoo Free CDN ». On ajoute ce lien dans le code, précisément dans la partie de <head>.

```
<link rel="stylesheet"href="http://yui.yahooapis.com/pure/0.6.0/pure-min.css">
```

Puisqu'il s'agit d'un système responsive et « Mobile First », on doit aussi attacher les liens nécessaires pour le fonctionnement de ces propriétés, ainsi, pour la mise en place des classes de Grid. De plus, c'est important de télécharger les classes de Pure.

```
<!--[if lte IE 8]>-->
```

```
  <link rel="stylesheet"
```

```
href="http://yui.yahooapis.com/pure/0.6.0/grids-responsive-old-ie-min.css">
```

```
<![endif]-->
```

```
<!--[if gt IE 8]><!-->
```

```
  <link rel="stylesheet"
```

```
href="http://yui.yahooapis.com/pure/0.6.0/grids-responsive-min.css">
```

```
<!--<![endif]-->
```

Ainsi, des autres liens nécessaires pour la création des éléments responsives.

```
<link rel="stylesheet"
```

```
href="http://yui.yahooapis.com/combo?pure/0.6.0/base-min.css&pure/0.6.0/grids-min.css&pure/0.6.0/grids-responsive-min.css">
```

II. Les méthodes de la page web

Navigation bar

Les pages web commencent souvent par une barre de navigation tout en haut de la page, ou il y en a beaucoup de choix pour le visiteur du site ou l'utilisateur normalement. Ce bar peut être considéré comme un menu contenant une liste des liens afin de naviguer sur le site et découvrir les autres pages. Afin de créer des menus responsives, Pure a déjà donné des Layouts afin d'aider un développeur à implémenter ce bar, ou il y a plusieurs types :

- Responsive Vertical Menu
- Responsive Horizontal-Scrollable Menu
- Responsive Horizontal-to-Vertical Menu

Dans notre conception, le second choix semble bien et aussi compatible avec les autres éléments de la page. On a utilisé dans notre code des classes de Pure déjà défini comme : pure-menu pure-menu-horizontal pure-menu-scrollable custom-menu custom-menu-bottom custom-menu-tucked pure-menu-item.

On remarque qu'ils ont la forme d'une liste. Pour la rendre responsive, on note que le bar horizontal n'apparaît jamais. Ce qui nécessite une partie du code en JavaScript, et on la situe dans l'élément script après le code de HTML. On peut considérer ça comme une limite pour Pure CSS. Ce qui nécessite d'écrire cette partie de la part du développeur.

```
<div class="custom-menu-wrapper">
  <div class="pure-menu custom-menu custom-menu-top">
    <a href="#" class="pure-menu-heading custom-menu-brand"> BURGER EXPRESS</a>
    <a href="#" class="custom-menu-toggle" id="toggle"><s class="bar"></s><s class="bar"></s></a>
  </div>
  <div class="pure-menu pure-menu-horizontal pure-menu-scrollable custom-menu
custom-menu-bottom custom-menu-tucked" id="tuckedMenu">
    <div class="custom-menu-screen"></div>
    <ul class="pure-menu-list">

      <li class="pure-menu-item"><a href="#" class="pure-menu-link">Accueil</a></li>

      <li class="pure-menu-item"><a href="#" class="pure-menu-link">Menu</a></li>

      <li class="pure-menu-item"><a href="#" class="pure-menu-link">Branches</a></li>
      <li class="pure-menu-item"><a href="#" class="pure-menu-link"></a></li>
      <li class="pure-menu-item"><a href="#" class="pure-menu-link">Mon panier</a></li>
      <li class="pure-menu-item"><a href="#" class="pure-menu-link">Mon compte</a></li>
    </ul>
  </div>
</div>
```

bar responsive

Carrousel :

On peut trouver plusieurs noms ou notation de cette fonctionnalité comme Image Slider, Full screen Slider. Ils ont tous à peu près les mêmes fonctions et même étapes d'implémentation. Notre but ici est de créer ce genre de fonctions en utilisant uniquement HTML et CSS et qu'ils soient capable d'assurer cette fonctionnalité mais dans le cadre de Framework Pure CSS. Donc, on n'utilise pas de JavaScript. Ainsi, il doit être responsive.

· Markup :

On peut la nommer comme ça, c'est l'étape de la création des sliders comme des listes, et on peut spécifier notre `<input>` comme étant radio par exemple. C'est l'étape de la création du code HTML. On note la structure nécessaire à la création. On met en place le carrousel de type "container" avec la navigation prev/next.



```
<!--Slider-->
```

```

    <div class="carousel-wrapper" style="height: 800px;">
    <span id="target-item-1"></span>
    <span id="target-item-2"></span>
    <span id="target-item-3"></span>
    <div class="carousel-item item-1">
```

```

    <h1>My burger is French</h1>
    <a class="arrow arrow-prev" href="#target-item-3"></a>
    <a class="arrow arrow-next" href="#target-item-2"></a>
    </div>
```

```
<div class="carousel-item item-2 light">
```

```

    <a class="arrow arrow-prev" href="#target-item-1"></a>
    <a class="arrow arrow-next" href="#target-item-3"></a>
    </div>
```

```
<div class="carousel-item item-3">
```

```

    <a class="arrow arrow-prev" href="#target-item-2"></a>
    <a class="arrow arrow-next" href="#target-item-1"></a>
```

```
</div>
</div>
```

· Styling :

C'est l'étape de la création du code CSS, ou autrement dit, la modification dans les classes déjà générées par Pure CSS, pour être compatible avec le code de la page web. On doit alors préciser que le carrousel va occuper la totalité de l'écran par la largeur.

```
.carousel-wrapper {
  position: relative;
}
/* line 8, ../sass/cari.scss */
.carousel-wrapper .carousel-item {
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;
  padding: 25px 50px;
  opacity: 0;
  transition: all 0.5s ease-in-out;
}
/* line 18, ../sass/cari.scss */
.carousel-wrapper .carousel-item .arrow {
  position: absolute;
  top: 0;
  display: block;
  width: 50px;
  height: 100%;
  -webkit-tap-highlight-color: transparent;
  background: url("../arrow.png") 50% 50%/20px no-repeat;
}
/* line 27, ../sass/cari.scss */
.carousel-wrapper .carousel-item .arrow.arrow-prev {
  left: 0;
}
/* line 31, ../sass/cari.scss */
.carousel-wrapper .carousel-item .arrow.arrow-next {
  right: 0;
  -webkit-transform: rotate(180deg);
  transform: rotate(180deg);
}
/* line 38, ../sass/cari.scss */
.carousel-wrapper .carousel-item.light {
  color: white;
}
/* line 41, ../sass/cari.scss */
.carousel-wrapper .carousel-item.light .arrow {
```



```

background: url("../arrow.png") 50% 50%/20px no-repeat;
}
/* line 54, ../sass/cari.scss */
.carousel-wrapper [id^="target-item"] {
  display: none;
}
/* line 58, ../sass/cari.scss */
.carousel-wrapper .item-1 {
  z-index: 2;
  opacity: 1;
  background: url("../Burger2.png") no-repeat;
  background-size: cover;
}

```

- **Navigation:**

On peut utiliser plusieurs moyens ou techniques qui permettent les navigations entre les slides, comme les check boxes, mais ici, on a utilisé des flèches qui sont définis dans le code de HTML, ce qui rend la navigation plus facile pour l'utilisateur.

- **Responsive Design:**

C'est le code de Media Query par CSS3 qui est responsable de rendre l'interface responsive en utilisant soit max-width et min-width.

```

@media (max-width: 480px) {
  /* line 47, ../sass/cari.scss */
  .carousel-wrapper .carousel-item .arrow, .carousel-wrapper .carousel-item.light .arrow {
    background-size: 10px;
    background-position: 10px 50%;
  }
}

```





Carousel Responsive

Grid :

Dans la troisième partie, on a mis une sorte de Carte pour que l'utilisateur peut choisir facilement ce qu'il a commandé, il s'agit de trois parties, constituant ses requêtes. La notion de Grid et ses classes sont fondamentales qu'on utilise de Pure CSS dans une page Web.

Dans Pure Grids, il existe 2 types de classes :

The grid class (pure-g) et unit classes (pure-u ou pure-u-*)

Le choix de l'unité correspond au choix de la largeur d'une partie dans la page. De plus, on peut diviser la page sous forme des différentes lignes de Grid en changeant la classe de Grid.

Notamment, la notion de Grid va nous introduire le fait de créer des parties de la page responsive tout en tenant en considération la taille de l'écran souhaité. Tout est déjà défini dans les classes de Grid. Dans Pure, c'est plus préférable d'utiliser l'unité de em à la place de px, mais le fait de changer c'est possible, on doit modifier ce qu'on nomme « Default Media Query ».

