Lucas Sauvage
Chloé Guiglielmi

Arnaud Garnier
Théo Donzelle

Compréhension et analyse des technologies

Pour notre application, nous avons choisi de créer une plateforme de partage de recettes de cuisine. Notre application intègre des fonctionnalités et pages communes : une page d'accueil, une/des pages de détail d'une recette, une page d'ajout et une fonctionnalité de recherche. Notre application nous permet donc d'intégrer des tableaux pour les ingrédients, un formulaire pour ajouter des recettes, des photos et des vidéos des différentes recettes. Nous avons également un menu pour effectuer une navigation entre ces différentes pages.

Tutoriel pour Bootstrap :

Bootstrap en quelques mots :

Bootstrap est un outil facile à utiliser pour le web responsive design. Le principe de base est simple, Bootstrap vous met à disposition une grille qui découpe votre page web en 12 colonnes. De cette façon vous pouvez facilement positionner vos éléments et viser 4 tailles de support différent : les Smartphones, les tablettes, les écrans de taille moyenne et les écrans de plus grande taille. En plus de cette fonctionnalité indispensable, Bootstrap propose tout un tas d'autres fonctionnalités : tableau, barre de navigation, formulaire, bouton, couleur et typographie, etc.

Comment installer Bootstrap :

Bootstrap est simplement constitué d'une feuille CSS, d'un fichier JavaScript et de quelques polices, il vous suffit donc de télécharger le code sur : getbootstrap.com/getting-started . Le site vous propose 3 versions différentes de Bootstrap, je vous conseille pour débuter de prendre la version compilée. Vous obtiendrez donc trois dossiers : css, fonts et js. Vous n'avez plus qu'à glisser ces dossiers dans votre répertoire comprenant vos fichiers HTML. Enfin pour utiliser Bootstrap, il vous faudra inclure le CSS et le JavaScript dans vos fichiers HTML :

- Entre les balises <head> de votre fichier le CSS :

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

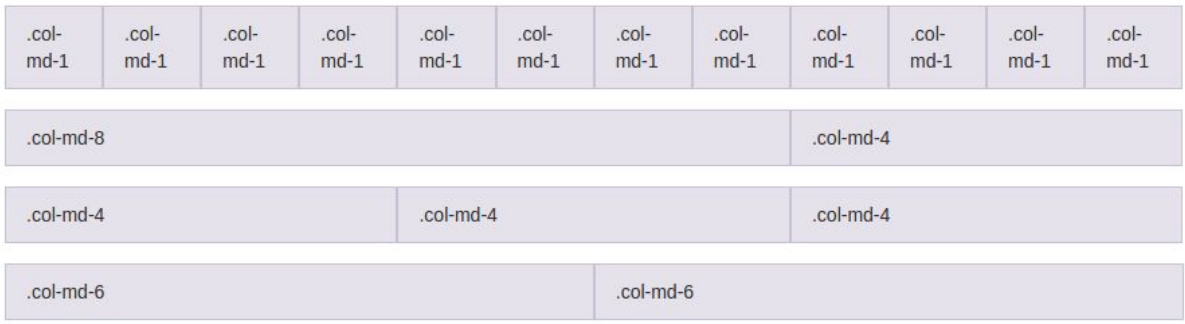
- Juste avant de fermer votre body, les fichiers JavaScript :

```
<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
<!-- Include all compiled plugins (below), or include individual files as needed -->
<script src="js/bootstrap.min.js"></script>
```

Bootstrap fonctionne donc sans outil particulier, mise à part un navigateur web et un environnement de développement. (Même si vous pouvez toujours essayer de créer votre application avec un bloc note.)

Comment utiliser Bootstrap :

Bootstrap fonctionne principalement en utilisant les attributs class des balises HTML. Par exemple pour utiliser leur système de grille, il vous faudra ajouter les attributs suivants, en fonction de ce que vous voulez faire :



Ces attributs pour class sont généralement utilisés dans des balises de façon à pouvoir les enchaîner. Par exemple pour découper votre page en deux colonnes, votre code ressemblera à :

```
<div class="row">
  <div class="col-md-6">.col-md-6</div>
  <div class="col-md-6">.col-md-6</div>
</div>
```

L'attribut "col-md-6" signifie donc que vous allez utiliser 6 colonnes sur 12 et cela pour des supports de taille comprise entre 992px et 1199px. C'est donc grâce à ce système que vous pouvez adapter votre page en fonction des différents supports. Voici un détail des différents types de colonnes :

	Extra small devices Phones (<768px)	**Small devices** Tablets (≥768px)	**Medium devices** Desktops (≥992px)	**Large devices** Desktops (≥1200px)
Grid behavior	Horizontal at all times	Collapsed to start, horizontal above breakpoints		
Container width	None (auto)	750px	970px	1170px
Class prefix	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`

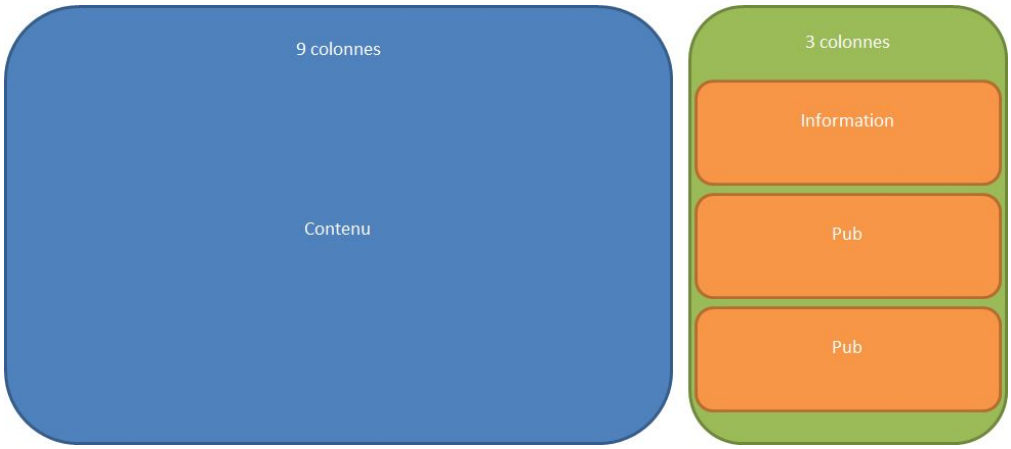
Il existe toutes sortes d'attributs que Bootstrap vous propose pour vous faciliter la vie. Toutes les fonctionnalités fournies par Bootstrap sont responsives et je vous suggère d'aller voir la documentation pour plus d'information :

- Pour le CSS : getbootstrap.com/css/
- Pour les composants : getbootstrap.com/components/
- Pour le JavaScript : getbootstrap.com/javascript/

