Adaptation des interfaces

Thomas GILLOT - Adrian PALUMBO - Arnaud ZAGO

Introduction

Matthieu, chef de rayon, souhaite faire l'inventaire des produits de son magasin. Il cherche donc une application pour l'aider à exécuter sa tâche.

Imaginons qu'il travaille dans un un petit magasin : il cherche quelque chose de très simple, qui ne nécessite rien de plus. Il aura par exemple seulement accès à un ordinateur.

En revanche, si le magasin est un peu plus grand, il devra se déplacer ; il recherchera donc un équipement plus spécifique tel qu'un téléphone, et cherchera à gagner du temps *(commande vocale, reconnaissance de code barre, etc...)*.

Enfin, si c'est une grande surface, il aura à sa disposition un ordinateur portatif fixé au poignet.

Le responsable de Matthieu souhaite pouvoir voir l'inventaire en temps réel. Cette fonctionnalité est notamment utilisée sur un grand écran surnommé "dashboard" dans la salle des équipes, pour voir quels sont les rayons ou l'inventaire n'est pas encore fait.

Au cours de ce rapport, nous décrirons l'utilisation de différentes technologies pour répondre à cette problématique. L'idée derrière cela est de voir les limites/les avantages/les inconvénients de chacunes de ces technologies.

Bootstrap 4 *(Thomas GILLOT)*

Réalisation de l'exemple

Lors du développement de l'exemple, à l'aide de la librairie graphique Bootstrap v4, je n'ai pas eu besoin de beaucoup me documenter au préalable car j'ai déjà eu à développer des applications WEB en utilisant cette librairie, durant mes études ou encore durant mon stage de quatrième année. En revanche, j'ai du réfléchir comment développer un exemple qui démontre toutes les limites de cette librairie mais aussi ses nombreux avantages. Le véritable défi du développement de cet exemple n'était donc pas l'apprentissage de la technologie, comme pour une autre technologie du groupe, mais de pouvoir me servir de mon background pour pousser cette technologie jusqu'à ses limites.
Cet exemple, se présente donc sous la forme de deux pages HTML statiques utilisant la librairie Bootstrap.

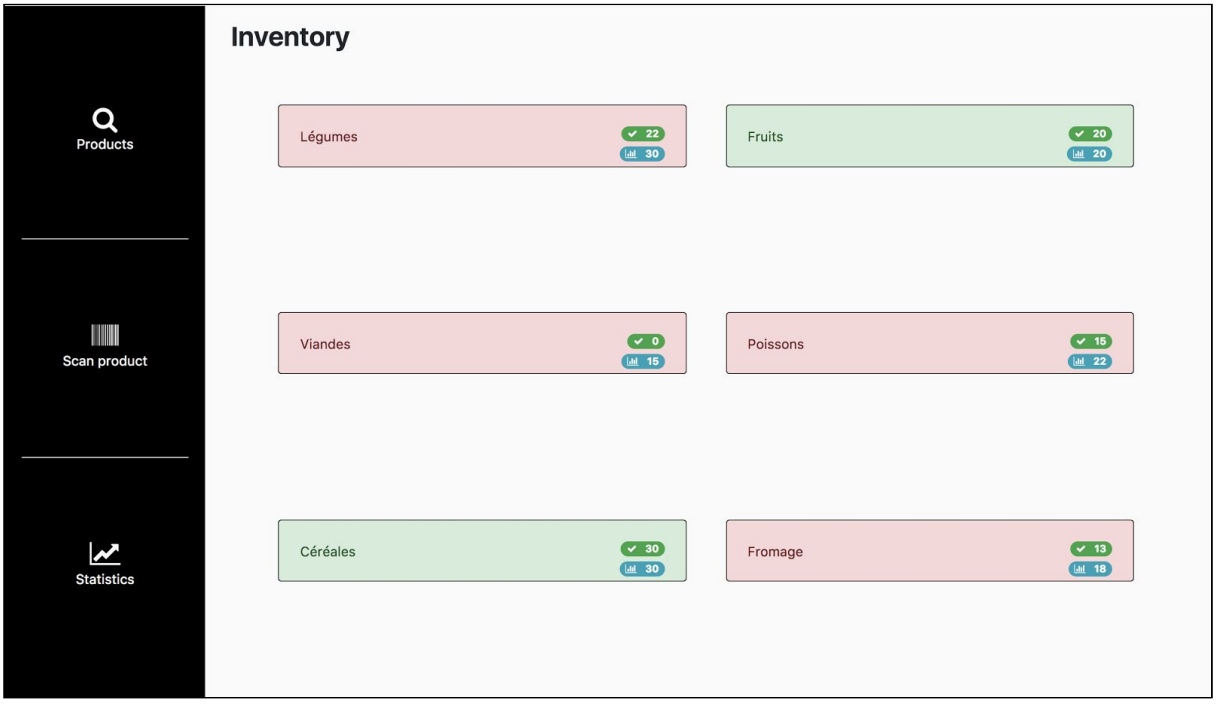


Figure 1: Page principale de l'exemple (Adaptation: Grand écran)

```
<nav class="sidebar d-none d-md-block col-md-2 col-xl-2 navbar-inverse ">

    <ul class="nav nav-pills flex-column">

        <li class="list-item">
            <span onclick="toAisles()" class="sidebar_link"><i class="fa fa-search fa-2x" aria-hidden="true"></i><br>Products</span>
        </li>

        <li class="list-item">
            <span class="sidebar_link"><i class="fa fa-barcode fa-2x" aria-hidden="true"></i><br>Scan product</span>
        </li>

        <li class="list-item">
            <span class="sidebar_link"><i class="fa fa-line-chart fa-2x" aria-hidden="true"></i><br>Statistics</span>
        </li>


    </ul>

</nav>
```

