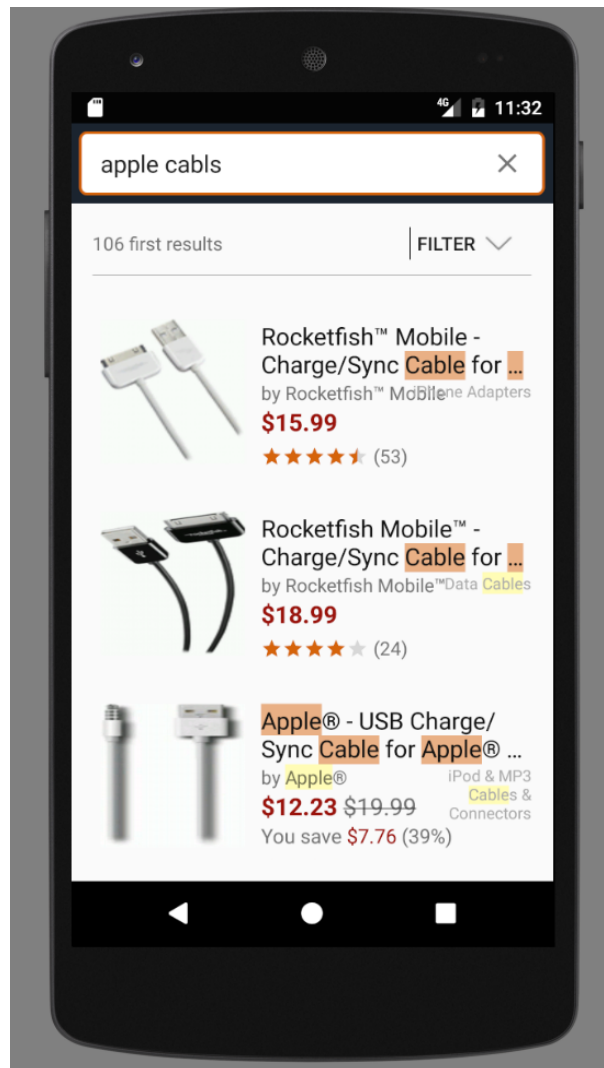


WorkShop Algolia @ WildCodeSchool

Se concentrer sur l'Expérience Utilisateur grâce aux APIs:
coder l'interface de recherche d'Amazon en quelques heures



Étape 0 : Les données

Pour cet atelier, on va utiliser un jeu de données de produits, chacun décrit par des attributs :

```
{  
  "name": "Apple - iPhone 5s 16GB Cell Phone - Space Gray (Verizon Wireless)",  
  "thumbnailImage": "http://img.bbystatic.com/.../1752456_s.gif",  
  "shipping": "Free shipping",  
  "type": "HardGood",  
  "shortDescription": "4-inch Retina display ... HD video recording FaceTime HD camera",  
  "bestSellingRank": 339,  
  "salePrice": 699.99,  
  "manufacturer": "Apple",  
  "promoPrice": 699.99,  
  "promoted": false,  
  "customerReviewCount": 1397,  
  "category": "Cell Phones with Plans",  
  "image": "http://img.bbystatic.com/.../1752456_sc.jpg",  
  "url": "http://www.bestbuy.com/site/apple-iphone-...",  
  "objectID": "1752456"  
}
```

Exemple de produit : iPhone 5s 16Gb par Apple, vendu 699,99\$

Ces données sont déjà chargées dans vos comptes Algolia, et vous pourrez y accéder avec les identifiants que vous avez reçu. Vous avez donc un moteur de recherche qui contient vos données et attend qu'on cherche dedans : à vous de développer l'application qui servira d'interface utilisateur !

Étape 1 : Partons d'une liste de noms

Notre premier objectif va être de lister les noms des produits, et de chercher dans cette liste.

On va partir d'une application vide, y ajouter Algolia, et poser la base de notre interface : une barre de recherche, et une liste de résultats.

Mise en place du projet : template et bibliothèque

Pour commencer, récupérez le template d'application duquel on va partir : <https://github.com/PLNech/workshop-wildcodeschool>

Ouvrez le dans Android Studio et lancez l'application pour découvrir un magnifique *Hello World*. L'aventure commence ici ! 🚀

Commençons par ajouter Algolia au projet. On va utiliser [InstantSearch Android](#), qui a été spécialement conçue pour faciliter la vie d'un développeur Android qui veut utiliser Algolia. Pour ajouter InstantSearch dans votre projet, rajoutez la dépendance suivante dans le *build.gradle* de votre application :

compile 'com.algolia:instantsearch-android:0.5.2'

Une fois la dépendance récupérée (n'oubliez pas de Sync Gradle), vous pouvez commencer à utiliser les composants que vous fournit InstantSearch Android.

Ajouter les widgets

InstantSearch Android se base sur un système de widgets qui communiquent lorsque votre utilisateur interagit avec votre application. Ici, nous allons utiliser deux Widgets essentiels : la **SearchBox** qui permet de taper une recherche et les **Hits** qui affiche les résultats correspondants.