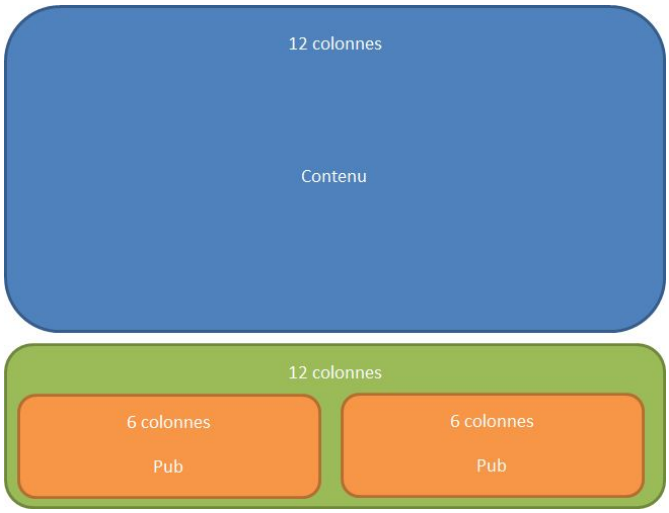
Vous pouvez également adapter facilement le CSS de Bootstrap en modifiant directement sur le site ce que vous voulez customiser : getbootstrap.com/customize/. Cela vous permet par exemple de rajouter plus de colonnes de façon à avoir un placement plus précis.

Comment j'ai réalisé mon exemple :

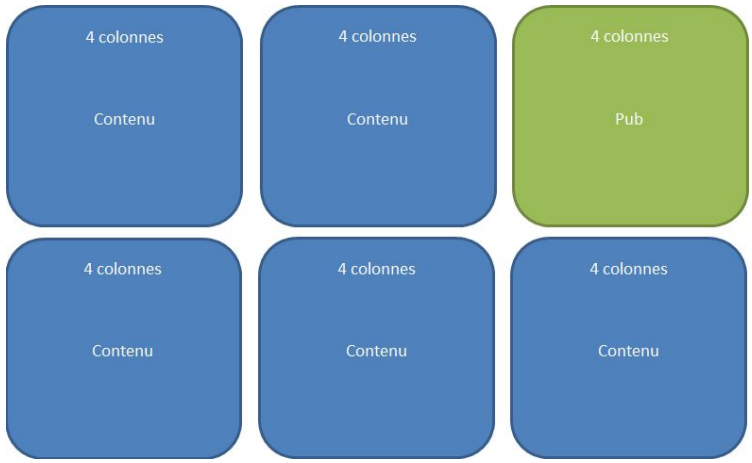
Concernant mon utilisation de Bootstrap, j'ai découpé la majorité de mes pages de cette façon : (Le "contenu" de ces pages est découpé de manières différentes en fonction de chaque page)



Ce contenu se repositionne pour des plus petits supports de cette façon :



Pour ma page d'accueil, le découpage est différent :



Ce contenu se repositionne également pour des plus petits supports. Sur des tablettes, vous aurez des blocs prenant 6 colonnes, et sur des Smartphones, des blocs prenant 12 colonnes.

Concernant mes outils de développement, j'ai utilisé comme IDE WebStorm qui me permet notamment d'envoyer mon projet sur Git afin de travailler sur plusieurs ordinateur. Pour visualiser mon travail, j'utilise le navigateur web Google Chrome qui me permet, à l'aide de l'inspecteur d'éléments, de facilement tester mon application en simulant la taille de n'importe quelle tablette ou Smartphone. Cette technique a toutefois des limites, le rendu réel sur un Smartphone ne sera pas tout à fait le même. En effet l'inspecteur d'éléments de Google Chrome ne prend pas en compte les barres de navigation des téléphones. Pour tester directement sur mon téléphone, je me suis donc servi de Browsersync qui permet de très facilement lancer un serveur en local pour consulter l'application sur n'importe quelle machine depuis un navigateur web.

Pour déployer la solution, c'est le même principe que pour tout site web. Il suffit d'envoyer votre dossier comprenant les fichiers HTML, les dossiers Bootstrap et les photos ou vidéos sur un serveur. Ainsi votre application sera déployée et accessible par tout le monde.

Concernant les capacités d'adaptation de mon application, celle-ci s'adapte à la taille des écrans. C'est donc une application responsive. Pour tester ces capacités, il vous suffit d'ouvrir le site avec un Smartphone, une tablette, un ordinateur et même une table ayant une résolution d'au moins 1920px en largeur. L'adaptation pour table n'est pas directement visible et n'a qu'un intérêt sur une page "détail" du site. En effet, certains écrans d'ordinateur ont eux aussi une résolution de 1920x1080. C'est pourquoi la navigation reste classique sur toutes les pages, mise à part la page de détail d'une recette. En effet, sur cette page vous trouverez un bouton en haut à droite vous permettant de passer en mode table.

<u>Les limites de Bootstrap :</u>

Après mon utilisation de Bootstrap, je constate quelques limites que j'ai dû corriger en ajoutant ma propre feuille CSS :

- Lorsque l'on imbrique des colonnes dans des colonnes, nous avons également une imbrication de padding qui n'est pas forcément nécessaire et nous fait perdre de la place.

Notamment lorsque cette colonne est sur un des deux bords. Il faut donc manuellement supprimer des paddings, afin d'avoir un meilleur rendu.

- L'utilisation de colonnes simplifie grandement le système mais cela peut devenir très compliqué en cas de nombreuses imbrications de colonnes.
- Bootstrap ne propose pas de différenciation de taille d'écrans au-dessus de 1200px. Vous pouvez toutefois ajouter vos propres tailles manuellement dans le code.

Tutoriel pour Pure CSS:

Pure CSS en quelques mots :

Pure CSS est un outil de responsive web design. Il s'agit de petits modules CSS que l'on peut ajouter facilement à sa page web. Tout comme Bootstrap, pure CSS se base sur un système de grilles pour disposer ses éléments. Cependant, Pure CSS utilise uniquement du CSS, et ne présente par conséquent aucune trace de Javascript ou tout autre technologie.

Comment utiliser Pure CSS :

Pour utiliser Pure CSS, il suffit d'ajouter la ligne suivante à sa page web :

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

Il est cependant possible de modifier nous-même les modules CSS présents, en téléchargeant sur le site les feuilles de styles et les les liant à notre page web.

Pure CSS utilise des modules CSS : il s'agit d'une certaine syntaxe adoptée dans la construction des feuilles de style, qui permettent une meilleure réutilisabilité de ces dernières.
Comme dit précédemment, Pure CSS se base sur les grilles pour le positionnement des éléments. Les grilles ne suivent pas exactement les mêmes règles que celles de Bootstrap, bien que leurs comportements soient similaires. Ici, une grille est définie par la classe pure-g, et les classes de grilles unitaires (placées en dessous de la classe grille) sont pure-u et pure-u-*. Les éléments fils d'un pure-g doivent nécessairement être des classes de type grilles unitaires (pure-u ou pure-u-*). Concernant la taille des grilles unitaires, on peut définir une grille unitaire qui occupe tout l'espace (pure-u, qui est équivalent à la classe pure-u-1) ou on peut choisir d'utiliser la classe pure-u-*. Cette dernière permet de définir la largeur des grilles unitaires avec des fractions. Par exemple, une grille unitaire de classe pure-u-1-5 occupera 1/5ème (soit *20%*) de la place qui lui est allouée.