Figure 2 : Exemple du code de la sidebar - Uniquement présente sur les moyens à grands écrans

Pour la liste des différents rayons (figure 1) nous pouvons voir que sur grands écrans, existe une sidebar permettant de naviguer entre les différentes fonctionnalités de l'application (par soucis de temps et de qualité du travail rendu, seule la partie "Products" a été développée). Cette navbar ne rendant pas forcément très bien sur petits écrans, elle a été remplacée par une navbar habituelle sur petits écrans. Nous voyons sur la figure 2, que les classes "d-none d-md-block" du tag "nav" permettent d'afficher cette sidebar, uniquement à partir des écrans de taille moyenne (d-none étant établie pour les très petits et petits écrans). Cet exemple d'adaptation est un exemple d'une des forces de Bootstrap qui est la facilité d'adapter notre application aux différentes tailles d'écrans très facilement. Voilà donc un exemple de réalisation de l'exemple, nous verrons d'autres adaptations dans le paragraphes liés aux capacités d'adaptation

Outils de développement et de test

En revanche, afin de rendre la page plus personnalisée et dynamique, des petits ajouts de CSS purs et de JavaScript vanilla couplé à l'outil JQuery ont été nécessaires. Ces petits ajouts, montrent une des limites de la librairie Bootstrap pour le développement de front-end. En effet, Bootstrap ne propose pas d'outils pour la mise en place de certaines fonctionnalités présentes sur le CSS purs (telles que les transitions, le style sur certaines propriétés comme hover ou encore focus). Donc certains outils de développement ont donc été nécessaires comme JQuery, mais l'ajout de cet outil n'a pas été trop gênant car il est indispensable au fonctionnement de certains composants de Bootstrap. Cela a donc été un outil totalement transparent à ajouter.

```
openModal = function(row){
    var table = document.getElementById('products');

    var rowIndex = (row.rowIndex === 1)? 0 : row.rowIndex - 1;
    console.log(table.children[1].children[row.rowIndex]);
    $('#name_modal').text(table.children[1].children[rowIndex].children[2].innerText);

    $('#id_modal').text("Id: "+ table.children[1].children[rowIndex].children[0].innerText);
    $('#img_modal').html(table.children[1].children[rowIndex].children[1].innerHTML);
    $('#qté_modal').text("Qté: "+ table.children[1].children[rowIndex].children[3].innerText);
    $('#inv_modal').html("<span> Inventaire effectué:  </span>" + table.children[1].children[rowIndex].children[4].innerHTML);


    $('#product-detail').modal('toggle');


};
```

Figure 3: Exemple de fonction JavaScript utilisé - Ouverture du détail de l'article

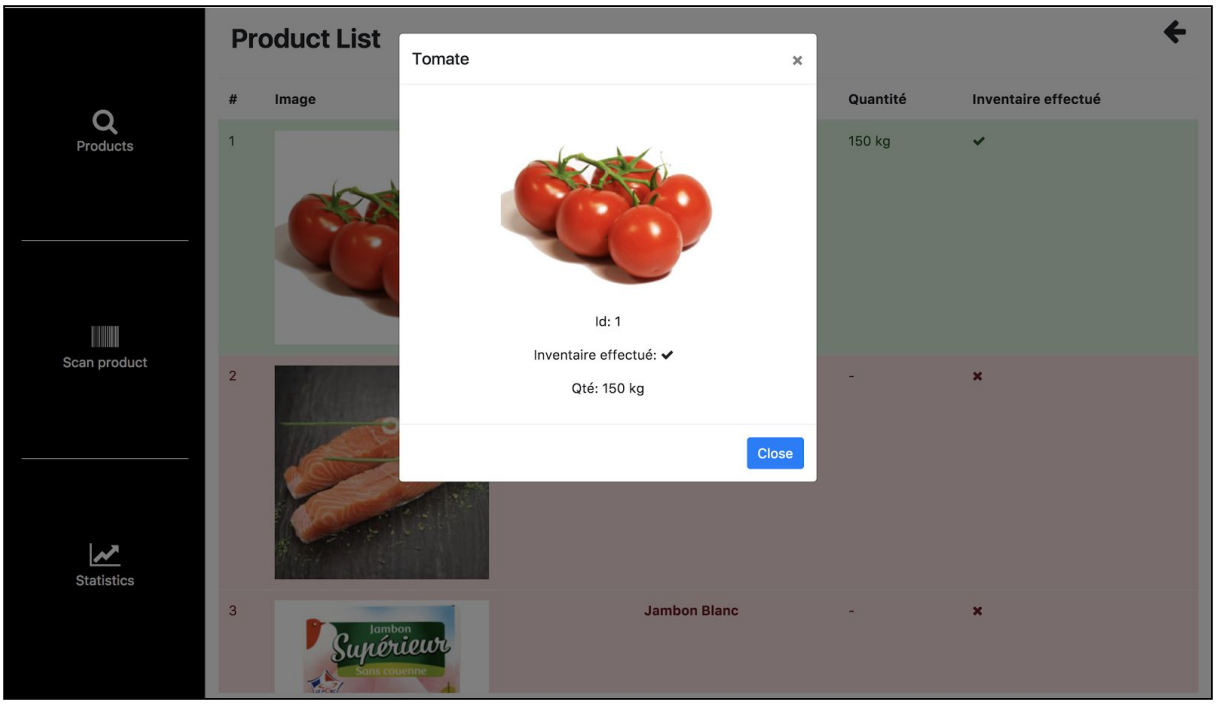


Figure 4: Ouverture du détail d'un des articles

Sur les figures ci-dessus, nous pouvons voir une des principales fonctions, écrite en JavaScript à l'aide l'outil JQuery, que j'utilise pour l'affichage dynamique d'une modal donnant le détail d'un article. Cette fonction permet lors d'un clic sur un article de la liste, d'afficher ses informations sur la fenêtre modale. Nous pouvons voir que pour l'ouverture de ce type de modal ("*modal fade*"), l'utilisation de la ligne "*$('product-detail).modal('toggle')*" est obligatoire, ce qui montre bien que l'utilisation de certains outils fournis par JQuery sont obligatoires pour utiliser des composants de Bootstrap.