Surtout on ne doit pas oublier d'attacher le lien nécessaire pour assurer ces fonctionnalités.

```
<!--[if lte IE 8]>
```

```
<link rel="stylesheet"
```

```
href="http://yui.yahooapis.com/combo?pure/0.6.0/base-min.css&pure/0.6.0/grids-min.css&pure/0.6.0/grids-responsive-old-ie-min.css">
```

```
<![endif]-->
```

```
<!--[if gt IE 8]><!-->
```

```
<link rel="stylesheet"
href="http://yui.yahooapis.com/combo?pure/0.6.0/base-min.css&pure/0.6.0/grids-min.
css&pure/0.6.0/grids-responsive-min.css">
<!--<![endif]-->
```

On revient alors à notre page web. Afin de créer la carte, on a divisé la page en trois parties. Par créer une classe de Grid ou il y a des classes des unités a l'intérieur. On répète les mêmes étapes pour les trois parties divisées sous forme des « Boxes ». On choisit ainsi la taille de l'écran souhaité pour rendre ces parties de la page responsive.

```
<div class="pure-g">
  <div class="pure-u-1 pure-u-lg-1-3">
    <div class="l-box">
      <h2>
        Sandwich
      </h2>
      <p></p>
      <button class="button-xlarge pure-button">Commander</button>
    </div></div>
```

Sandwich



Commander

Footer:

La dernière partie de notre page, qui est aussi la meilleure pour mettre fin à une page web, ce qu'on nomme « le footer ». Ici, on doit le créer par nous-même, ce n'est pas défini dans le Framework de Pure, de plus, il existe un problème néanmoins célèbre, que cet élément n'est pas facilement attaché au bas de la page. On peut noter que cet élément n'est pas pratique à faire dans le Framework Pure.

Pour créer cet élément bien adapté et bien visible à l'écran de l'utilisateur, on doit le diviser sous forme de classes de Grids. On peut attacher ici la forme base de l'élément dans la page web.

```
<div class="footer">  
  <div class="pure-g">  
    <div class="pure-u-1 pure-u-lg-1-2">  
      <div class="pure-u-1 pure-u-lg-1-6">
```

```
<p></p>

</div>
<div class="pure-u-1 pure-u-lg-1-2"> <p></p></div>
</div>
<div class="pure-u-1 pure-u-lg-1-2">
<p>Contact us</p>
</div>

</div></div>
```

III. Test de l'adaptation

On essaye de toujours faire des tests après chaque application. On va sur le browser, google chrome par exemple, et on essaye de varier l'émulateur pour voir comment la page va s'adapter sur les différentes tailles de l'écran, quel que soit petit (moins de 700 x 700), ou grands (plus de 4000 x 4000). On remarque évidemment que la page est bien responsive, dépendante de ce qu'on a mis comme dimensions, et aussi elle s'adapte bien sur les grands écrans sans avoir besoin de créer beaucoup des fonctions néanmoins compliquées dans notre code.

Bootstrap

Dans le cadre du module Adaptation des interfaces, j'ai choisi de d'utiliser Bootstrap 3, afin de découvrir cette technologie dont j'ai jamais eu l'occasion d'utiliser.

Bootstrap est un framework **mobile first** open source HTML, CSS, JS conçu pour le RWD (responsive web design). Assez connu pour son système de grille et ces composants, Bootstrap 3 permet à une page web de s'adapter à l'écran d'un dispositif quelque soit sa taille.

Le Mobile First permet de réfléchir à ce qui est essentiel dans un site et à garder l'essentiel pour les petits dispositifs qui sont de plus en plus utilisés pour naviguer sur des pages web. Ensuite, lors du passage du site sur des écrans plus grands taille (tablette ou desktop), on ajoute ce qui peut être considéré comme secondaire.

I. Installation et mise en place:

La première chose à faire est de se rendre sur le site officiel du framework (<http://www.getbootstrap.com>). Une fois que nous sommes sur la page de démarrage du site, trois options de téléchargement s'offrent à nous.

Télécharger la première option "une version compilée et minimisée" de Bootstrap. (Ma version était la 3.3.7)

Ce fichier est une version allégée qui ira au plus grand nombre de projets, proposant les fichiers principaux nécessaires au bon fonctionnement du framework.

Maintenant que nous avons téléchargé Bootstrap, examinons sa structure. Dans le répertoire dist (signifiant "distant"), nous pouvons observer à première vue que le framework se résume à très peu de fichiers (seulement dix dans sa version compilée). Le dossier est séparé en trois sous-parties : css (les fichiers de styles), js (les fichiers JavaScript), fonts (les icônes proposées par Bootstrap).

Une fois le fichier index.html créer nous pouvons chargé les fichiers téléchargés grâce à la balise <LINK> dans le header:

```
<link href="css/bootstrap.min.css" rel="stylesheet">
```

L'attribut rel="stylesheet" précise que le document en question est une feuille de style externe. L'attribut href=" URL " donne l'URL de la feuille de style, dans ce cas, le chemin vers le fichier bootstrap.min.css.

On peut aussi avoir besoin d'une feuille de style externe, donc on ajoute cela au header:

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

Afin d'utiliser tous les composants que Bootstrap offre, nous devons inclure aussi les plugins JS et JQuery comme suit:

```
<script
```

```
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
```

```
<script src="js/bootstrap.min.js"></script>
```

Enfin, afin de faire des media queries pour certaines limitations de bootstrap, nous inclurons: `<meta name="viewport" content="width=device-width, initial-scale=1">`

II. Outil de développement et de tests:

J'ai réalisé ce projet sur l'éditeur open source **brackets**. Je testais sur un navigateur Chrome ou Firefox pour détecter quelques bugs de JavaScript à l'aide de Firebug.

Avec l'outil de développement de Chrome, nous pouvons modifier la taille de l'écran du dispositif ou même lancer la page web sur des écrans mobiles émulés.

III. Exemples et adaptations:

Plus tard dans ce tuto, je vais souvent utiliser le terme de container, comme son nom l'indique, cette balise sert à contenir des composants (élément HTML, JS ou d'autres composants que Bootstrap propose). Bootstrap 3 définit 2 types de conteneurs:

`<div class="container-fluid">` => 100% de la largeur de l'écran.

`<div class="container">` => a une largeur spécifique qui change de valeur à l'aide des media queries.

Afin de rendre la page web en mode Full screen, quelque soit la taille de l'écran, j'ai utilisé `container-fluid`, en imbriquant des lignes et des colonnes.

Les composants intégrés de Bootstrap 3:

- Mise en place d'une barre de navigation responsive:

A l'aide de la classe `<nav class="navbar navbar-default">` on peut créer une barre de navigation complètement adaptable à notre site de burger.

Cette classe contient une classe `"header"` et une classe `"nav navbar-nav"` contenant une liste d'items pointant sur différents liens du site (Se connecter, personnaliser le burger ...).

La classe contenant la liste des liens sera encapsulée par une classe

`<div class="navbar-collapse">` ayant comme attribut `collapse`, ce qui rend par défaut, une barre de navigation réduite sur l'écran d'un smartphone, alors qu'en élargissant on voit apparaître la barre de navigation complète:

```
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li> <a href="#section1">Preview</a> </li>
    <li> <a href="login.html">Se connecter</a> </li>
    <li> <a href="#section4">Personnalise ton burger</a> </li>
    <li> <a href="#section5">Footer</a> </li>
  </ul> <!--/nav navbar-nav -->
</div> <!--/navbar -->
```

Le fait d'avoir une classe header, va nous permettre de rajouter un bouton de menu ayant un attribut **data-toggle="collapse"** qui définit son rôle (afficher ou cacher le contenu de la barre de navigation sur les écrans de smartphone) et un autre attribut **data-target=".navbar-collapse"** désignant le contenu de la barre de navigation sur lequel ce bouton appliquera son rôle. Notons que collapse fait aussi partie des fonctionnalités offertes par bootstrap 3 lorsqu'on inclut la librairie JS. D'où le code du header suivant:

```
<div class="navbar-header">
  <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
    <span class="sr-only">Toggle navigation</span>
    <span class="icon-bar"></span>
    <span class="icon-bar"></span>
    <span class="icon-bar"></span>
  </button>
```

On peut rajouter à ça aussi, la classe **"navbar-brand"** qui permet de mettre le nom/logo du site à gauche de la barre de navigation. Bien sûr, tout ce code est encapsulé par un conteneur fluid, afin de rendre l'affichage en mode plein écran. Cette barre de navigation sera adaptable en mode collapse sur les écrans de smartphone et s'élargit en affichant tout son contenu sur les écrans de tablettes (ou plus grand).

écran tablette ou plus grand



Burger Express Preview Se connecter La carte Personnalise ton burger Footer

écran smartphone



Burger Express



Burger Express

Preview

Se connecter

La carte

Personnalise ton burger

Footer

- Composants JS: Mise en place d'un carrousel responsive.

Afin de réaliser notre carousel, bootstrap propose un plugin JS simple à inclure comme vue dans la section "**Installation et mise en place**". Il suffit juste de rajouter ce qui suit:

1) Pour la création du carousel :

```
<div id="carousel-example-generic" class="carousel slide" data-ride="carousel">
```

Nous ouvrons, ici, une balise `<div>` accompagnée des classes `.carousel` et `.slide`. L'attribut `data-ride="carousel"` permet d'établir la mise en forme. Ajouter un identifiant à cette balise pour garantir un bon fonctionnement des scripts liés au carousel.

2)

```
<ol class="carousel-indicators">
```

Déclaration par la suite de la liste qui va contenir les marqueurs ronds situés en bas au centre de l'image. Ceux-ci nous permettront de naviguer au fil du carousel. Nous affecterons ensuite un marqueur à chaque image.