Pure CSS contient de nombreuses tailles ainsi prédéfinies. Il est cependant possible d'en créer sois-même, avec un outil bas niveau : le **Pure Grids Rework Plugin**, que l'on peut obtenir sur github.

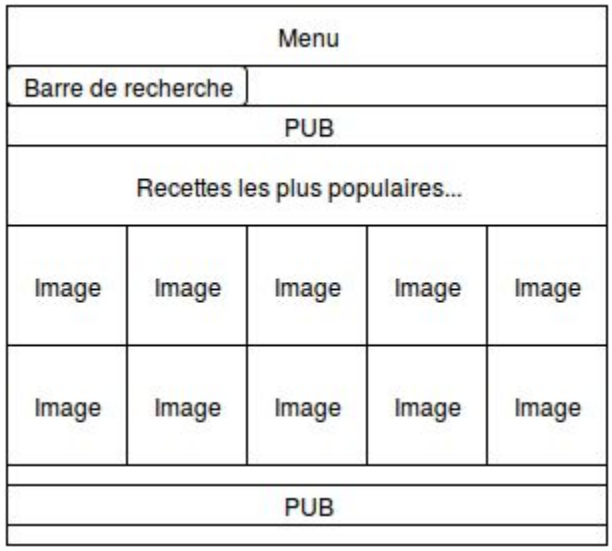
Comment j'ai réalisé l'exemple avec Pure CSS :

Dans le cadre de ce projet, j'ai téléchargé les feuilles de style CSS de Pure CSS, contenues dans un dossier *pure-release-0.6.0/* (trouvable sur le site de Pure CSS), afin de pouvoir avoir la main sur l'édition du style de la page dans son intégralité. La topologie que j'ai utilisée pour réaliser le projet est la suivante :

Concernant la construction des pages de mon application, je décris ci-après les étapes de leur construction.

La page d'accueil de l'application doit permettre 3 choses : proposer un menu de navigation (qui devra être présent et identique sur toutes les pages), une barre de recherche ainsi que les recettes les plus populaires. L'affichage des images des recettes est organisé en une grille ⅕ (c'est-à-dire qu'on a 5 images par lignes, qui se réorganiseront en fonction de la taille du device, et notamment sur les petits devices on aura une image par ligne). J'ai choisi ici de rendre la page d'accueil sobre et de n'afficher que les images des recettes. La raison de ce choix est avant tout de

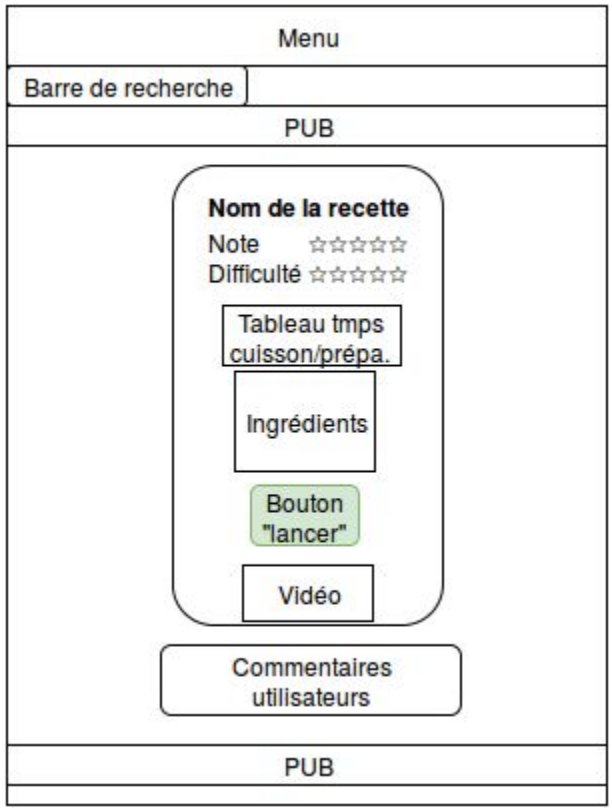
permettre à l'utilisateur de choisir une recette qu'il juge appétissante. Au survol d'une image, on ajoute un effet de zoom sur cette dernière, afin de clarifier la navigation sur ordinateur (sur support tactiles, l'intérêt sera moindre puisqu'on ne survole pas les recettes avec une souris).
La disposition de la page d'accueil, en utilisant les grilles, est la suivante :



La page permettant de consulter une recette est constitué d'une unique grille, située au centre de l'écran. Elle comporte un menu, une barre de recherche, un cadre contenant les informations sur la recette, ainsi qu'un espace "commentaires" pour les utilisateurs. L'affichage de la note des utilisateurs, de la difficulté ainsi que des temps de préparation, cuisson et du nombre de parts (ces trois derniers sont placés dans un tableau) sont situés sous le nom de la recette, tout en haut du cadre. En dessous se trouve la liste des ingrédients ainsi qu'un bouton permettant de lancer la recette, qui affichera un texte explicatif sous forme d'étapes à suivre. Enfin, on place une vidéo explicative de la recette en bas du cadre (cette dernière est optionnelle et pourra ne pas être présentes sur les différentes pages de recettes). On pourra retrouver en bas de page les commentaire des utilisateurs, ainsi que la possibilité d'en ajouter un sois-même.
Sur les petits écrans, quelques modifications seront à effectuer : l'affichage des temps de cuisson/préparation et du nombre de parts, sur les écrans trop petits, pose problème. En effet, le tableau sort du cadre de la recette et le rendu est mauvais (il faut scroller un peu pour avoir toutes les informations). Pour pallier à cela, on affichera ces informations les unes en dessous des autres (toujours dans un tableau) mais cette fois-ci , on le placera en-dessous de la liste des ingrédients. En effet, si l'on consulte l'application sur un support mobile, il se peut que l'utilisateur soit en déplacement, et notamment dans un supermarché, à la recherche des ingrédients nécessaires à la réalisation de la recette. Afin de limiter le scroll de page qui peut devenir agaçant pendant un déplacement, le fait d'afficher les ingrédients avant peut être plus pratique.

Sur de grands écrans (d'une résolution supérieure ou égale à 1920px), on pourra également noter une modification à effectuer au niveau de l'affichage des ingrédients. En effet, l'affichage actuel (un texte centré sur l'écran) n'est plus très lisible lorsque l'écran est large (on a l'impression que le texte est écrasé au centre de l'écran). Ces résolutions d'écran sont notamment celles des tables interactives, et sont donc susceptibles d'être utilisés directement en cuisine. Afin de simplifier la visualisation des ingrédients, on remplace l'affichage en forme de texte en tableau, qui occupe 80% de la largeur de la grille unitaire associée. Ainsi, la visualisation des ingrédients est plus facilement lisible sur de grands dispositifs.