Etant donné que l'exemple se présente sous la forme de page HTML statiques, les outils de tests complexes n'ont pas été nécessaires. En effet, je me suis servi des outils d'adaptation fournis par la plupart des navigateurs (j'ai testé la page sur Google Chrome, Mozilla Firefox et Safari). Pour ce qui est du test des deux fonctions JavaScript écrites à l'aide de JQuery, je n'ai pas jugé bon de mettre en place des tests, la logique du code étant très simple et ce dernier très court.
Pour l'environnement de développement j'ai utilisé WebStorm qui est un IDE qui est à mon goût parfait pour le développement WEB.

Déploiement de l'exemple et exécution

L'exécution de cet exemple est vraiment très simple. En effet ce sont des pages HTML statiques. Il suffit de récupérer le code à l'adresse suivante: *https://github.com/Rouletus/adaptation*.
Ensuite il suffit d'aller dans le dossier contenant le code, et ouvrir la page "index.html" à l'aide de votre navigateur préféré, et vous pouvez exécuter cet exemple.

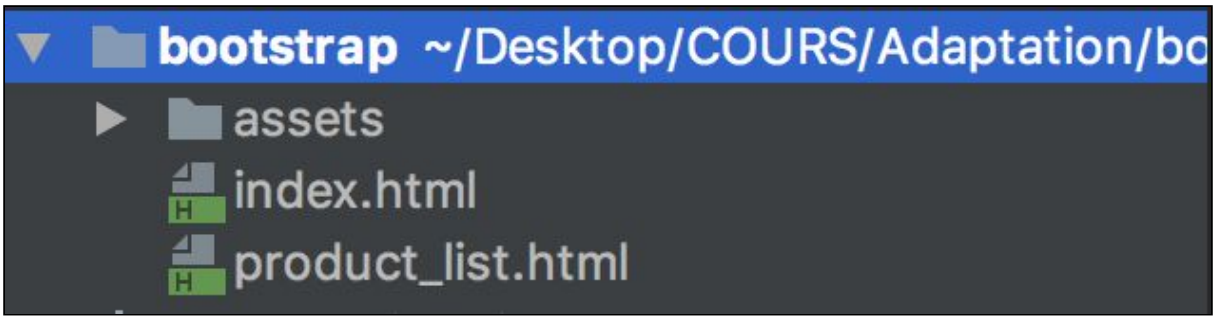


Figure 5: Arborescence de fichiers

Vous n'aurez pas d'autres actions à effectuer ou librairies à installer, puisque tout ce dont la page statique a besoin se trouve dans le dossier "assets" (y compris Bootstrap et JQuery).
Pour le déploiement, sur un serveur distant par exemple, il suffirait de mettre ces fichiers statiques sur un serveur (Apache par exemple). Pour accéder à l'exemple il suffirait d'aller au chemin contenant le fichier "index.html" et le serveur se chargera d'afficher la page.

Tests des capacités d'adaptation

Pour tester les capacités d'adaptation j'ai testé suivant trois scénarios :

- l'utilisateur est dans les rayons, il doit donc utiliser un pocket PC fixé à son poignet pour effectuer son inventaires : adaptation aux petits voire très petits écrans.
- l'utilisateur utilise un écran de taille moyenne (tablettes, ordinateur 11" ...) pour effectuer son inventaire : adaptation aux écrans de taille moyenne
- l'utilisateur veut afficher son inventaire sur un écran de type dashboard pour pouvoir voir l'état de l'inventaire courant : adaptation aux grands voire très grands écrans

Pour me permettre d'effectuer ces différents scénarios, j'ai utilisé les outils développeurs fournis par mon navigateur préféré (i-e: Chrome).

L'adaptation sur tous les tailles de dispositifs, est un des points forts de Bootstrap. En effet très simplement à l'aide de mots-clés à insérer sur toutes les classes Bootstrap (telles que "col" pour le placement sur la grille, ou encore "m*, l*" pour appliquer une marge sur les composants. Ces mots-clés définissant l'adaptation aux différentes tailles d'écrans sont :

- "" pour un très petit écran (ex: "ml-2" applique une marge gauche de 2 à partir des très petits écrans, c'est-à-dire dont la largeur ne dépasse pas 565 px)
- "sm" pour les petits écrans (ex: "ml-sm-2" applique une marge gauche de 2 à partir des petits écrans, c'est-à-dire dont la largeur ne dépasse pas 767 px)
- "md" pour les moyens écrans (ex: "ml-md-2" applique une marge gauche de 2 à partir des moyens écrans , c'est-à-dire dont la largeur ne dépasse pas 991 px)
- "lg" pour les écrans larges (ex: "ml-lg-2" applique une marge gauche de 2 à partir des écrans larges, c'est-à-dire dont la largeur ne dépasse pas 1199 px)

- “xl” pour les écrans larges (ex: “ml-xl-2” applique une marge gauche de 2 à partir des très grands écrans, c’est-à-dire dont la largeur dépasse `1200` px)

Nous venons de voir que les très grands écrans, sont considérés comme des écrans de largeur supérieure à 1200px. C’est une des limites de Bootstrap car ce mot-clé là, ne permet pas un affichage sur des dashboards par exemple. On peut donc se retrouver avec des ordinateurs (portable ou moniteurs externes) et des dashboards qui auront la même adaptation. Pour avoir une adaptation sur des écrans plus grands encore, il faudrait ajouter des media-queries CSS3 pour ces derniers.