3)

```
<li data-target="#exemple-carousel" data-slide-to="0" class="active"></li>
```

Premier marqueur. Cliquer sur celui-ci nous permettra d'afficher directement la première image que nous listerons par la suite (`data-slide-to="0"`). Grâce à la classe `.active`, cette image sera affichée par le navigateur une fois la page chargée. Il est également nécessaire de rappeler l'identifiant du carousel cible (`data-target="#exemple-carousel"`).

4)

```
<li data-target="#exemple-carousel" data-slide-to="1"></li>
```

```
<li data-target="#exemple-carousel" data-slide-to="2"></li>
```

```
</ol>
```

Compléter la liste.

5)

```
<div class="carousel-inner">
```

À l'intérieur de cette balise, nous allons décrire chaque diapositive composée d'une image et d'un contenu HTML.

6)

```
<div class="item active">
```

Description de la première diapositive. La mise en forme est assurée par la classe `.item`. Comme nous souhaitons que cette image soit affichée au chargement de la page, nous ajoutons la classe `.active`.

7)

```

```

 pour l'affichage de l'image

8) Ajout du contrôleur pour le carousel:

```
<a class="left carousel-control" href="#carousel-example-generic" role="button" data-slide="prev">
```

```
<span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>
```

```
<span class="sr-only">Previous</span>
```

```
</a>
```

```
<a class="right carousel-control" href="#carousel-example-generic" role="button" data-slide="next">
```

```
<span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>
```

```
<span class="sr-only">Next</span> </a>
```

Afin de pouvoir slider les images dans le carousel, les class `left/right carousel-control` permette de réaliser cela. En ajoutant un bouton représenter sous forme de icon (on peut trouver plein d'autres icons pour d'autre contexte dans la section `glyphicon` de la doc). Il est également nécessaire de rappeler la méthode appliqué du carousel cible (`data-slide="prev"`).

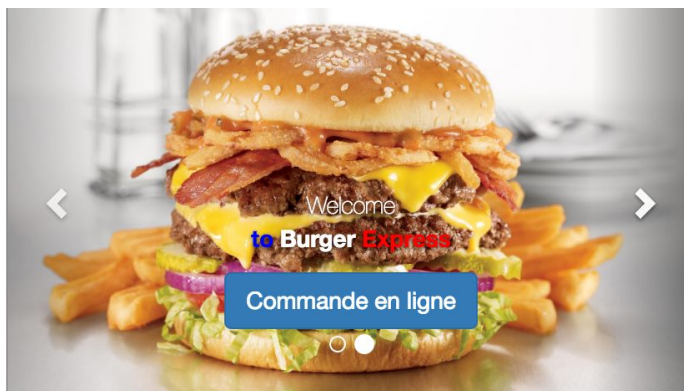
9) Bien sûr comme l'exemple précédent, on encapsule tout ce code dans un container-fluid afin de rendre le carrousel en plein écran sur le navigateur.

10) Nous avons un carrousel donc en plein écran, mais les éléments(image ou texte) que contient ce carrousel ne le sont pas, pour faire cela on rajoute dans le fichier style.css le code suivant: `.carousel .item img { width: 100%; }`

De plus, afin de rendre le texte à l'intérieure du carrousel responsive, nous utilisons des media queries, par exemple sur les écrans des tablettes on aura ce code:

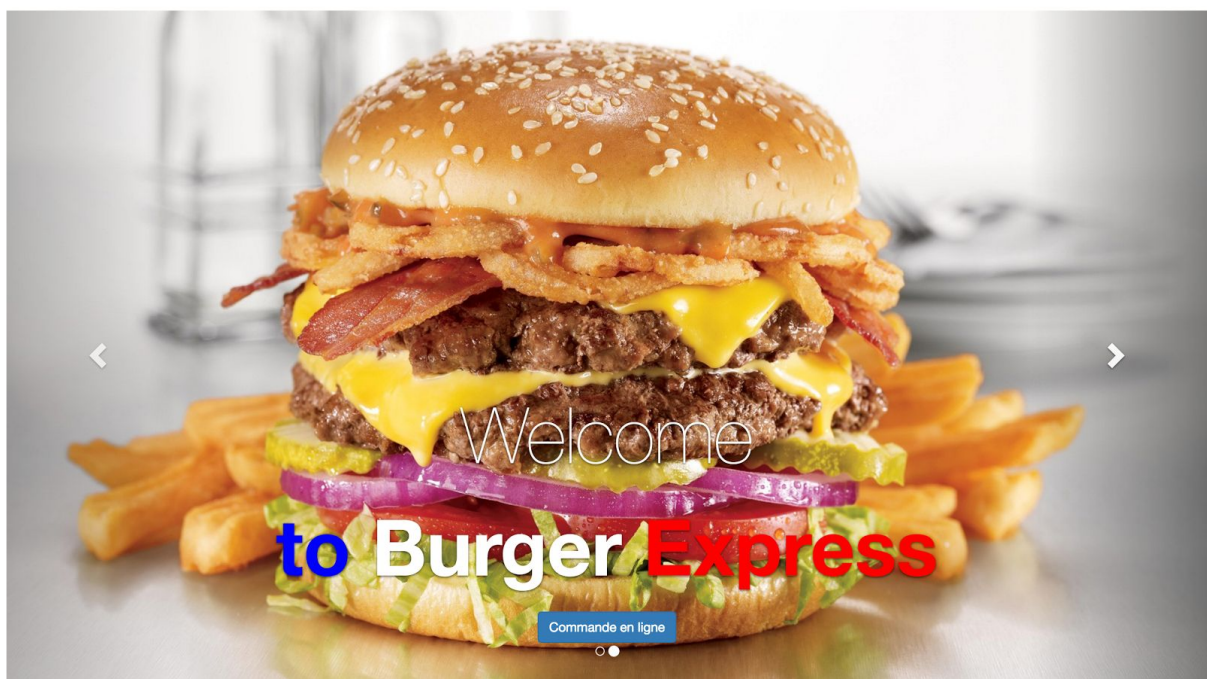
```
@media screen and (min-width: 992px) { .carousel-caption p { font-size: 3em; } }
```

Après toutes ses étapes le carrousel sera adaptable à toute taille d'écran ainsi que le texte présent à l'intérieure à l'aide des media queries, cela est faisable grâce au paramétrage de la taille d'écran présent dans l'outils développeur de Google Chrome.



écran smartphone

Ecran Desktop



- **Système de grille:**

Bootstrap considère 4 sortes de médias : les petits, genre smartphones (moins de 768 pixels), les moyens, genre tablettes (moins de 992 pixels), les écrans moyens (moins de 1200 pixels) et enfin les grands écrans (plus de 1200 pixels). Vous trouverez à la figure suivante un tableau pour illustrer les différences de réaction selon la catégorie.

Classe	col-xs-*	col-sm-*	col-md-*	col-lg-*
Valeur de référence	< 768 px (smartphone)	>= 768 px (tablette)	>= 992 px (desktop)	>= 1200 px (large desktop)

Exemple Footer:

Afin de réaliser le footer, on commence par mettre les composants row et column dans un conteneur fluid comme suit:

```
<div class="container-fluid" id="section5">
  <div class="row">
    <div class="col-sm-4 col-md-3">
      <ul class="list-unstyled">
        <li>Burger Express</li>
        <li><a href="#">About us</a></li>
        <li><a href="#">Contact & support</a></li>
        <li><a href="#">Enterprise</a></li>
      </ul>
    </div>
  </div>
```

id : me sert pour ajouter un lien dans ma barre de menu vers cette section, et pour pouvoir mettre en forme cette partie en utilisant #section5 dans ma feuille de style css.

col-sm-4 : la liste présente à l'intérieure de cette classe va occuper 4 colonnes de large (sur 12) à partir d'un smartphone

col-md-3 : la liste va occuper 3 colonnes de large à partir d'une tablette (jusqu'à la taille d'un écran sur desktop, 1200 px)

Si la taille de l'écran est inférieure à celle d'un smartphone, les règles des places occupées par les colonnes ne s'appliquent plus, chaque colonne sera affiché sur une ligne. En copiant le code de la div contenant les colonnes, 4 fois à la suite de ce code, on obtient comme résultat, 4 colonnes qui s'affichent sur un écran de taille desktop ou plus grand, 3 colonnes qui s'affichent sur une lignes et une en dessous pour l'écrans d'une tablette(même si toutes le colonnes appartiennent à la même "row") et les une à la suites des autres pour les écrans de smartphone (< 768px).

Pour régler le problème de l'écran de la tablette, bootstrap propose des attributs en plus pour la classe des colonnes,comme hidden-tailleClasse qui permet de cacher une

colonne/contenu pour un certain dispositif. Voici un exemple qui supprime la colonne qui apparaît seul à la ligne sur une tablette : `<div class="col-md-3 hidden-sm">`.

le système de grille s'applique sur d'autre exemple présent dans notre site: preview, la carte et personnalisation du burger.

Afin de réaliser une carte de la vitrine sous de 3 box (ayant la même hauteur) contenant des images et un bouton commandez, j'ai du aussi utiliser le système de grille.