Enfin, la réalisation de la recette est décomposée en étapes. On peut passer d'une étape à une autre à l'aide d'un bouton, et une barre de progression nous permet de savoir à quel niveau de la recette on se situe. Ceci permet de pouvoir suivre la recette pas à pas, sans se perdre dans un amas de texte. Pour toutes les tailles d'écrans en dessous d'une résolution de 1920px, on masque la liste des ingrédients pendant la visualisation des étapes, dans le but d'avoir un affichage plus sobre et plus facilement lisible. Pour les grands écrans, on pourra conserver la liste des ingrédients qui ne nuit pas à la lisibilité des étapes de la recette.
L'image suivante résume l'organisation de la page de consultation de recettes :



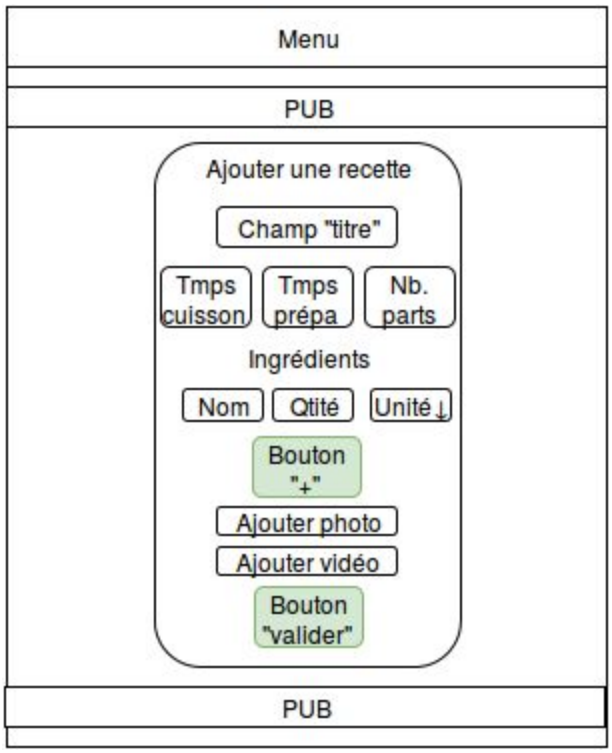
Enfin, la dernière page est celle du formulaire permettant l'ajout de nouvelle recette. Cette page comportera des publicités, un menu, ainsi bien évidemment que le formulaire. On s'abstiendra ici de mettre la barre de recherche, qui n'a pas vraiment sa place sur cette page (et qui donc surchargerait inutilement la page). Ce formulaire doit permettre d'ajouter de façon simple tous les éléments nécessaires à une recette (afin de constituer de manière automatique une page de recette complète). Ainsi, on doit pouvoir ajouter :

- Le nom de la recette ;
- Les temps de cuisson et préparation ;
- Le nombre de parts ;
- La liste des ingrédients avec leur quantité ;
- L'image de la recette ;
- Les étapes de réalisation de la recette ;
- La vidéo de la recette (champ optionnel).

Afin d'ajouter des ingrédients, on propose une ligne constituée de 3 champs : le nom de l'ingrédient, la quantité ainsi que l'unité de mesure (ml, cl, g, cuillère à café...). Un bouton (noté "+") permet de rajouter un champ supplémentaire pour un nouvel ingrédient. Un bouton similaire

est présent pour ajouter des champs de texte, permettant d'ajouter les étapes de réalisation de la recette.

Enfin, un bouton "Envoyer" est placé en bas du formulaire et permet de le valider (on affiche au passage un message à l'utilisateur pour le notifier que sa recette à bien été ajoutée). On redirige par la suite l'utilisateur directement sur la nouvelle page de recette ainsi créée.
L'organisation finale de la page doit ressembler à cela :



Pour ce qui est des outils de développements utilisés, j'ai édité mes pages html et mes feuilles de style CSS avec un éditeur de texte (Sublime Text 2). J'ai versionné mon code avec l'outil github, et j'ai testé le résultat sous deux navigateurs différents (Firefox et Chromium). J'ai utilisé la fonction de test de taille d'écrans de Chrome pour visualiser l'application sur des tailles d'écrans différentes. Pour le déploiement de l'application, cela revient à héberger un site web.

Les limites de Pure CSS:

Pendant le développement de l'exemple, j'ai pu constater de certaines limites avec l'utilisation de Pure CSS. On peut notamment citer le fait que Pure CSS ne contienne pas de Javascript (contrairement à Bootstrap). Ceci a pour conséquence de nombreuses restrictions si on s'en tient à l'utilisation stricte de Pure CSS. Par exemple dans le formulaire, afin de proposer des nouveaux champs pour ajouter des ingrédients/des étapes, le fait d'utiliser Javascript permet d'avoir un rendu intuitif (bouton "+" qui permet d'ajouter des champs), chose qui serait impossible à faire uniquement en CSS. De même pour le système d'étapes qui s'enchaînent avec un bouton "suivant". La validation du formulaire ne peut être faite en utilisant uniquement du html/css. Le passage par PHP et Javascript est nécessaire, afin de vérifier les données entrées par l'utilisateur et de les stocker (on pourrait imaginer stocker les recettes dans un fichier JSON, et générer automatiquement avec Javascript des pages CSS avec ces données).
Une autre chose à noter est la multitude de tailles d'écrans existants, qui fait qu'il est très difficile de traiter tous les cas. L'outil de test de taille d'écrans de Chrome permet de couvrir une grande partie des résolutions existantes, mais pas toutes.

Enfin, il n'est pas très intuitif d'éditer sois-même les tailles des grilles de Pure CSS avec le plugin existant, et les tailles prédéfinies peuvent dans certains cas paraîtres restrictives.

Tutoriel pour Angular 2 :

Je vais maintenant passer à Angular 2, vous présenter les principes de bases, comment mettre en place une application Web avec ce framework et finir avec une brève conclusion sur cette technologie.

Angular 2 en quelques mots :

Angular 2 est un framework qui utilise les Web Components, qui permettent aux développeurs de pouvoir créer des balises personnalisables et réutilisables. Un Web Component d'Angular 2 permet donc de configurer ses propres balises, il se compose en quatre parties :

- un fichier HTML qui déterminera la structure et le contenu de notre composant,
- un fichier CSS qui se chargera de définir la présentation, la mise en forme de notre composant,
- un fichier TypeScript* qui définira la partie fonctionnelle et les données de notre composant, pour faire le parallèle avec AngularJS il a le même rôle qu'un *Controller*,
- et enfin un deuxième fichier TypeScript qui sera le fichier de test de ce composant et qui utilise Jasmine, un framework de test pour Javascript.