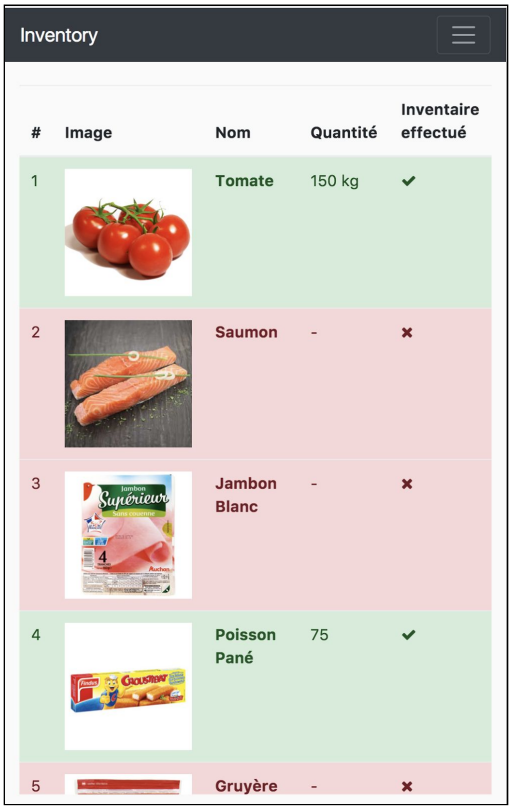


Figure 6: Vue de la liste de produit sur un petit écran

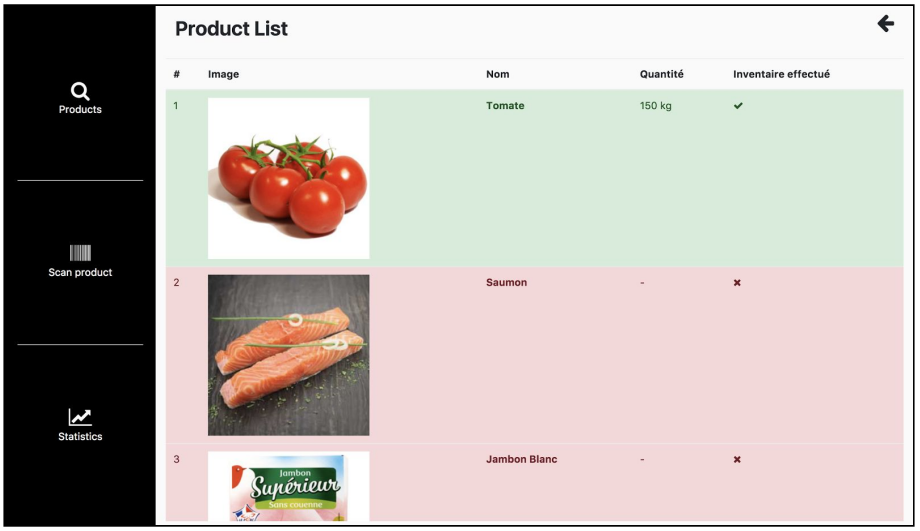


Figure 7: Vue de la liste de produits sur un grand écran.

De plus, Bootstrap met à disposition des composants “prêts à l’emploi”, qui permettent une adaptation encore plus simplifiée. En effet l’adaptation aux différents types d’écrans est déjà faite pour vous. Il vous suffit juste de vous en servir en les remplissant à votre guise. Malheureusement, ces composants sont assez bruts au niveau design. Il vous faudra les customiser afin que votre application ne ressemble pas à n’importe quelle application bootstrap brute. Dans cet exemple j’ai utilisé les composants navbar, et le composant modal (figures 4, 6, 8).

```
<div id="product-detail" class="modal fade">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 id="name_modal" class="modal-title">Tomate</h5>
                <button type="button" class="close" data-dismiss="modal" aria-label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                <div class="text-center">
                    <span id="img_modal"><img  src="assets/img/tomate.jpg" class="rounded" ></span>

                    <p id="id_modal">Id: 1</p>
                    <p id="inv_modal">Inventaire effectué: <i class="fa fa-check" aria-hidden="true"></i></p>
                    <p id="qté_modal">Qté: 150kg</p>
                </div>
            </div>
            <div class="modal-footer">
                <button type="button" data-dismiss="modal" class="btn btn-primary">Close</button>
            </div>
        </div>
    </div>
</div>
```

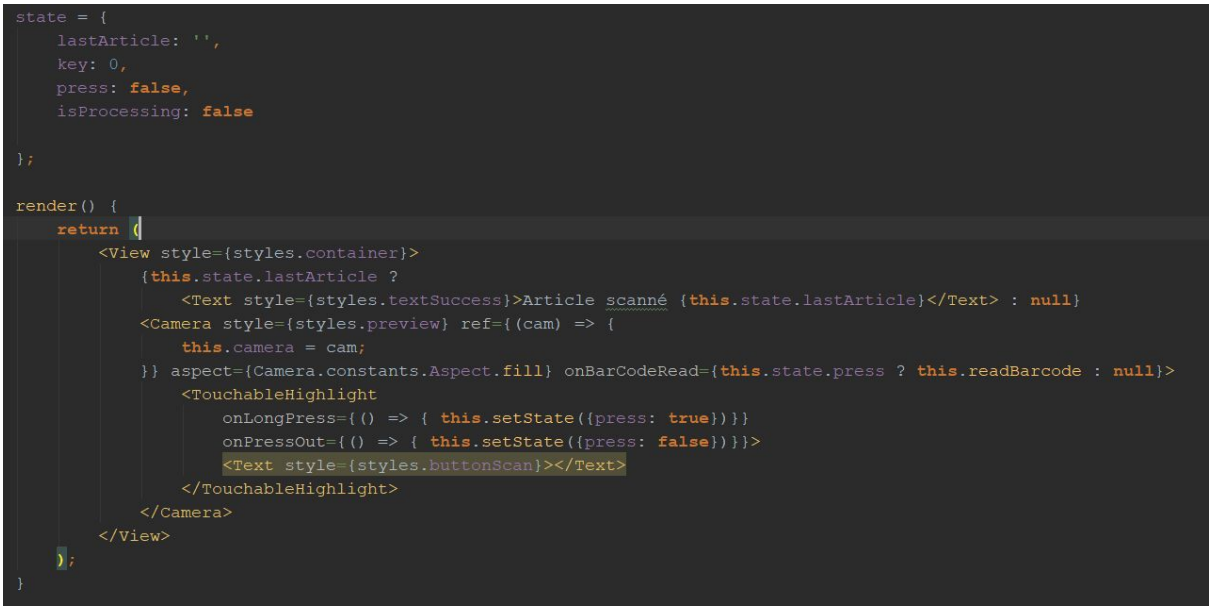
Figure 8: Vue du code de la modal du détail d’un produit