Tout d'abord, **personnaliser le container** :

- html: `<div class="container-fluid my-custom-container" id = "section3">`
- CSS:

```
.container-fluid.my-custom-container { padding-top: 25px; padding-bottom: 25px; }  
.my-custom-container.row { padding-top: 25px; padding-bottom: 25px; background: rgba(255, 255, 255, 0.3); }
```

ensuite inclure les lignes et les colonnes comme l'exemple précédent, ainsi que mon box personnaliser:

- html:

```
<div class="row">  
  <div class="col-sm-4">  
    <div class="box my-custom-box">  
      <h2>Oh My God!</h2>  
        
      <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad. adipiscing elit, sed do eiusmod tempor</p>  
      <a href="#" class="btn btn-primary">Commander</a>  
    </div>  
  </div>
```
- CSS:

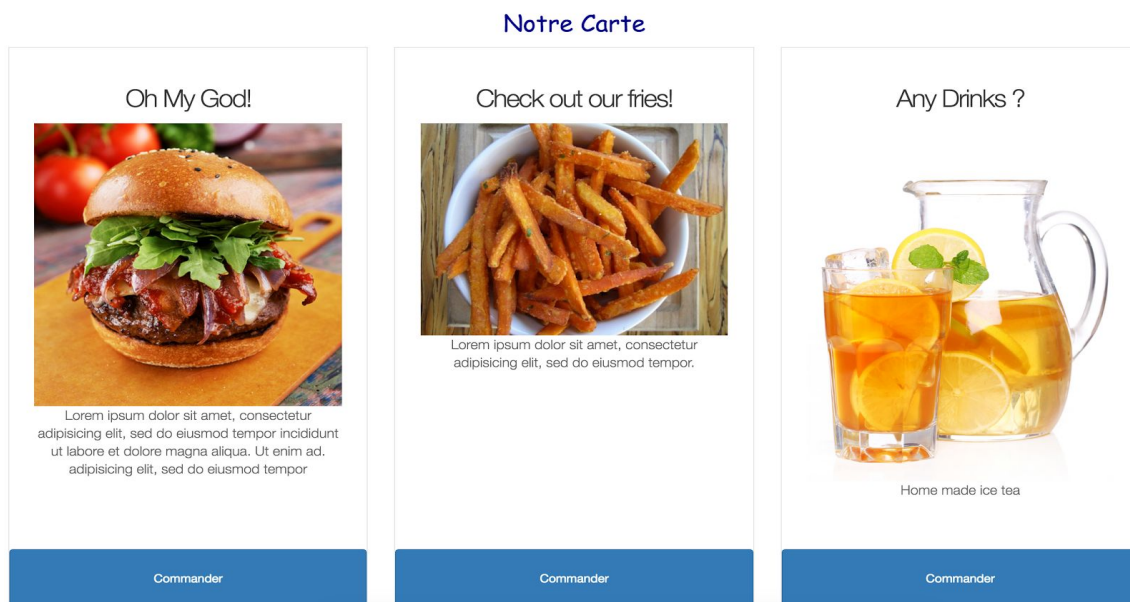
```
.box.my-custom-box {  
  padding: 25px 7% 0;  
  text-align: center;  
  position: relative;  
  border: solid 1px #e3e3e3;  
  background: rgba(255, 255, 255, 0.5);  
}
```

Afin de bien positionner les contenus des boxes, nous aurons aussi besoin d'appliquer du css sur ce contenu en utilisant `.box.my-custom-box img { ... } .box.my-custom-box p { ...}`.

Ainsi pour l'adaptation, `"col-sm-4"` en sorte d'affiche 3 boxes à partir de la taille d'un écran de tablette, pour les écran de smartphone, nous obtiendrons chaque box sur une ligne. Une des limitations de bootstrap est que la taille de la colonne(ici ca sera la box) dépendra du contenu, ce qui donne des boxes de tailles différentes. Pour régler ce problème nous allons utiliser du JS en appliquant la méthode `matchHeight()` sur tous les boxes. Faudra inclure dans le header :

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery.matchHeight/0.7.0/jquery.matchHeight.js"> $(document).ready(function(){ $(''.box').matchHeight(); }); </script>
```

Voici une prise d'écran sur un écran desktop:



- Exemple: Personnaliser votre burger

Afin de réaliser cet exemple, j'ai divisé la tâche en 2 parties:





Écran d'un smartphone:

```
<div class="container">
  <div class="row">
    <div class="col-xs-12 visible-xs">
      <table class="table table-striped table-bordered">
        <thead>
          <tr> <th>Produit</th> <th>Prix</th> <th class="td-actions"></th> </tr>
        </thead>
        <tbody>
          <tr> <td></td> <td>4,5€</td> <td> .. </td> </tbody>
        </tbody>
      </table>
    </div>
  </div>
  <button class="btn btn-danger deletetrow">
    <div class="glyphicon glyphicon-remove deletetrow" aria-hidden="true"></div></button>
```

Cette classe n'apparaîtra que sur les écran du smartphone à l'aide de l'attribut **visible-xs** , tout en affichant un tableau contenant des lignes représentant le produit avec les informations nécessaires. Le faite de rajouter un bouton deletetrow ou addrow permet d'interagir avec la page en ajoutant des produits. Tout ce qui est interaction nécessite du

Java Script donc ses méthodes ont été défini dans le header vue que bootstrap ne traite pas les données de la page.

Choisi tes propres ingrédients

Produit	Prix	
Ham	4,5€	
Cheese	1€	
Bacon	1.5€	
Lettuce	0.5€	

Add +

Voici l'écran sur un smartphone

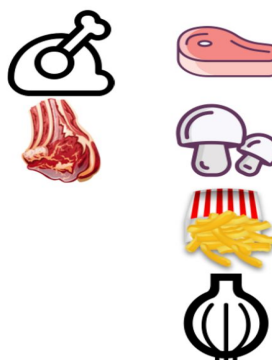
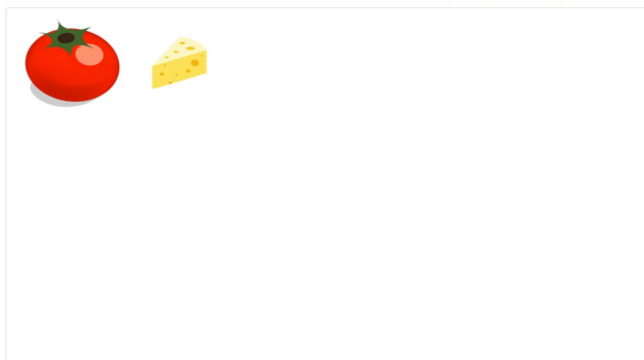
Sur l'écran d'une tablette ou plus grand:

```
<div class="container-fluid" id="section4">
  <div class="row">
    <p class="titleCommande"> Choisi tes propres ingrédients </p>
    <div class="col-sm-8 hidden-xs">
      <div class="panel panel-default">
        <div class="box panel-body" ondrop="drop(event)"
ondragover="allowDrop(event)">
          </div> <!-- panel-body -->
        </div> <!-- panel panel-default -->
      </div>
      <div class="box col-sm-2 hidden-xs">
         ... </div>
```

Dans ce code, on utilise la classe **panel** de bootstrap pour déposer les aliments, avec un **drop**, ce panel occupera 8 colonnes sur 12. A sa gauche nous aurons les aliments qu'on souhaite inclure à notre burger, des images ayant une méthode **drag** avec un attribut **draggable**, ces aliments sont affichés sur 2 colonnes. Les méthodes sont définis en JS et le fait de rajouter **box** au panel et aux colonnes, rend ses 3 éléments (panel et 2 colonnes d'aliments) de même taille. Cet exemple est adaptable sur toute taille d'écran, plus la taille de l'écran augmente, plus la hauteur du panel/des colonnes contenant les aliments diminue, mais tous ces éléments seront à la même hauteur.

Voici l'écran sur un desktop ou sur une tablette:

Choisi tes propres ingrédients



IV. Tests des capacités d'adaptation

Comme vue précédemment, j'ai fait en sorte de rendre ce site adaptable à toute taille d'écran avec l'outil développeur de chrome, en paramétrant la taille à des largeurs très grandes. Le fait d'utiliser les conteneurs fluid m'a permis d'afficher les colonnes ayant une largeur égale à la taille de l'écran. J'ai testé sur différent navigateur (Safari et firefox)

Ionic

À travers cette technologie, on s'est concentré sur la partie "commande d'un burger" de l'exemple, elle propose à l'utilisateur de choisir un burger, un dessert, le mode de livraison ainsi que le mode de paiement. Elle propose également d'afficher les "burgers Express" existant sur une carte Gmap , ainsi que la possibilité de partager une photo.

Ionic est un framework facilitant grandement le développement sur mobile, il a été aisé d'exploiter ses différentes possibilités comme l'accès aux capteurs et à l'API Google Map.

Il adapte également la vue en fonction de la dimension de l'écran et de l'os, les checkbox par exemple n'auront pas la même apparence ce qui est un vrai plus en développement car cela élimine une duplication massive de code.

A travers ce tutoriel nous expliquerons comment a été réalisé l'exemple, comment le déploie t'on, comment l'exécuter, et enfin la méthode de test d'adaptation.

I. Installation et déploiement de l'exemple

Ionic se repose sur le framework Cordova pour les accès aux capteurs du smartphone, il est donc nécessaire de l'installer :

```
"npm install -g cordova"
```

Puis installez Ionic :

```
"npm install -g ionic"
```

Vous pouvez à présent créer votre projet.

Créez et positionnez vous sur un dossier qui contiendra le dossier du projet, puis entrez la commande :

```
"ionic start NomDeLapplication"
```

Depuis le dossier du projet ajoutez la plateforme android pour pouvoir tester plus tard l'application sur cette plateforme :