On peut alors utiliser les balises définies par nos Web Components pour créer nos pages Web, rendant le code beaucoup plus modulaire et mieux découpé que si l'on utilise uniquement HTML et CSS. En effet, chaque composant a un rôle et un seul et peut être réutilisés à plusieurs endroits. Après un peu de théorie je vais maintenant passer à un peu de pratique.

*TypeScript : "TypeScript est un langage de programmation libre et open-source développé par Microsoft [...]. C'est un sur-ensemble de JavaScript c'est-à-dire que tout code JavaScript correct peut être utilisé avec TypeScript" - *Wikipédia*

Comment mettre en place un projet Angular 2 :

Démarrer un projet Angular 2, si l'on sait s'y prendre, est vraiment très rapide. Tout d'abord il faut avoir *Node.js 4 ou plus* et *npm 3* ou plus (qui est le gestionnaire de paquet de Node.js) et de lancer dans un terminal les commandes suivantes :

```
> sudo npm install -g angular-cli      # Installation de Angular-CLI
> ng new my-app-name                   # Création de notre application Angular 2
> cd my-app-name
> ng serve                             # Lancement d'un serveur web
```

J'ai donc utilisé Angular-CLI pour mettre en place mon projet Angular 2 mais aussi pour lancer un serveur web pour faire tourner mon application. Il m'a servi aussi pour générer mes différents composants.

Comment utiliser Angular 2 :

Une fois qu'on a mis en place notre projet avec Angular-CLI, on se retrouve avec la structure de fichiers suivante :

```
[my-app]
|____ e2e
|____ node_module
|____ src      # Dossier qui nous intéresse
```

```
|____ app
|____ assets
|____ environment
|____ index.html
```

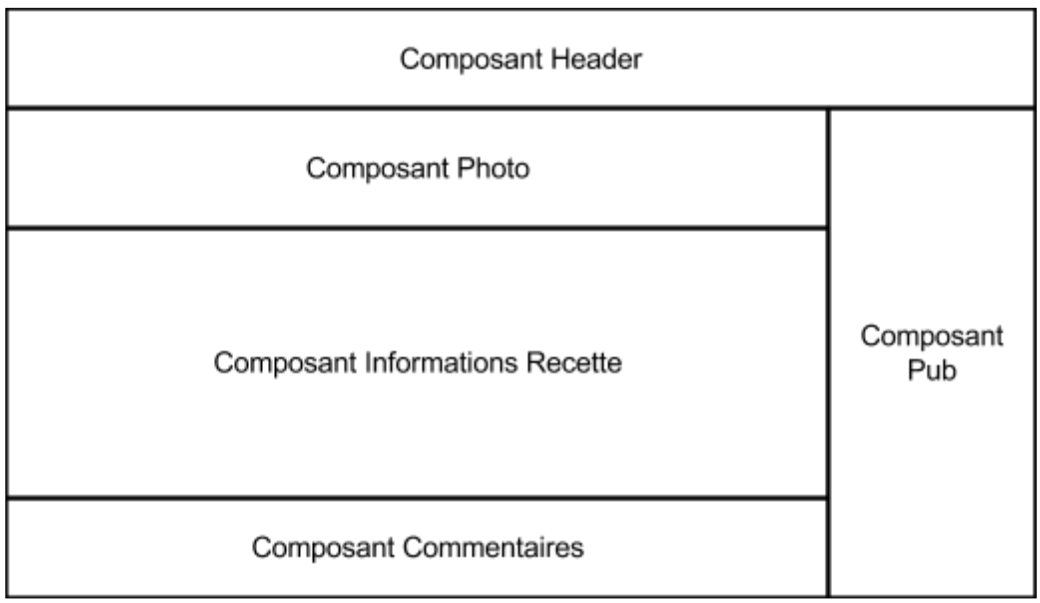
On trouvera dans le dossier “app” tous nos composants, ayant chacun un dossier à lui avec les quatre fichiers dont j’ai parlé dans la première partie. L’utilisation d’Angular-CLI permet de créer plus facilement de nouveaux composants et de garder un bonne organisation de ceux-ci.
Le fichier “index.html” est, en quelques sortes, la racine de notre application, c’est la page HTML qui contiendra (directement ou non) tous nos composants.
Pour en créer un nouveau, il suffit d’exécuter la commande suivante :

```
ng g component my-new-component
```

Pour développer, il suffit donc de créer ses différents composants puis d’organiser les différentes vues en utilisant les balises correspondants à nos composants créés.
Aussi, en complément d’Angular, il est possible d’utiliser Boostrap et ainsi bénéficier de tous les avantages de ce framework (système de grilles, éléments stylisés, etc.).

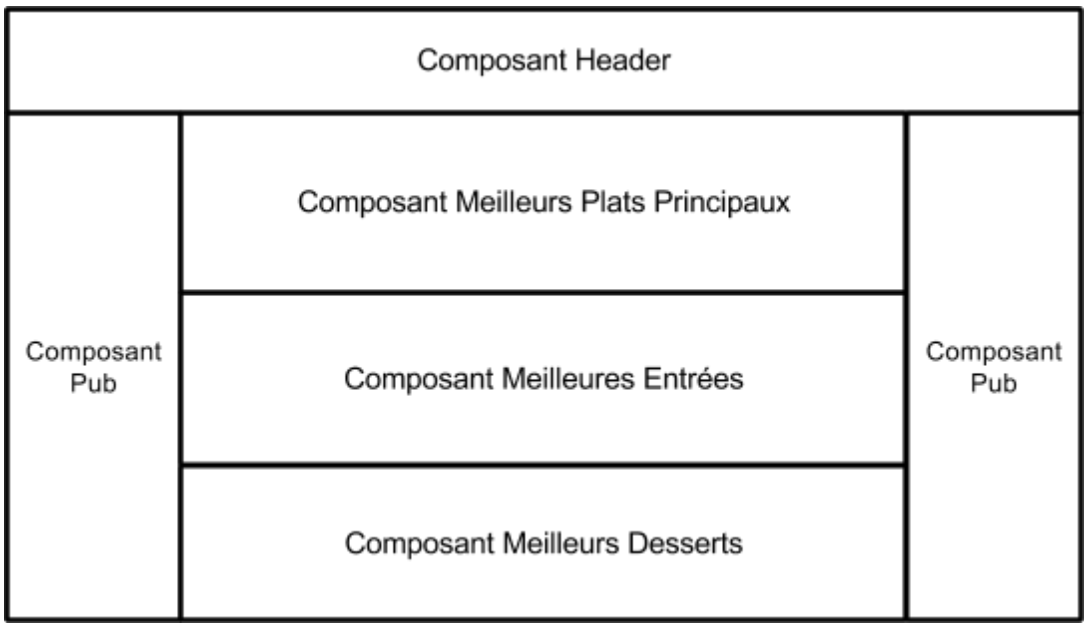
Comment j’ai réalisé mon exemple :