Synthèse (Avantages / Inconvénients)

- Avantages
    - Grille très optimisée et très stable permettant de placer les éléments de façon simplifiée en tenant compte de la taille de l’écran
    - Composants dont l’adaptation est déjà faite
    - Bon panel de tailles d’écrans disponible
    - Prise en charge de nombreuses règles CSS de manière simple et rapide
    - Prise en main rapide
- Inconvénients
    - Répétition de beaucoup de parties du code pour pouvoir faire des adaptations (sidebar / navbar ont du code en commun)
    - Peu d’interactions utilisateurs disponibles (hover ou encore focus doivent être faites à la main en CSS)
    - Obligation d’importer la librairie JQuery pour pouvoir faire fonctionner certains composants
    - Taille des très très grands écrans non prise en compte (ex: Dashboards)

React Native (Arnaud ZAGO)

Pour réaliser ce projet en React Native j'ai dû commencer par me plonger dans la documentation sachant que je n'en avais jamais fait. De plus ayant un background Angular la transition avec React était d'autant plus compliqué, car la gymnastique n'est pas du tout la même. Effectivement le principe de React Native repose sur celui de ReactJs. C'est à dire un composant comporte un état et des propriétés. Ce qui n'est pas exactement la même chose que de développer avec une logique totalement objet présente dans Angular.



```
state = {
    lastArticle: '',
    key: 0,
    press: false,
    isProcessing: false

};

render() {
    return (
        <View style={styles.container}>
            {this.state.lastArticle ?
                <Text style={styles.textSuccess}>Article scanné {this.state.lastArticle}</Text> : null}
            <Camera style={styles.preview} ref={(cam) => {
                this.camera = cam;
            }} aspect={Camera.constants.Aspect.fill} onBarCodeRead={this.state.press ? this.readBarcode : null}>
                <TouchableHighlight
                    onLongPress={() => { this.setState({press: true})}}
                    onPressOut={() => { this.setState({press: false})}}>
                    <Text style={styles.buttonScan}></Text>
                </TouchableHighlight>
            </Camera>
        </View>
    );
}
```

Capture d'écran d'un bout de code d'un composant de mon projet

Effectivement dans cette capture d'écran nous observons que le composant dispose d'un objet state qui correspond à son état. Puis nous pouvons observer que dans la fonction render, qui permet de définir de manière visuel le composant, chaque sous composant dispose de différentes propriétés tel que 'style, aspect…' ou autres.
Afin de ne pas s'étendre trop longuement sur le sujet nous pouvons résumer que les propriétés permettent de modifier le composant de manière instantané en lui passant des variables en quelque sorte. Et l'état permet de définir ce qui doit être affiché ou non. Effectivement lorsque que nous exécutons la fonction `setState()` celle ci permet alors de modifier l'état du composant. Et une fois l'état modifier React se charge de recharger les bouts de vues impactés.

Lors du développement de la solution je me suis tout d'abord tourné vers un téléphone virtualisé pour tester l'application. Dans le but de pouvoir développer n'importe où et n'importe quand la solution. Effectivement nous pouvons build l'application en branchant le téléphone comme n'importe quel système de création d'application. React Native délivre effectivement un build natif sur l'appareil ciblé. Puis pour développer dans

un environnement facilement testable, le système déploie un serveur à l'adresse http://localhost:8081 de la machine de dev. Ce serveur permet le rechargement de l'application sans avoir à tout rebuild et donc d'appliquer les modifications effectuées sur les fichiers JS en une fraction de seconde. Par contre si l'on souhaite rajouter un plugin ou autre il faut rebuild entièrement l'application.
Puis en arrivant sur la dernière semaine de développement j'ai découvert dans la documentation une manière très efficace et simple pour tester sur un device physique en liant le serveur de production au téléphone par le biais du Wi-Fi. Et donc pouvoir avoir son téléphone libre de tout mouvement, tout en développant en même temps.
J'ai donc pu utiliser un téléphone physique afin de valider que tout fonctionnait bien.