```
"ionic platform add android"
```

Ensuite ajoutez deux plugins permettant la manipulation des capteurs,pour ce faire entrez les commandes suivantes :

```
"cordova plugin add cordova-plugin-geolocation"
```

```
"cordova plugin add cordova-plugin-camera"
```

Déposer le livrable, qui écrasera le dossier www du dossier que vous avez créé.

L'application peut enfin tourner correctement.

II. Exécution de l'exemple

Deux modes d'exécutions existent.

le premier sur navigateur web permet d'évaluer rapidement le rendu de l'application, on a également accès aux messages d'erreurs via la console de développement du navigateur. La commande exécutant l'application sur le navigateur :” ionic serve -g ”

Le deuxième consiste à exécuter l'application directement sur le mobile, pour déployer sur un smartphone il suffit de le connecter au pc et de lancer la commande, pour un mobile Android :

“ ionic build android && ionic run android ”

Pour un mobile IOS :

“ ionic build ios && ionic run ios ”

III Outil de développement et de tests

Les outils de développement utilisés sont Ionic, Cordova, npm, Firefox et notepad++

Les outils de tests sont un smartphone Android et Firefox avec le module Firebug.

Firefox est utile pour rapidement obtenir un rendu et résoudre les bugs rapidement.

Le smartphone permet de tester le bon fonctionnement de l'application, la bonne integration de l'interface, des séquences de fenêtres, et du capteur GPS et de l'appareil photo, en effet le rendu peut être différent si on test sur Firefox ou sur une cible mobile.

IV Réalisation de l'exemple

L'exemple a été conçu sur 3 axes, un premier dédié à la commande d'un burger, un deuxième sur l'affichage des différents points de retrait d'une commande, et un troisième émulant une fonction de partage.

Première partie : Commande

Dans la première partie l'utilisateur progresse dans une série d'étapes avant la validation de la commande,

Choix du burger, du dessert, du mode de livraison, et mode de paiement.

Pour une interface claire et efficace pour les deux interfaces (IOS et Android), un bouton retour est fréquemment implanté puisque sur IOS il n'existe pas de touche retour.

On peut le voir dans cet exemple :



1) Sélection d'un burger

La sélection d'un burger se fait via une liste de checkbox, la validation se fait via le bouton posé en pied de page.

Le widget `<ion-checkbox>` a été employé afin d'automatiser la présentation de chaque burgers.

```
"<ion-checkbox ng-model="filter.red" ng-repeat="burger in burgers" type="item-text-wrap"
ng-click="burgerSelected({{burger.id}})" style="background-color: rgba(255,255,255,.70)">"
```

Au sein de la balise `<ion-checkbox>` seront listés tous les burgers existant, considérez `ng-repeat = "burger in burgers"` comme une boucle itérant sur tous les "burgers" chaque burger contient un id permettant d'être identifié.

Cet identifiant est utile lors de la sélection du burger associé, on peut le voir dans la fonction `ng-click` où on appelle la fonction `burgerSelected({{burger.id}})` en lui passant l'identifiant du burger sélectionné.

Un élément `<ion-footer-bar>` a été posé au pied de l'écran pour valider la sélection.

2) Sélection d'un dessert

La sélection d'un dessert est facultatif, le rendu du bouton validant ou non l'ajout d'un dessert change en fonction, Ionic est utile pour ce genre de situation -> l'affichage conditionnelle des composants.

On utilise le même balisage (`<ion-checkbox>`) avec les fonctions associées, `ng-repeat, ng-click...`

Pour générer la liste des desserts, où les desserts sont spécifiés dans le fichier `services.js`

```
"
  <ion-footer-bar class="bar-assertive" style="bottom:0px;" ng-show="!cookieCheck &&
!milkShakeCheck" ng-click="goPaiement()">
    <h1 class="title" style="text-align:center;">Sans dessert pour moi</h1>
  </ion-footer-bar>
  <ion-footer-bar class="bar-balanced" ng-show="cookieCheck || milkShakeCheck"
ng-click="goPaiement()">
    <h1 class="title" style="text-align:center;">Valider</h1>
  </ion-footer-bar>
"
```

la `ng-show` affiche ou non le bouton, si aucun dessert n'est sélectionné on affiche "*Sans dessert pour moi*", sinon "*Valider*", cette condition est vraie ou fausse en fonction de l'état

des variables `$scope.cookieCheck` et `$scope.milkShakeCheck` que l'on peut voir dans le contrôleur "commandDessert", contenu dans le fichier `controllers.js`

Le bouton "back" appelle la fonction `back`.

```
<button class="button icon-left ion-chevron-left button-clear button-light"
ng-click="back('cmdBurger')"></button>
```

On peut voir la fonction `ng-click="back('cmdBurger')"`, qui appelle `back` quand le bouton est pressé.

Il est facile de générer plusieurs vues, un développement en natif demanderait plus de temps.

3) Sélection du mode de livraison

On affiche cette fois deux radiobox où un seul choix est permis.

Lors de la sélection d'un choix la fonction `ng-click` est actionnée, appelant la fonction `serverSideChange`,

fonction éditée dans le fichier `controllers.js`, dans le contrôleur `commandModesLivraison`.

A la validation on affichera une carte Google Maps si il y a livraison, sinon on redirige directement vers le paiement.

4) Validation du lieu de livraison

Certainement l'une des parties les plus intéressantes du projet, l'emploi de l'API Google Map.

On souhaite également afficher la position du client. On récupère donc la position à l'aide de Cordova, et on marquera la position du client sur la carte.

Tout ceci se fait en plusieurs étapes.

I) Générer une clé API Google Maps JS

sur Google console, générez une clé, copiez la et collez là dans le fichier `index.html` de cette façon:

```
<script src="http://maps.google.com/maps/api/js?key= VOTRECLE &sensor=true"></script>
```

II) Ajouter un composant sur la page HTML : `<div id="map" style="height:80%;" data-tap-disabled="true"></div>`

III) Écrire dans le contrôleur de la page html associée

Voici le code implémentant la fonction de récupération de la position GPS ET de son affichage sur la carte Google Maps.

```

var geoSettings = {frequency: 30000, timeout: 100000,enableHighAccuracy: false};
var geo = $cordovaGeolocation.getCurrentPosition(geoSettings); // donne la position

geo.then(function (position) {
    var latLng = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
    var mapOptions = {
        center: latLng,
        zoom: 15,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };

    $scope.map = new google.maps.Map(document.getElementById("map"), mapOptions);
// Instanciation d'un carte Google

    var marker = new google.maps.Marker({
        position: latLng,
        map: $scope.map,
        animation: google.maps.Animation.DROP
    });

    marker.setMap($scope.map); // ajout du marqueur de la position du client sur la carte

},
function error(err) {
    $scope.errors = err;
}
);
geo; // appel provoquant la capture de la position et de son affichage

```

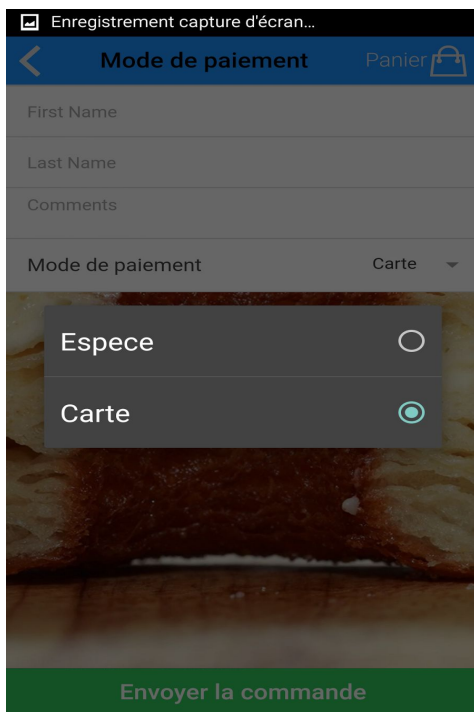
Deux boutons ont été ajouté pour la navigation, au dessus de ces deux widgets la carte Gmap.

A la page suivante vous trouverez le rendu de l'interface.



5) **Choix du mode de paiement et validation**

A cette étape finale on souhaite que l'utilisateur entre ses informations personnelles dans différents champs (Nom , Prénom) , et le mode de paiement.