Pour réaliser mon exemple j’ai dans un premier temps défini brièvement quels éléments de mon application seront des composants à eux seuls (même si cela à bien bougé au fur et à mesure que je prenais en main le framework). Ce découpage devait ne pas avoir est trop gros pour ne pas se retrouver avec des composants trop conséquents, ayant trop de responsabilités (ce qui aurait limité la modalité de notre application). Mais pas non plus trop petit, car avoir beaucoup de composants peut rendre plus difficile de s’y retrouver dans le code. Il m’a donc fallu trouver la bonne échelle pour rendre l’utilisation des Web Components efficace.
Étant donné que c’est mon premier projet avec cette technologie, je n’ai peut-être pas suffisamment de recul pour savoir si mon découpage est correct mais voici le découpage que j’ai réalisé et les différents composants que j’ai créé.
Tout d’abord pour la page d’une recette :

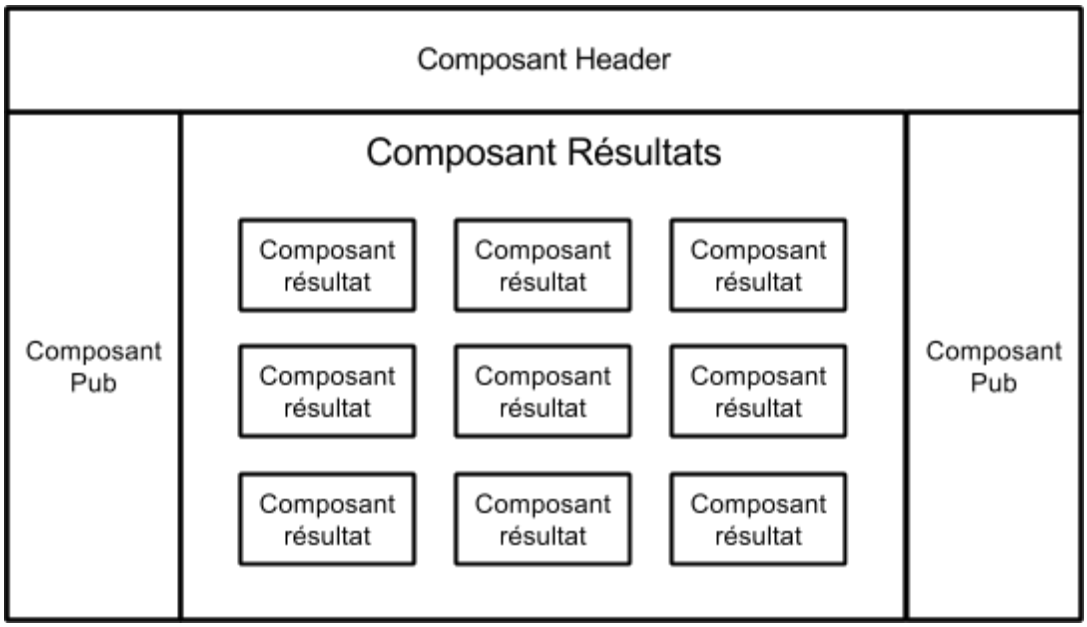


J’ai donc un premier composant qui est le *header* et qui va servir à la navigation dans mon application. Ensuite il y a le composant *photo* que j’ai placé juste en dessous de l'en tête car il

permet de capter l'attention, il permet d'illustrer la recette, il la rend plus attractive. Puis vient le composant *informations recette*, c'est l'élément central car c'est l'information principale que contient cette vue. Le composant *pub* quant à lui, a été placé sur le côté tout le long de la page, ainsi la pub est toujours visible lors de la consultation de cette page. Enfin vient le composant *commentaire* qui a été mis en pied de page, car il est le moins important. En effet, il faut déjà être un minimum intéressé par la recette pour vouloir consulter les commentaires.
On peut rappeler que pour utiliser un composant il suffit d'utiliser sa balise et non de copier tout le code qui représente ce composant. Donc si l'on veut réorganiser les composants d'une autre manière, cela devient très simple. Aussi, si l'on veut, par exemple, rajouter de la pub à gauche, il suffit de rajouter une deuxième fois la balise du composant pub et de la placer à notre guise. C'est en partie en ça que Angular 2 et les Web Components sont très intéressants.
Ensuite pour la page d'accueil voici la composition :

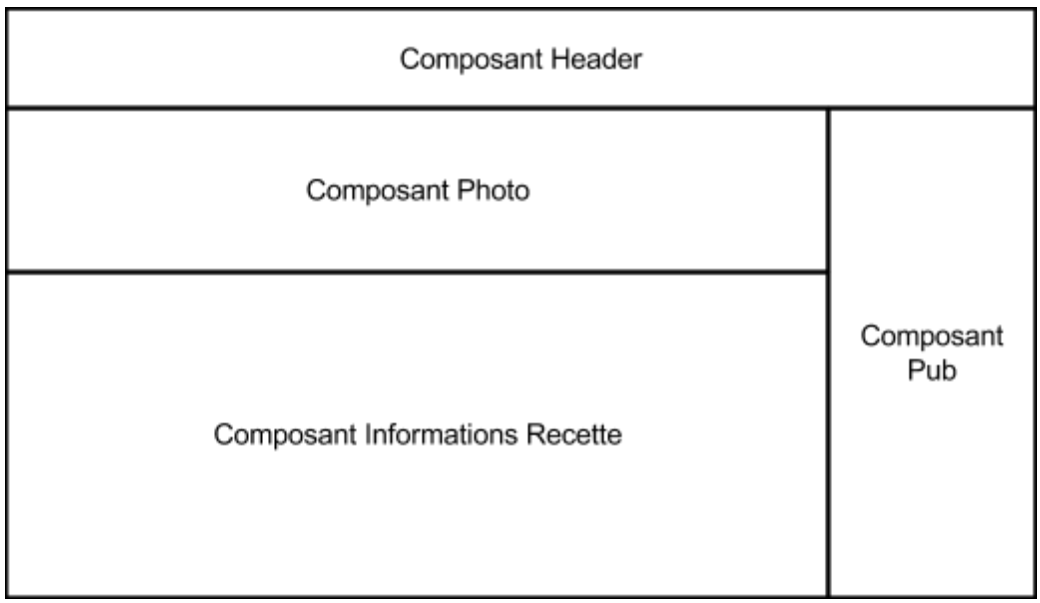


Pour cette vue j'ai préféré centraliser le contenu en l'encadrant par deux composants de publicités et simplement afficher les meilleurs recettes, réparties en trois catégories (plats principaux, entrées et desserts).
Puis vient la vue des résultats d'une recherche :



Pour cette vue j'ai repris le même positionnement des pubs que pour la page d'accueil. Et pour l'affichage des résultat j'ai opté pour une grille plutôt qu'une liste car je trouve cette dernière trop linéaire, trop standard. Sur cette vue on peut voir qu'on a la possibilité d'imbriquer nos composants (même si sur cet exemple l'utilité n'est pas très grande) et d'utiliser des composants dans nos composants.