Maintenant je vais vous expliquer comment exécuter la solution. Tout d'abord installez react-native (nous supposons que vous avez node et npm à jour sinon rdv ici https://nodejs.org/en/), le plus simple étant de se rendre sur cette page et de suivre scrupuleusement chaque étape en fonction de votre configuration https://facebook.github.io/react-native/docs/getting-started.html. Sachant que la solution que j'ai développé est configuré pour android (n'ayant pas de Mac je n'ai pas pu faire les config pour IOS).
Dans les grandes lignes la page précédente vous demandera d'avoir un sdk à jour ainsi qu'un jdk avec des variables d'environnements bien formées puis de lancer dans un terminal `npm install -g react-native-cli`.
Ayant eu de nombreux problèmes en utilisant npm pour installer les différentes dépendances (problème que je ne m'explique toujours pas), il semblerait que tout fonctionne mieux avec Yarn. Alors partez tout de suite installer Yarn avant de continuer https://yarnpkg.com/lang/en/.
Maintenant il suffit de récupérer le code sur GitHub à l'adresse suivante : https://github.com/zarnifoulette/IHM_Adaptation.git.
Une fois tout ceci effectué nous pouvons alors brancher un appareil (virtuel ou non) android à la machine puis exécuter la commande `react-native run-android`. Cette commande permet de build le projet pour la cible Android et de l'envoyer automatiquement à l'appareil Android connecté à la machine. 1, 2.....3 tada l'application vient d'apparaître sur votre appareil et vous n'avez plus qu'à la tester !
Petites précisions si vous voulez apporter des modifications et rebuild l'application il est préférable de supprimer le dossier `android/app/build` et le dossier `android/build`. Ce qui vous évitera de perdre quelques heures de recherche sur Google.
Pour tester les capacités d'adaptations il faut se mettre dans la peau d'un employé voulant faire l'inventaire dans un magasin et jouer sur les 3 axes de l'adaptation. A savoir l'environnement, l'utilisateur et le dispositif. Pour une meilleur compréhension des captures d'écran de l'application sont sur la page suivante.

Établissons quelques minis scénarios :

- **Environnement** :

Autour de vous l'environnement est hostile et ne dispose pas assez de lumière, vous ne pouvez donc pas scanner le code barre du produit. Pas de problème l'application s'adapte à votre besoin car vous pouvez utiliser la voix pour faire votre inventaire.

Imaginons maintenant qu'il y ait trop de bruit autour de vous pour utiliser la voix, la lumière n'étant toujours pas au rendez vous, vous pouvez alors saisir manuellement le code barre et la quantité des articles comportant le même code barre.

- **Utilisateur** :

Vous devez déplacer des cartons d'articles et vous ne voulez pas sortir un à un les articles ?! Vous pouvez alors activer le mode vocal afin d'avoir les mains libres. Ou saisir manuellement les informations une fois le carton déposé.

Si vous avez besoin d'aller rapidement vous pouvez utiliser le scan afin de rapidement passer d'un article à un autre.

Vous préférez faire un peu de calcul mental et compter tous les même articles d'un coup puis saisir le nombre total dans l'application ?! C'est possible aussi avec la saisie manuelle.

- **Dispositif** :

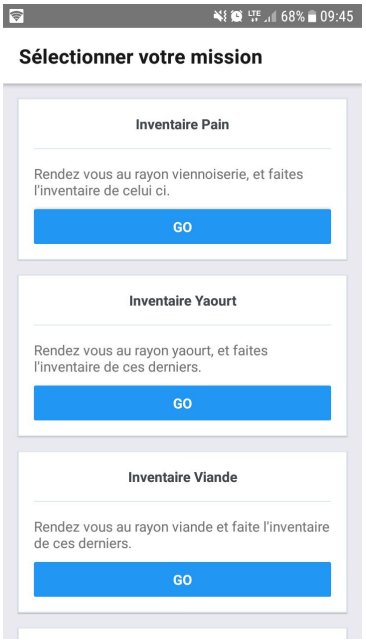
Si vous êtes sur un smartphone classique tout ira bien, sachant que l'application est faite pour ce type d'appareil.

En revanche si vous êtes sur tablette vous aurez un problème lors du scan car il faut rester appuyé sur un bouton pour le déclencher, afin de reprendre le principe des douchettes de caisse. Or sur tablette il est difficile de la tenir à une main et d'appuyer en même temps sur le bouton. L'adaptation n'est donc pas optimal. Il faudrait penser à supprimer ce bouton quand on passe sur un grand appareil par exemple.

Enfin si votre appareil ne dispose pas des capteurs nécessaires l'adaptation sera très réduite. Sans appareil photo pas de scan et sans micro pas de reconnaissance vocal. Il ne vous restera alors plus que la saisit manuel. Ce qui n'est pas toujours pas meilleur solutions vu les scénarios précédents.

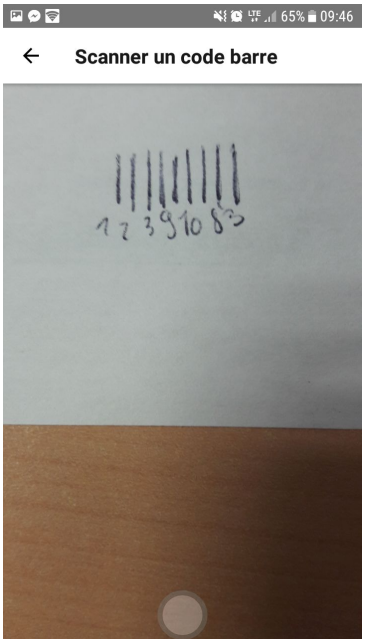
L'idée ici était de pousser au maximum les 3 axes de l'adaptation afin de voir comment l'on pouvait gérer la plasticité. Et comment l'adaptation en général ce comporte.