```
<div class="list">
<label class="item item-input">
  <input type="text" placeholder="First Name">
</label>
<label class="item item-input">
  <input type="text" placeholder="Last Name">
</label>
<label class="item item-input">
  <textarea placeholder="Comments"></textarea>
</label>
```

```
<div class="list">
```

```
<label class="item item-input item-select">
  <div class="input-label">
    Mode de paiement
  </div>
  <select>
    <option ng-click="carte = false;">Espece</option>
    <option selected ng-click="carte = true">Carte</option>
  </select>
</label>
```

```
</div>
```

```
</div>
```

On visualise encore une fois l'adaptation de l'application -> le design du sélecteur qui emploi basé sur Android.

Deuxième partie : Affichage des différents restaurants

Dans cette partie on souhaite afficher une carte Gmap pour afficher la position des différents restaurants, la carte est cette fois plus responsive, en effet lorsque l'on veut cliquer sur le marqueur d'un restaurant une boîte de dialogue doit s'ouvrir en affichant le nom du restaurant et son adresse.

Pour réaliser cela il faut :

- Ajouter aux paramètres de la fonction du contrôleur \$ionicPopup.
- instancier une carte GoogleMap en donnant en paramètre l'id du div de la page HTML associée à son contrôleur, ce div contiendra la carte Google Maps.
- Générer un marqueur "marker"
- Ajouter un listener à `google.maps`
- En son sein un autre listener ayant en paramètre le marker sélectionné
- Puis définir la boîte de dialogue "alertPopup"

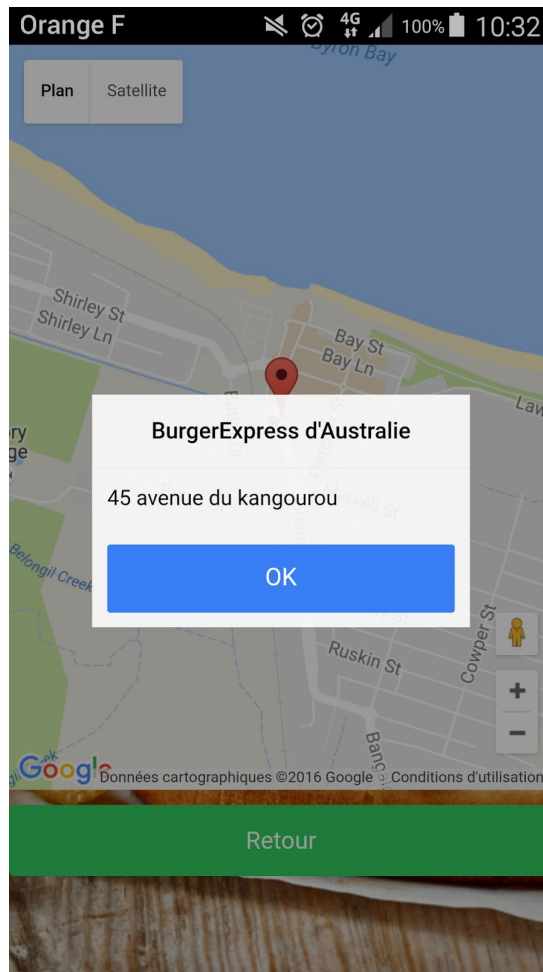
```
var latLng = new google.maps.LatLng(-28.643387, 153.612224);
var mapOptions = {
    center: latLng,
    zoom: 15,
    mapTypeId: google.maps.MapTypeId.ROADMAP
};

$scope.map = new google.maps.Map(document.getElementById("map"), mapOptions);

var marker = new google.maps.Marker({
    position: latLng,
    nom: "BurgerExpress d'Australie",
    adresse: "45 avenue du kangourou",
    map: $scope.map,
    animation: google.maps.Animation.DROP
});
google.maps.event.addListenerOnce($scope.map, 'idle', function(){

google.maps.event.addListener(marker, 'click', function () {
    var alertPopup = $ionicPopup.alert({
        title: marker.nom,
        template: marker.adresse
    });

    alertPopup.then(function(res) {
        // Custom functionality....
    });
});
});
```



Voici le rendu sur mobile, le marqueur a été sélectionné et affiche son contenu.

V Tests des capacités d'adaptation

Les tests se concentrent sur deux axes, l'adaptation en fonction des différentes dimensions des écrans et l'adaptation de l'interface selon la plateforme, les widgets n'auront pas le même rendu sur ios et Android.

Tests sur différents smartphone, un Galaxy Core Prim et un Galaxy S4 ont été utilisés puisqu'ils ont tous deux, une résolution, une diagonale de l'écran, et une version du système d'exploitation propres à chacun.

Il est malheureusement impossible de générer une application IOS sur un Windows.

Différents cas de tests ont été définis:

- Tests du rendu de l'interface sur les différents mobiles en mode portrait/paysage
- Test des accès aux capteurs(Appareil photo, GPS)
- Test des éléments responsive, Carte Google Maps, l'affichage conditionnel des boutons.

Troisième partie : Accès à l'appareil photo pour un partage

Dans cette partie nous souhaitons simuler le partage d'un instant dans un restaurant BurgerExpress.

Ce partage se base sur une capture photo, l'API Cordova nous permet de prendre une photo très simplement.

Temporellement l'interface se décompose en trois phases :

- Lancement de la capture photo
- Prise de la photo
- Affichage du rendu

On réutilise l'affichage conditionnel des boutons, dans la première phase on propose deux choix :

- Capturer une photo
- Revenir au menu

Si l'utilisateur prend une photo on affichera le rendu sur la meme fenetre (phase 3).

Deux autres choix se présentent alors à lui :

- Partager la photo
- Invalider la photo et revenir au menu

L'interface comporte une balise img qui s'affiche si une photo a été capturée (ng-show) Et trois boutons, pour partager, prendre une photo ou annuler.

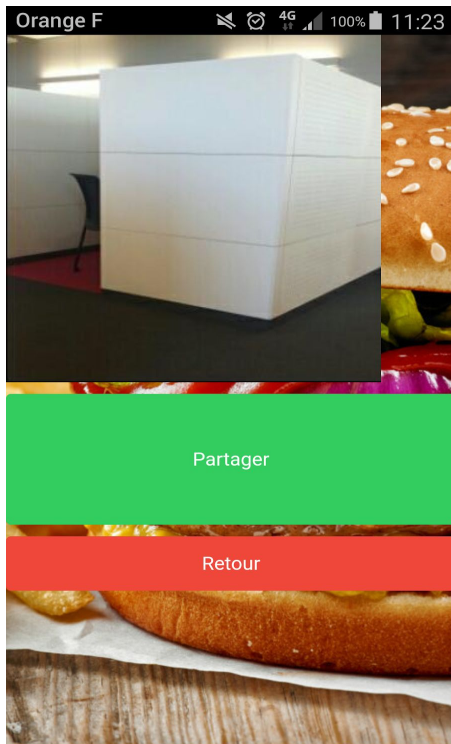
Techniquement pour lancer la capture photo il faut tout d'abord ajouter au paramètre du contrôleur du partage la variable \$cordovaCamera et inclure le code suivant :

```
$scope.takePicture = function() {
  var options = {
    quality : 75,
    destinationType : Camera.DestinationType.DATA_URL,
    sourceType : Camera.PictureSourceType.CAMERA,
    allowEdit : true,
    encodingType: Camera.EncodingType.JPEG,
    targetWidth: 300,
    targetHeight: 300,
    popoverOptions: CameraPopoverOptions,
    saveToPhotoAlbum: false
  };
```

```
$cordovaCamera.getPicture(options).then(function(imageData) {  
  $scope.imgURI = "data:image/jpeg;base64," + imageData;  
}, function(err) {  
  // ICI les erreurs  
});  
}
```

Le bouton de capture appellera la fonction **\$takePicture**.

Voici le rendu après capture d'une photo.



Angular2

Angular2 est le dernier framework créé par Google.

Il est l'évolution d'AngularJS, encore extrêmement populaire, mais ne garde de son prédécesseur que le nom.

Contrairement à AngularJS orienté responsive web design, celui-ci, pour des raisons de performances et de productivité, est orienté WebComponents.

Le WebComponents permet de créer des composants réutilisables ou bien d'utiliser ceux fournis par le navigateur, ce qui évite les imports de librairies externes.

Le langage recommandé pour développer avec ce framework est TypeScript. Même s'il est possible de développer directement en JavaScript, TypeScript est un code interprété qui permet de générer un code JavaScript sûr et qui permet notamment le typage statique des variables.

I. Tâche implémentée

La tâche qui m'incombe consiste à créer une interface d'administration qui permettra à l'utilisateur d'éditer la liste des menus proposés et d'éditer la liste des restaurants de la chaîne.

II. Installation de l'environnement de développement

Disposant d'un environnement Unix grâce à MacOSX, j'ai installé les programmes nécessaires au développement et au déploiement ainsi que leurs dépendances en lignes de commandes, d'autant que ces programmes sont eux-mêmes disponibles uniquement en lignes de commandes.

NodeJS et npm

NodeJS est le framework Javascript qui permet le développement côté serveur, c'est lui qui permettra le déploiement rapide de l'application en la lançant sur un micro-serveur embarqué.

npm est son gestionnaire de paquet et nous permet d'installer non seulement tous les modules nécessaires à l'application elle-même mais aussi les programmes d'aide au développement comme **angular-cli**.

Installation NodeJS et son package manager npm avec Homebrew :

```
> brew install node
```

angular-cli

angular-cli permet la génération rapide de toute la structure de l'application pour commencer à développer en ne se souciant que de nos composants et pas de leur environnement logiciel. Il permet aussi de générer des composants vides, des services, des tests... et de lancer l'application sur un mini-serveur !

Installation avec npm :

```
npm install -g angular-cli
```

-g pour l'installer avec toutes ses dépendances.

WebStorm

Webstorm est l'environnement de développement web selon JetBrains connu pour son IntelliJ (java). Il permet entre autre l'autocomplétion, interprète (fait de son mieux) le code TypeScript dans Angular2. Il intègre entre autre un terminal, npm et des utilitaires de versionning.

Firefox

Bien que cela soit insuffisant en production, j'ai effectué mes essais sur Firefox, uniquement, qui m'a permis de déboguer grâce à sa console et sa fonctionnalité "inspecter un élément".

III. Création de l'application

Création initiale avec angular-cli

La création du projet démarre avec l'outil angular-cli, présenté ci dessus.

Celui-ci me permet de créer l'intégralité de la structure de mon projet en une seule ligne de commande.

Pour cela je me place dans le répertoire où je souhaite créer mon projet et c'est parti !