Ouvrez le layout *main_activity.xml* pour y ajouter les deux widgets :

SearchBox

```
<com.algolia.instantsearch.ui.views.SearchBox  
    android:id="@+id/searchBox"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

La **SearchBox** est une extension de la vue système [SearchView](#). Vous pouvez donc utiliser les attributs de SearchView pour la configurer :

- *android:queryHint* pour choisir le texte qui s'affiche avant que l'utilisateur commence à écrire.
- *android:iconifiedByDefault* pour réduire la SearchView à une icône avant que l'utilisateur n'appuie dessus, plutôt que de s'afficher ouverte dès le début.

D'après vous, quel réglage de *iconifiedByDefault* est le plus adapté pour donner envie d'interagir avec l'interface ?

De la même manière, quel *queryHint* pouvez vous choisir pour inciter l'utilisateur à commencer à chercher des produits ?

Ajoutez à votre **SearchBox** les attributs qui vous semblent appropriés.

Au delà des attributs que propose *SearchView*, la **SearchBox** dispose d'attributs permettant de configurer son comportement:

- *algolia:autofocus* pour placer le curseur dans la SearchBox au lancement de l'activité, faisant apparaître le clavier.
- *algolia:submitButtonEnabled* pour placer le curseur dans la SearchBox au lancement de l'activité, faisant apparaître le clavier.

L'autofocus invite l'utilisateur à interagir immédiatement avec votre interface. Cela dit, l'apparition du clavier va cacher la majorité des résultats de recherche, qui aident à comprendre ce qu'on pourrait chercher.

Quel réglage d'autofocus vous semble préférable ?

submitButtonEnabled permet d'afficher le bouton Submit d'une SearchView, que nous cachons par défaut dans la SearchBox. L'objectif est de faire une interface de *search-as-you-type*, où les résultats s'affichent au fur et à mesure que l'utilisateur tape : comme on va envoyer une requête à Algolia à chaque caractère, un bouton Submit est inutile (pas besoin d'appuyer dessus pour avoir des résultats) voire trompeur (on s'imagine que rien n'est envoyé tant qu'on n'appuie pas dessus) !

Hits

```
<com.algolia.instantsearch.ui.views.Hits
    android:id="@+id/hits"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

Les **Hits** sont une extension de la vue système [RecyclerView](#). À nouveau, tous les attributs classiques peuvent être utilisés, et nous en définissons deux nouveaux :

- *algolia:hitsPerPage* va décider combien de résultats sont chargés par page. La pagination est gérée automatiquement dans les hits avec un défilement infini : lorsqu'on approche de la fin d'une page, les résultats de la page suivante seront chargés et ajoutés automatiquement à la liste. Il faut trouver un juste milieu entre des pages trop longues (on gâcherait de la bande passante en téléchargeant des données inutiles, et le forfait de nos utilisateurs en prendrait un coup) et des pages trop courtes (on passerait notre temps à charger de nouveaux résultats, et l'interface n'aurait plus l'air fluide).

Le réglage par défaut de 20 résultats par page est un bon compromis dans la plupart des cas, on va garder cette valeur pour notre atelier.

- *algolia:remainingItemsBeforeLoading* vous permet de choisir quand Algolia va charger la page suivante : lorsqu'il restera N résultats à afficher avant d'atteindre la fin de la liste, une requête est envoyée pour obtenir la suite des résultats à ajouter aux **Hits**. Si cette valeur est trop petite, l'utilisateur va arriver en bas de la page avant qu'on ait le temps de charger les résultats, mais si elle est trop grande on risque de charger des résultats inutiles : il va falloir à nouveau trouver un compromis.

D'après vous, à quel moment faut-il charger les nouveaux résultats ?

Utilisez la valeur qui vous semble appropriée.

Créer le layout pour les hits

Pour qu'InstantSearch sache comment afficher les résultats, vous allez lui donner un layout qui définit leur apparence. Pour cette première étape, on va partir d'un layout très simple qui affichera juste le nom du produit.

Ce layout va permettre à InstantSearch d'instancier automatiquement des vues pour chaque résultat, et d'associer à chaque vue les bonnes données grâce à la [Data Binding Library](#). Le Data Binding d'Android permet de faire beaucoup de choses, mais pour cet atelier nous allons nous contenter de laisser InstantSearch s'en servir pour nous dans le layout de nos résultats de recherche.

Activez le databinding en ajoutant la ligne suivante dans le bloc *android* de votre *build.gradle*:

```
dataBinding.enabled true
```

Créez un nouveau layout appelé *hits_item.xml* dans lequel vous allez ajouter ce qui suit.