Sélection de la mission souhaitée

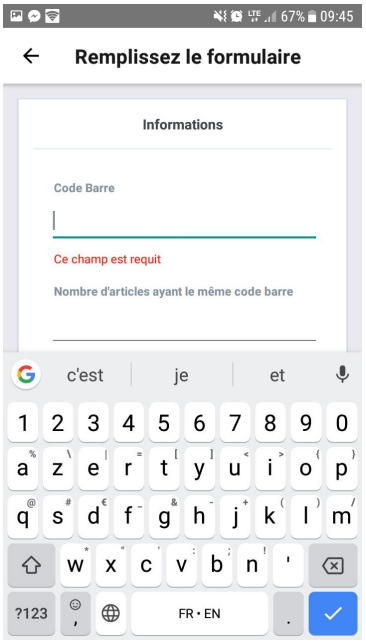


Sélection de l'outil souhaité

Outil scan code barre

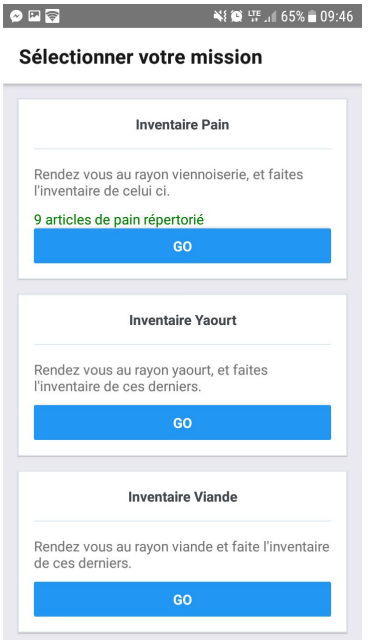


Outil formulaire



Outil reconnaissance vocal

Résultat sur la page mission



Angular (Adrian PALUMBO)

Explications

Premièrement, qu'est-ce qu'Angular ? Google, derrière Angular, nous indique que c'est "*une seule façon de développer des applications avec Angular*" et qu'on peut "*réutiliser son code et ses compétences pour construire des applications pour n'importe quelle cible. Pour le web, le web mobile, le mobile natif et pour le bureau natif.*"

Partant du postulat de Google, j'ai alors choisi cette technologie pour illustrer notre exemple. Ce dernier est plutôt simple à implémenter je dois dire : en deux commandes, on peut avoir un projet qui fonctionne *(à condition d'avoir npm sur sa machine)* !

npm install -g @angular/cli && ng new my-app

Angular utilise des **components** pour fonctionner. Ce sont des "briques" qu'on peut réutiliser à volonté *(d'où l'idée de "réutiliser son code" prônée par Google)*. Ces derniers fonctionnent avec toujours quatre parties *(donc possiblement quatre fichiers)*, qui utilisent Typescript et de l'HTML/CSS :

- La vue en HTML ;
- Le style en CSS ;
- Un fichier Typescript ;
- Pourquoi pas un fichier de tests.

Exemple de component

Outils & Déploiement

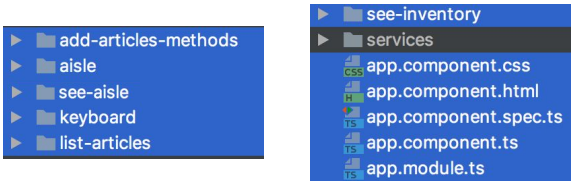
Pour lancer notre projet, rien de plus simple : une seule commande suffit, **ng serve**. Cette dernière lance un serveur local sur le port 4200, par défaut. Il suffit donc d'ouvrir son navigateur préféré et d'aller à l'adresse **http://localhost:4200** pour voir l'application tourner. Le gros avantage est que Angular détecte automatiquement les changements effectués dans le code, et recharge l'adresse.

Comme il s'agit de simples fichiers HTML, CSS et Typescript, n'importe quel éditeur fera l'affaire. Cependant, pour faciliter la vie du développeur, il existe de nombreux outils et plugins qui feront l'auto-complétion, etc. Personnellement, j'ai utilisé ***WebStorm*** qui comprend parfaitement Angular et les méthodes qu'il y a derrière.

L'application

Pour styliser l'application, j'ai opté pour Bootstrap, qui est simplement une bibliothèque CSS facilement intégrable à Angular. J'ai ainsi pu adopter tout un tas de style très rapidement.

Pour respecter les principes d'Angular, j'ai donc découpé l'application en **components** ; on en compte ainsi sept.



Certains components de l'application
En bleu - app.component est un seul et même component

Ce qui est intéressant dans l'usage des **components**, c'est qu'on attribue à chaque fichier une seule responsabilité. On obtient donc des fichiers "maîtres" très légers ; par exemple, dans le **component** "*see-aisle*", qui représente la vue d'un rayon, on retrouve un code très léger, car utilise différents **components**.

```
<div id="containerGlobal" class="mt-4">
    <div id="usedWhenBig">
        <div id="rowOfAisleAlert">
            <div class="col-11 mx-auto col-md-12">
                <app-aisle
                        [aisle]="aisle"
                        [smallTitle]="true"
                        [cancelButton]="true"
                        [displayActions]="false"></app-aisle>
            </div>
        </div>

        <app-add-articles-methods
                *ngIf="aisle"
                id="addArticlesMethods"
                [aisleId]="aisle.id"
                class="d-md-none d-lg-none d-xl-none"
                (productAdded)="productAdded($event)"></app-add-articles-methods>
    </div>

    <div class="row d-none d-sm-none d-md-block d-lg-block d-xl-block">
        <div class="col-11 mx-auto col-md-10 col-lg-9 col-xl-8 mt-4">
            <app-list-articles
                    *ngIf="aisle"
                    [refresh]="refresh"
                    [aisleId]="aisle.id"></app-list-articles>
        </div>
    </div>
</div>
```

Code du component see-aisle

Vue du component see-aisle

On remarque que ce dernier utilise trois **components** pour afficher cette vue, qui sont des balises HTML :

- **<app-aisle>**
- **<app-add-articles-methods>**
- **<app-list-articles>**

Pour bien comprendre l'intérêt des **components**, regardez attentivement l'encadré "vert" dans la capture d'écran ci-dessus, avec le nom du rayon et le bouton "Annuler" ; ce dernier présente le rayon dans lequel on est, avec les actions possibles.