```
> ng new burger-express
```

Ainsi à l'état initial mon projet comprend :

- un répertoire "**e2e**" pour les tests unitaires (non utilisés) ;
- un répertoire "**node_modules**" qui contient toutes les bibliothèques installées non seulement à l'initialisation du projet, mais aussi à chaque fois que l'on exécute la commande "*npm install*" ;
- le répertoire "**source**" où se trouve le code source de mon projet ;

- les fichiers de configuration, dont “package.json” qui décrit non seulement les dépendances du projet, mais aussi les scripts liés à l’exécution du projet et des tests.
- l’index.html qui redirige vers l’app.component ;

Structure

Afin de créer l’interface d’administration je vais m’appuyer sur la structure de base du répertoire src/ :

- **app.component** : le composant racine du projet. C’est lui qui appelle tous les autres ;
- **app.module.ts** : la classe qui déclare les composants, importe les modules des libraries externes et qui déclare les services en tant que providers afin de les exécuter au démarrage de l’application.

sur la création de quatre composants :

- **menu-entry.component** ;
- **menu-editor.component** ;
- **restaurant-entry.component** ;
- **restaurants-editor.component** ;

sur la création d’une classe **app-routing.module.ts** pour gérer les changements de “pages”.

Components

L’intérêt essentiel d’Angular 2 est son orientation WebComponent. Au delà du terme, c’est toute une philosophie de développement qu’il faut intégrer. WebComponent est un composant borné, qui comporte un template, une ou plusieurs feuilles de style et un fichier de logique qui implémente le comportement du composant et qui peut être écrit en TypeScript, en Javascript ou en Dart.

Pour ma part, j’ai choisi le TypeScript, pour une raison simple : je ne connaissais aucun des trois langages et l’essentiel de la documentation et des tutoriaux étaient écrits pour TypeScript.

Ces composants peuvent eux-même être composés de composants et interagir entre eux grâce à un système d’entrées sorties et de propagations de variables locales. Ainsi, on parle de composants parents et enfants.

Les entrées sorties sont des variables du composants, précédées par les décorateurs @Input et @Output. Quant aux variables locales, elles sont déclarées dans le template, précédées du caractère ‘#’ et se propagent dans tous les composants enfants. On peut également “mock” un enfant dans le parent grâce au décorateur @ViewChild. Ainsi on a accès à l’ensemble des variables de l’enfant (non utilisé dans ce projet).

La création de chaque composant est facilitée par l’utilisation d’angular-cli grâce à la commande :

```
> ng g component nom_du_composant
```

Exemple de code pour le component *menu-entry* :

```
import { Burger } from './menu-editor/burger';

@Component({
  selector: 'menu-entry',
  templateUrl: './menu-entry.component.html',
  styleUrls: ['./menu-entry.component.scss']
})

export class MenuEntryComponent {
  editable: boolean = false;
  @Input() burger : Burger;
  currentIngredient: string;

  constructor() {}

  modifyEntry() {}

  deleteEntry() {}

  validateEntry() {}

  addIngredient() {}

  removeIngredient(index: number) {
    if (index > -1) {
      this.burger.ingredients.splice(index, 1);
    }
  }
}
```

Un composant est donc composé d'un `@Component` qui intègre :

- un *selector* : nom du component dans le template l'appelant ;
- un *templateUrl* : le lien vers le template .html ou directement le temps .html en single ou multiline directement après l'item *template* (cette fois sans le suffixe *Url*) ;
- un *styleUrls* : tableau de styles .css ou .scss. Ou alors, idem que pour le template, le style directement après l'item *style* (sans le suffixe *Urls*).

menu-entry

Représente les tuiles qui constitueront le menu-editor.

Une menu-entry est composée de :

- un *nom* de burger,
- un tableau d'ingrédients,
- de l'url d'une image

déclarés dans le `.component.ts`.

Une menu-entry est composée de :

- un *nom* de burger,
- un tableau d'ingrédients,
- de l'url d'une image

déclarés dans le `.component.ts`.

Or éléments graphiques Material, le template d'une menu-entry est composée de :

- d'un `<button>` d'édition pour passer en mode édition ;
- d'un `<button>` de suppression de l'entry ;
- d'un `<button>` de validation de la modification de l'entry ;
- d'une `<input>` pour modifier le titre en mode édition ;
- d'une liste `` pour les ingrédients ;
- d'une autre `<input>` couplée à un `<button>` pour ajouter un ingrédient.

Angular2 permet d'échanger des informations entre le template et le TypeScript.

C'est comme cela que nous pouvons ajouter un ingrédient avec la fonction **addIngredient()** liée au bouton d'ajout grâce à l'évènement (click) comme suit :

```
<button md-icon-button color="accent" (click)="addIngredient()">
```

On peut également insérer du contenu à partir du template et le rediriger vers une variable comme ceci :

```
<md-input [(ngModel)]="currentIngredient" placeholder="Ingrédient"></md-input>
```

On appelle cela le data binding. L'`<input>` intègre son contenu directement dans la variable *currentIngredient*.

La variable *editable* (visible dans le code du component, plus haut) est utilisée au sein du template `.html` dans le cadre de directives conditionnelles **ngIf* qui permettent d'afficher les *inputs* et les boutons d'édition uniquement lorsque l'on est en "édition" (*editable=true*), et le label de nom uniquement lorsque l'on n'est pas en "édition" (*editable=false*).

```
<span *ngIf="!editable"><h3 >{{burger.nom}}</h3></span>
<span *ngIf="editable"><md-input [(ngModel)]="burger.nom"
placeholder="Burger"></md-input></span>
```

menu-editor

Le composant *menu-editor* sert de conteneur aux *menu-entries*. C'est lui qui appelle la fonction *getBurgers()* du service **menu-burgers.service.ts** pour remplir son tableau de burgers :

```
export class MenuEditorComponent implements OnInit {
  burgers: Burger[];

  constructor(private burgersService: MenuBurgersService) {}

  getBurgers() {
    this.burgersService.getBurgers().then(burgers => this.burgers = burgers);
  }

  ngOnInit() {
    this.getBurgers();
  }
}
```

Chaque burger du tableau est passé en paramètre au composant fils comme ceci :

```
<li *ngFor="let b of burgers" class="tuile"><menu-entry [burger]="b"></menu-entry></li>
```

Ce qui nous ramène au decorator `@Input` décrit plus haut. Le `<menu-entry>` fils du `<menu-editor>` attend un burger de la classe `Burger` en entrée (paramètre).

On peut aussi remarquer la présence de **ngFor*, qui est une autre directive. Ici elle permet de boucler sur l'ensemble des éléments du tableau *burgers* et de créer un événement de liste pour chaque et ainsi une *menu-entry* pour chaque.

restaurant-entry

restaurant-entry est fait sur la même base que *menu-entry*. Il est également possible de switcher sur le "mode éditable". La différence réside dans les informations qu'il contient.

Il intègre l'adresse du restaurant en trois variables :

- adresse ;
- code postal ;
- ville.

et des coordonnées de géolocalisation :

- lgt : longitude ;
- lat : latitude ;

qui seront passées en paramètre à l'élément de la library javascript angular2-google-maps qui me sert à afficher la carte Google Maps. Nous verrons cela plus en détail dans la partie dédiée à l'API Google Maps.

restaurant-editor

restaurant-editor reprend le même principe que *menu-editor* sauf qu'il contient le tableau des restaurants alimenté par le service *restaurants.service.ts*.

Services

Sur Angular2 les services sont des petites routines exécutées au démarrage de l'application, pour peu qu'elles soient déclarées au bon endroit, et qui permettent d'abstraire la récupération d'informations.

Côté utilisateur on récupère les informations avec toujours la même méthode, par exemple `getBurgers()`, mais côté service la technique de récupération peut différer.

Les services d'angular intègrent la notion de **promesse**, qui permet de simuler la présence d'une information qui n'est pas encore arrivée et de produire l'interface sans ces informations, qui seront mises à jour lorsqu'elles seront arrivées. Cela fait penser au fonctionnement du pattern *proxy*.

Voici le code de `menu-burger.service.ts` :

```
import {BURGERS} from "../mock-burgers";
import {Burger} from "../burger";

@Injectable()
export class MenuBurgersService {
  getBurgers(): Promise<Burger[]> {
    return Promise.resolve(BURGERS);
  }
}
```

On remarque le `@Injectable` qui caractérise les services.

Pour fonctionner, un service doit être ajouté dans les *providers* du `app.module.ts` (pour être exécuté) et injecté dans le composant qui en a besoin de la manière suivante :

```
constructor(private _burgersService: MenuBurgersService) { }

getBurgers() {
  this._burgersService.getBurgers().then(burgers => this.burgers = burgers);
}
```

Le constructeur se voit agrémenter un paramètre privé `_burgersService` (pour la convention de nommage). C'est l'injection.

la méthode `getBurgers()` abstrait la méthode `getBurgers()` du service.

La partie suivant le `.then()` est une spécificité du système de promesse. Elle indique que l'on récupèrera la donnée lorsqu'elle sera disponible.

Routing

Le routing permet la navigation entre les vues de composants. Pour fonctionner elle est composée d'un fichier `app-routing.module.ts` qui recense les "routes" qui sont des couples de paths et de composants.

Dans le template, un lien vers un path est toujours suivi des balises `<router-outlet>` :