Enfin voici la vue d'ajout de recette :



Pour cette dernière vue, j'ai repris la même présentation que pour l'affichage d'une recette, en enlevant seulement la partie commentaires. Je n'ai cependant pas réutilisé les mêmes composants que pour l'affichage. En effet, ceux pour cette vue, permettent d'éditer les différentes informations de la recette, d'ajouter des ingrédients, de mettre des photos, etc.

Les limites d'Angular 2:

Tout d'abord une des premières limites d'Angular 2 est que, pour l'instant, la communauté autour de ce framework n'est pas très grande. Donc il peut être difficile de trouver une réponse à ses questions quand on est confronté à un problème. Ceci est dû au fait que les développeurs sont

probablement encore réticent à l'utiliser car, du fait qu'elle soit nouvelle, elle est potentiellement amenée à beaucoup évoluer.
Ensuite le deuxième point que l'on peut soulever, c'est que dans un projet Angular 2 en ayant quatre fichiers par composants, on peut vite se retrouver avec un très grand nombre de fichiers. Et il peut donc est difficile de s'y retrouver si l'on ne garde pas une bonne organisation. Mais Angular-CLI corrige ce défaut et permet de garder une bonne organisation.

Conclusion Angular 2 :

En développant mon application en Angular 2, j'ai pu facilement me rendre compte de son potentiel, le découpage d'une page en Web Components permet de rendre beaucoup plus structuré le code d'une application Web. Cela corrige ce qui fait le plus défaut aux technologies du web et permet d'avoir un code propre, facilement maintenable, réutilisable et dans lequel on ne s'y perd pas. De plus, le plus gros défaut d'Angular 2, qui est son manque de communauté, devrait assez vite se résoudre par lui même.

Tutoriel pour Ionic :

Ionic en quelques mots :

Ionic est un framework permettant de développer des applications hybrides (c'est-à-dire qui exécutent une application web dans une application native). Son principal atout est qu'il permet de ne développer qu'une seule application, pour plusieurs plates-formes. On peut ainsi, avec un code source, avoir une application sur iOS et Android.
Il est basé sur les technologies du web (HTML, CSS et Javascrip via AngularJS), mais permet tout de même les accès aux capteurs que permettrait une application native.
Ionic utilise le framework Cordova, mais a l'avantage d'ajouter le look and feel des applications natives. Avec Ionic, nos applications ne ressemblent plus à de simples pages web, un utilisateur lambda ne pourra pas deviner que l'application a été codé en AngularJS.

Comment installer Ionic :

J'ai installé la version la plus récente d'Ionic (2.1.0), mais mon projet est en Ionic 1 car je maitrise mieux AngularJS 1 que 2.
Pour installer cette version sur Ubuntu, il faut au préalable avoir Node.js (https://nodejs.org/en/).
Ensuite, il suffit d'entrer en ligne de commande: "npm install -g ionic".
Une fois Ionic installé, pour pouvoir développer sur une plateforme donnée, il faut le SDK adapté. Comme j'ai développé sur Android, j'ai téléchargé le SDK pour Android (*android-sdk_r24.4.1-linux.tgz)* sur le site officiel d'Android Studio (https://developer.android.com/studio/index.html).

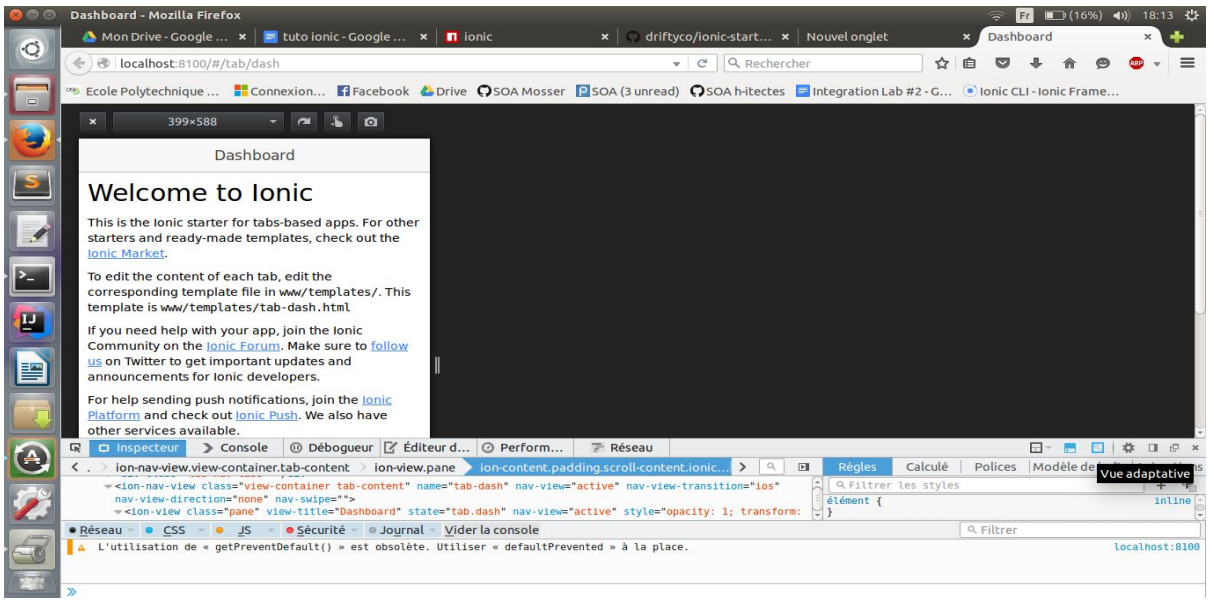
Comment utiliser Ionic :

Pour démarrer un nouveau projet, il faut entrer *ionic start myapp [template]* dans un terminal, à l'emplacement voulu pour le projet. La partie template est optionnelle, elle peut correspondre à un repository GitHub, un projet Codeopen, un répertoire local ou à un nom de projet Ionic (proposé sur GitHub : https://github.com/driftyco/). S'il n'est pas précisé, le template

sera *ionic-starter-tabs*. Pour commencer une application neutre, vous pouvez utiliser le template *blank*.
Une fois l'application téléchargée, il faut préciser la (ou les) plateforme(s) cible(s): *ionic platform android* (pour mon cas). Cette commande est à entrer dans un terminal, en étant placé dans le répertoire du projet créé.

On peut la lancer une première fois pour avoir une idée du rendu. Pour la lancer, il y a 3 options :

- La lancer directement sur smartphone : *ionic run android* (en ayant connecté un smartphone en mode développeur). Pour développer sur d'autre plateforme, il suffit de remplacer android par le nom de la plateforme.
- La lancer sur un émulateur : *ionic emulate android* , ou *ionic run android* (sans avoir connecté de smartphone).
- La lancer sur votre navigateur par défaut : ionic serve. Pour visualiser le rendu sur smartphone, les navigateurs proposent une vue adaptative. Sur Firefox, il faut faire : clique droit sur la page, "Examiner l'élément" puis cliquer sur "Vue adaptative" comme montré sur l'image ci-dessous.