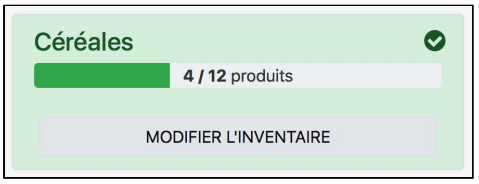
Dans la vue principale de l'application, qui liste tous les rayons, j'ai aisément pu ré-utiliser ce **component** pour garder une cohérence d'affichage, juste en jouant avec quelques attributs de la balise associée.

Component see-aisle

```
<app-aisle
        [aisle]="aisle"
        [smallTitle]="true"
        [cancelButton]="true"
        [displayActions]="false"></app-aisle>
```

Component see-inventory

```
<app-aisle
        [aisle]="aisle"
        [displayActions]="true"></app-aisle>
```



Capacités d'adaptations

Premièrement, l'application est capable de s'adapter aux différents dispositifs. En effet, grâce à Bootstrap, j'ai pu gérer les différents types d'appareils.

Par exemple, afficher la liste de tous les produits dans un rayon donné n'est absolument pas pertinent sur un petit dispositif alors utilisé pour faire l'inventaire, non pour le gérer. Cependant, pouvoir gérer les produits d'un rayon sur un PC de bureau a du sens. Dans le même **component**, on retrouve donc :

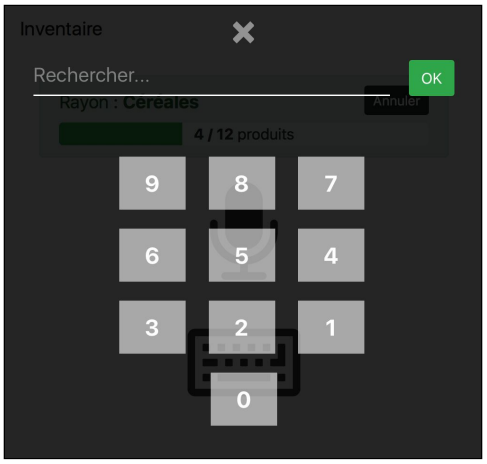
Vue sur un PC de bureau

Vue sur un petit dispositif

De plus, dans la vue sur petits dispositifs, les icônes sont les uns en dessous des autres. Cependant, utiliser cette méthode si on est en mode paysage ferait perdre beaucoup de place et impliquerait un scroll *(ou de très petits icônes)*. J'ai donc géré ce cas aussi :

Vue sur un petit dispositif en paysage

L'adaptation vient aussi avec les méthodes pour faire l'inventaire sur un produit donné. On peut utiliser la voix *(si jamais on a les mains encombrées)* ou un clavier, mais un clavier spécial, fait pour les petits dispositifs :



Vue clavier petits dispositifs

Cependant, malgré la diversité des projets, et souhaitant faire comme Arnaud avec React Native, je n'ai pu trouver un *plugin* pour scanner les codes-barres avec la caméra. Le développer est possible, mais cela prendrai un certain temps.

Pour tester toutes ces capacités d'adaptation, il suffit de prendre un navigateur et de jouer avec la taille de ce dernier. Si on réduit, alors on sera considéré comme un "petit dispositif" servant à scanner les produits pour faire l'inventaire. Au contraire, en agrandissant la taille du navigateur, on sera considéré comme un "PC de bureau" pour gérer l'inventaire.

Conclusion

Nous avons vu qu'il pouvait y avoir d'énormes différences dans l'utilisation d'un logiciel en fonction de la technologie que nous avons choisie.

Par exemple, Bootstrap ne s'adapte pas réellement au contexte ; n'étant qu'une bibliothèque CSS, on ne peut pas faire grand chose avec, à l'inverse des deux autres technologies utilisées.

En revanche, lorsqu'il faut positionner les éléments, le plus simple reste avec Bootstrap et son système de grille. Même si React Native reprend certains principes du CSS, il ne gère pas encore tout, et le positionnement peut parfois être plus complexe. Concernant Angular, gérant parfaitement le CSS et donc Bootstrap, on est totalement libre la dessus, et on peut donc utiliser Bootstrap *(ou autre)*.

La ou React Native et Angular reprennent l'avantage, c'est dans la modularité. Avec Bootstrap, on est obligé de répéter tout ou partie du code. Grâce au techniques de fonctionnement des deux premiers, on peut aisément exporter une partie du code pour la réutiliser ailleurs.

Un des gros avantages aussi pour React Native et Angular est que ce sont des frameworks relativement populaires, avec des communautés derrière, proposant de nombreux plugins. Cependant, pour le premier, dû à son "jeune âge", il a peu de maturité : il évolue *(trop ?)* régulièrement, certains plugins sont déjà obsolètes, certains manquent cruellement, il n'est même pas en version 1, ...

Comparé à Angular et Bootstrap, React Native est très proche du dispositif sur lequel il évolue : l'accès à ses capteurs n'en sont que facilités. Pour Angular, il faut utiliser des plugins *(s'ils existent)* qui ne fonctionnent qu'avec certains navigateurs *(notamment le plugin pour la reconnaissance vocale utilisé ne fonctionne qu'avec Google Chrome)*. Pour Bootstrap, on est contraint d'utiliser la bibliothèque jQuery pour avoir certaines fonctionnalités ; sans cette dernière, on est limité à du design.

Au final, il faut surtout développer pas en fonction de "ce qu'on aime" mais surtout en fonction de ce qu'on veut faire :

- Si on ne veut faire qu'un simple Dashboard, avons-nous vraiment besoin de développer proche du dispositif pour avoir accès à ses capteurs ?
- Si on souhaite faire une application mobile capable de scanner les aliments, une web-app est-elle suffisante ou devons-nous passer par du natif *(ou pseudo-natif comme React Native)* ?
- Si on n'a pas besoin d'accéder au capteurs mais on souhaite faire un panel d'administration du magasin, est-ce qu'utiliser des components serait une bonne idée ?