```
<ul>
  <li><a routerLink="/burgers">Burgers</a></li>
  <li><a routerLink="/restaurants">Restaurants</a></li>
</ul>
<router-outlet></router-outlet>
```

Angular Material

Dans l'optique de réutilisation de composants externes, Angular Material, fourni par google, me permet de récupérer des éléments graphiques tout droit issus de leur charte graphique. Ce qui me permet de "styler" mon application à moindre effort (sur le papier !).

Pour utiliser les éléments de Material, il faut :

- Ajouter l'import dans l'`app.module.ts` ;
- faire `> npm install` dans un terminal à la racine du projet ;
- et de la patience parce que ça a du mal à marcher du premier coup !

La remarque négative que je peux faire suite à mon expérience personnelle, est que d'une part le site officiel de AngularMaterial2 est en beta sur Github, et d'autre part la plupart des tutoriaux le concernant décrivent un processus d'installation déprécié. Bon courage pour trouver celui qui va marcher !

Google Maps API

La tâche décrite dans l'article prévoyait que j'utilise les composants du navigateur pour récupérer la position GPS, possible notamment sur les navigateurs de smartphone. Cela est possible techniquement, cependant les seules solutions que j'ai trouvées pour faire ça étaient archaïques, la plupart du temps implémentées pour AngularJS et par manque de temps et d'idées j'ai préféré abandonner.

Je me suis reporté sur l'intégration d'éléments de l'API Google Maps dans l'interface de l'application.

Pour cela j'ai trouvé la library `angular2-google-maps` par Sebastian Müller.

Je l'ai donc intégrée de la même manière que la librairie Angular Material :

- D'abord l'import dans `app.module.ts` ;
- puis `> npm install`.

Ce qui nous permet, ensuite, d'utiliser les balises suivantes pour importer une fenêtre sur une carte de l'API Google Maps :

```
<sebm-google-map [latitude]="restaurant.lat" [longitude]="restaurant.lng">  
  <sebm-google-map-marker [latitude]="restaurant.lat"  
  [longitude]="restaurant.lng"></sebm-google-map-marker>  
</sebm-google-map>
```

On peut remarquer la latitude et la longitude de `restaurant-entry` passées en paramètres de la `map` et du `map-marker`.

Responsive

Media queries

Des média queries classiques placées dans les feuilles de styles des `restaurant-entries` et des `menu-entries` permettent de passer la largeurs des tuiles à `width=100%` de manière à conserver la lisibilité lorsque l'écran est étroit. Cela introduit le comportement responsive de l'application au niveau des feuilles de style.

```
@media screen and (max-width: 768px) {  
  width: 100%;  
}
```

Directive `ngSwitch` et `ngSwitchCase` + media queries

Côté template, on peut également faire dans le responsive, notamment en utilisant la directive `ngSwitch` couplée au média queries côté framework cette fois. En effet Angular2 offre la possibilité d'utiliser des méthodes qui implémentent l'accès au média queries, ce qui nous permet de faire des méthodes intégrable dans le template pour détecter la résolution et afficher le composant adapté.

Déploiement

Le déploiement s'effectue soit en utilisant la commande suivante dans un terminal :

```
> npm start
```

Cette commande envoie npm regarder dans le fichier package.json le script correspondant à start (ici *ng serve*).

Soit on utilise directement la commande décrite dans le .json :

```
> ng serve
```

Ces deux commandes ont pour effet de lancer l'application sur un mini-serveur atteignable par l'url : localhost:4200/.

Avantages et inconvénients

Avantages

Sur de nombreux aspects Angular2 est très intéressant. Tout d'abord on peut revenir aux points déjà décrits en introduction concernant le WebComponent. La notion de bornage des composants permet une réutilisation simple, dans le cadre de bibliothèques personnelles ou partagées. Elle nous impose une rigueur de structuration du développement là où le web est souvent considéré comme désordonné.

Angular2 se pose comme une solution intégrant une gestion du cycle de vie des variables et de nombreuses possibilités d'échanges d'informations entre les aspects du projet.

La puissance d'Angular réside aussi dans le fait de nous permettre d'utiliser les technologies standards tels que CSS3, SASS, HTML5, EcmaScript 6 etc... au sein du même projet, et ce de manière sécurisée.

On peut également intégrer des bibliothèques javascript externes pour permettre de faire ce qui n'est pas directement possible dans le framework Angular2, ce qui donne une infinité de possibilités, notamment le développement CrossPlatforms grâce à NativeScript.

Inconvénients

Cependant le développement en Angular2 pour un débutant tel que moi est semé d'embûches.

Le plus gros point noir que je puisse lui attribuer est le manque de support dû à sa jeunesse. En effet Angular2 dont la version finale n'est sortie que très récemment ne dispose que de peu de tutoriaux ou de posts sur StackOverflow, ce qui devient rapidement un handicap. Cela s'explique notamment par la rareté des développeurs qui sont passés à ce framework pour le moment. Là où les tutoriaux anglais sont nombreux, surtout pour poser les bases, les français sont inexistant.

Lorsqu'il s'agit de sortir des sentiers battus et d'utiliser des bibliothèques externes, implémentant par exemple les éléments graphiques ou l'API Google Map, cela devient un vrai parcours du combattant. Je peux par exemple citer Angular Material dans lequel Google propose les éléments graphiques Material, dont le projet est encore à l'état de bêta et le site renvoie vers le projet Github.

Je peux également citer les difficultés liées au débogage. Même si des progrès énormes ont été faits en matière de Web ces dernières années, et notamment avec Angular2, aucun ne peut se targuer de proposer un débogueur efficace comme cela existe avec les langages plus conventionnels tels que le Java, C# ou C++. Cela ne s'applique pas uniquement à moi, je suis sûr que tout le monde a rencontré le même problème. Lorsqu'un élément ne réagit pas comme on le voudrait, cela peut être extrêmement difficile de trouver une solution. Quand ce n'est pas l'ensemble de l'application qui refuse de tourner...

Conclusion

Les technologies servant à créer et implémenter des pages web avec des interfaces bien adaptés sont nombreuses. Et le choix entre elles dépend du développeur lui-même et ce qu'il veut créer à la fin.

Pure CSS est bien utile pour un développeur pour créer ce qu'il veut car les styles et les classes sont pratiquement mineurs, de même, il peut utiliser dans la majorité du temps de CSS ou de CSS3, ce qui est facile à maîtriser pour un développeur web. Il peut bien montrer sa créativité et rendre le site unique.

Par ailleurs, cela peut être considéré comme étant un inconvénient pour Pure CSS puisqu'il n'est pas assez utile ou serviable par rapport aux autres qui souhaitent utiliser des fonctions et des classes déjà existantes et obtenir des résultats plus facilement et aussi gagner du temps.

Donc, Pure garantit pour l'utilisateur le fait de personnaliser son travail pour créer l'interface avec la possibilité de modifier le code de CSS (la taille, la position, ...). Notamment, c'est pas le cas pour certains CSS frameworks. Finalement, on note que Pure n'est pas assez répandu, il n'est pas considéré comme un framework bien utilisable, dont on ne trouve pas facilement des tutoriels claires et structurés.

D'autre part, l'un des frameworks RWD les plus utilisés, Bootstrap 3 nous offre une documentation assez simple à utiliser. Un système de grille qui rend notre page web adaptable sur différentes tailles d'écrans et d'autres composants offertes par ce framework(JS ou CSS) . Cependant, ce système de grille n'est ni configurable ni paramétrable, ca sera le cas dans la prochaine version de bootstrap. Afin d'interagir et de personnaliser une page web, la technologie Bootstrap ne suffit pas, il faudra rajouter à cela un feuille de style, des media queries, du java script/Jquery et modifier des variables Less.

Concernant le frameworks Ionic, l'accent est clairement mis sur les fonctionnalités, le design proposé par les technologies RWD est bien meilleur, mais cela est compensé par les fonctionnalités proposées, telles que l'affichage des divers composants que l'on trouve sur les différentes plateformes mobiles, ListView, Button.

L'accès aux capteurs est un gros plus permettant aux développeurs de produire une application certes moins riche d'un point de vue design, mais possiblement complète en terme de fonctionnalités, et en terme d'adaptation puisque un même code génère un rendu différent en fonction du système d'exploitation (Android, IOs).

C'est ce qui la démarque des autres technologies, sa capacité à s'adapter à l'environnement en fonction du système d'exploitation et des différents capteurs.

Pour conclure avec Angular2, malgré les défauts liés à sa jeunesse, Angular2 est extrêmement puissant et propose un choix plus que large de possibilités. En plus de ses qualités technologiques on peut compter sur Google et sa puissance médiatique pour imposer sa solution comme un nouveau standard du web.

L'application d'interface d'administration quant à elle répond bien à l'adaptation de l'affichage en fonction de la résolution, mais absolument au type de device. A ce stade une comparaison avec Ionic est difficile. En effet, en l'état actuel, la manière qui semble la plus simple pour accéder à des éléments hardware des devices est d'intégrer le framework NativeScript. Ce dernier propose une réelle implémentation des APIs smartphones et non pas une intégration dans une WebView, ce qui en fait un sérieux concurrent à Phonegap.