Avec la dernière option, on peut modifier le code sans avoir à recompiler pour voir le résultat. Cette fonctionnalité est disponible aussi pour les deux premiers choix, mais en ajoutant l'option "-l" (pour live reload).

Pour compiler le projet, il faut faire *ionic build android* . Cette étape est optionnelle car le fait dans lancer l'application le fait déjà.

Le code à modifier pour personnaliser l'application se trouve dans le répertoire www. Comme dans un projet Angular, les fichiers sont séparés dans 3 répertoires principaux (pour ma part, je n'ai pas eu besoin de toucher au reste) :

- Les pages HTML, dans *templates*

- Les fichiers CSS pour le style des pages HTML, dans *CSS* (on peut aussi utiliser des feuilles Sass)
- Les fichiers Javascript pour le comportement des pages web, dans *js*. Il y a généralement un fichier *app.js* (pour les redirection d'une page à une autre et les fonctionnalités utilisées dans toutes les pages) et un fichier *controllers.js* (contenant les controllers pour chacune des pages de l'application, permettant de définir leur comportement).

En plus de ces dossier, la configuration générale de l'application se fait dans le fichier *manifest.json*.
Le fichier *index.html* contient les imports nécessaires au projet (link), quelques propriétés (méta) mais ne contient aucun élément destiné à être affiché (en dehors des éléments communs à toutes les pages). Elle contient une balise *<ion-nav-view></ion-nav-view>*, qui sera complétée par le contenu de chaque page.

Comment j'ai réalisé mon exemple :

Comme je savais que j'allais avoir besoin d'un menu, j'ai utilisé le template *sidemenu*. J'ai commencé par personnaliser l'application en permettant l'affichage d'une liste de recettes. J'ai décidé de stocker les informations sur les recettes dans un fichier JSON car c'est le format le plus facile à lire avec AngularJS (via Javascript). J'ai donc lu ce fichier pour stocker l'ensemble des recettes dans une variable, que je pensais parcourir dans le HTML avec un ng-repeat. Seulement l'affichage n'était pas optimal, certains éléments étaient décalés. C'est comme cela que j'ai appris qu'Ionic avait redéfini certaines directives d'AngularJS. Dans ce cas, il fallait utiliser *collection-repeat*.

Comme mon application est sur smartphone, j'ai pris en compte le fait que l'utilisateur a l'habitude d'avoir toutes les informations sans avoir à scroller. Par exemple, pour l'ajout d'une recette, au lieu de faire tout le formulaire sur une seule page, je l'ai fait sur 4 pages avec un bouton "Suivant" et un bouton "Précédent" pour naviguer d'un page à l'autre.
En effet, même si Ionic nous permet d'avoir des interfaces ressemblant à une application native, c'est à nous de faire en sorte que l'utilisateur puisse se servir de l'application de la même façon qu'une application native sur certaines points.

Ensuite, pour optimiser l'espace (souvent réduit sur smartphone) dans les pages du formulaires les plus longues (ingrédients et étapes de la recette : sous forme de listes), j'ai mis le minimum de champs possible. Pour les ingrédients par exemple, l'utilisateur à 3 champs, donc 3 ingrédients. S'il a besoin de plus, un bouton permet l'ajout dynamique de champs.

L'intérêt d'Ionic est qu'il permet l'accès aux capteurs du smartphone. J'ai donc décidé d'utiliser la caméra et la géolocalisation.

La caméra permet de prendre ou d'importer une photo lors de l'ajout d'une recette. Pour utiliser cette fonctionnalité, je me suis servie de cordova-plugin-camera (https://github.com/apache/cordova-plugin-camera). Pour ajouter le plugin, j'ai entré cordova *plugin add cordova-plugin-camera* dans un terminal placé dans le répertoire du projet. J'ai fait deux fonction, takePicture et getPicture (pour récupérer une photo depuis la Galerie). Elles utilisent toutes les deux navigator.camera.getPicture pour initialiser une variable qui sera utilisée

pour afficher l'image. Elles ne diffèrent que par la valeur sourceType des options, qui vaut 1 pour prendre une photo et 0 pour l'importer.

La géolocalisation a été un peu plus compliquée à mettre en place. Elle permet à l'utilisateur d'effectuer une recherche de recettes en fonction de sa position. S'il est près de Nice par exemple, on lui proposera des recettes Niçoises. J'ai pu récupérer la position (latitude et longitude) de l'utilisateur en utilisant cordova-plugin-geolocalisation (https://github.com/apache/cordova-plugin-geolocation). De même que pour la caméra, j'ai ajouté le plugin en faisant *plugin add cordova-plugin-geolocation.* Seulement, ce plugin ne me permettait que de récupérer les coordonnées, sans plus d'informations. Pour afficher la position de l'utilisateur, une ville est plus adaptée que des coordonnées. J'ai donc récupérer plus informations sur la position en faisant une requête HTTP à OpenStreetMaps (http://wiki.openstreetmap.org/wiki/API_v0.6). J'ai pu récupérer les nom de la ville, le pays, l'adresse... via un GET avec les coordonnées.

Les limites d'Ionic :

Comme mentionné plus haut, il faut bien se rappeler en cours de développement que même si nous utilisons des outils de développement web, nous développons une application smartpone. Sans cela, nous prenons le risque de nous retrouver avec une application qui s'utilise de la même façon qu'une page web.
Ensuite, il faut les SDK correspondant à chaque plateforme cible. C'est assez problématique lorsque l'on veut développer en iOS sans avoir de Mac par exemple.
Enfin, pour certaines fonctionnalités natives n'étant pas proposées par Cordova ou d'autres plugin, il faut écrire du code en natif. On perd alors l'intérêt du cross platform.

Conclusion :

Ces quatre expériences nous ont permis de découvrir et d'apprendre à maîtriser de nouvelles technologies. Mais aussi de réfléchir aux différents types d'interactions qu'il existe pour ensuite les mettre en œuvre. Cette mise en œuvre nous a également poussés à nous mettre à la place des l'utilisateurs pour faire en sorte que notre adaptation réponde au mieux à leur attente.

Avec ces frameworks nous avons développé la même application, avec des buts différents : réaliser du responsive web design, du cross-platform ou utiliser des web components. Ils permettent de développer rapidement une application (web ou smartphone), dont les interfaces sont toutes basées sur le principe de grilles. Ce principe est particulièrement utile pour avoir des interfaces "propres", utilisant le responsive design. Dans l'ensemble, ces frameworks nous ont offert les fonctionnalités annoncées, avec certaines limites à prendre en compte.