Un layout de data-binding est un layout standard, mais entouré d'un tag `<layout>`:

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
  <RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
  </RelativeLayout>
</layout>
```

Le tag `<layout>` ne peut avoir qu'un enfant direct: il faut donc mettre un *ViewGroup* (ici un *RelativeLayout*) à l'intérieur qui contiendra toutes les autres vues.

On va ajouter dans ce *RelativeLayout* une *TextView* qui servira à afficher le nom du produit, soit *name* dans nos données :

```
<TextView
    android:id="@+id/product_name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    algolia:attribute="@{"name"}"/>
```

Vous remarquerez que le dernier attribut utilise la forme `@{}` : c'est la syntaxe qui permet de passer des valeurs du layout à votre application avec le Data Binding. Ici nous transmettons une *string*, le nom de l'attribut que nous souhaitons afficher dans cette vue.

Il reste à indiquer aux Hits que ce layout servira à afficher les résultats, en ajoutant aux Hits l'attribut *algolia:itemLayout* avec la référence de votre layout:

```
algolia:itemLayout="@layout/hits_item"
```

Initialiser Algolia et faire une première recherche

Maintenant qu'on a créé notre interface et défini comment afficher les résultats, il ne reste plus qu'à relier les widgets à Algolia. Le coeur d'Algolia InstantSearch est le **Searcher**, qui va se charger de l'interaction avec notre moteur de recherche.

Créez un **Searcher** dans *MainActivity#onCreate* en spécifiant votre APPID, API KEY et le nom de l'index qu'on veut utiliser :

```
private static final String ALGOLIA_APP_ID = "DB842M0JOU";
private static final String ALGOLIA_API_KEY = "VOTRE_SEARCH_API_KEY";
private static final String ALGOLIA_INDEX_NAME = "VOTRE_INDEX";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Searcher searcher = new Searcher(ALGOLIA_APP_ID, ALGOLIA_API_KEY, ALGOLIA_INDEX_NAME);
```

Il faut maintenant relier notre interface au **Searcher**. C'est le rôle du **InstantSearchHelper**, qui va parcourir votre layout pour relier vos Widgets au Searcher :

```
new InstantSearchHelper(this, searcher);
```

Enfin, on veut afficher des produits dès le lancement de l'application, sans attendre que l'utilisateur cherche quelque-chose. On va donc demander au **Searcher** de faire une recherche vide, ce qui va afficher les résultats les plus populaires :

```
searcher.search(); // Do a search with empty query
```

Lancez l'application et tapez dans la barre de recherche : à chaque caractère, la requête est envoyée à Algolia et les résultats s'affichent.

Étape 2 : Des produits plus sexy

On a maintenant une base d'interface de recherche, mais on peut pas dire qu'elle donne envie d'être utilisée. Et si on affichait un peu mieux nos produits ?

Ajouter des vues pour les autres attributs

Maintenant qu'InstantSearch utilise votre layout pour afficher les produits, il suffit de le développer pour améliorer leur affichage !

On va rajouter une vue pour chaque attribut de nos données qu'on veut afficher :

- L'image du produit (*image*)
- Son prix (*salePrice*)
- Son fabricant (*manufacturer*)
- La catégorie dont il fait partie (*category*)

Pour chaque attribut, il va falloir ajouter une vue dans votre layout : Utilisez une *ImageView* pour afficher l'image et des *TextViews* pour les autres attributs.

De la même manière que pour *name*, il suffit d'ajouter à une vue l'attribut

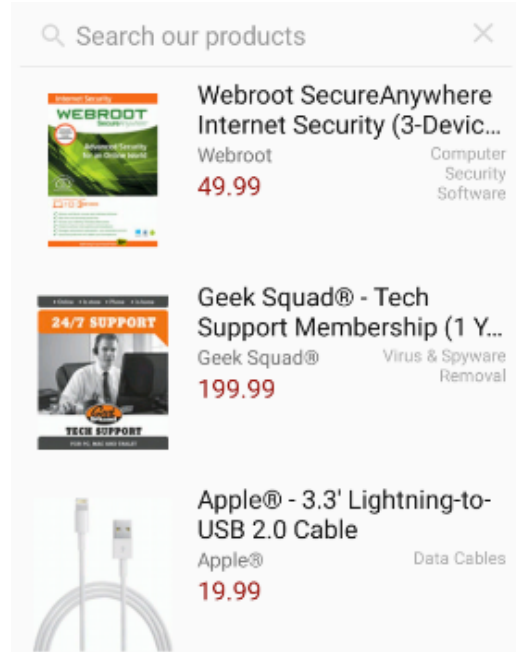
`algolia:attribute='{unAttributDeVosObjets}'` pour qu'InstantSearch utilise cette vue pour afficher *unAttributDeVosObjets*.

Une fois que les vues sont ajoutées dans votre layout, on va les configurer pour leur donner une apparence semblable à l'application Amazon :

- Nom : écrit large, sur deux lignes avec "..." si il dépasse
- Image : à gauche, largeur et hauteur fixe, centrée et ajustée au cadre
- Fabricant : en plus petit et en gris, à gauche sous le nom du produit
- Catégorie : Même taille que le fabricant mais plus clair, à droite sous le nom
- Prix : en rouge sombre (*#97100b*) et gras, sous le fabricant

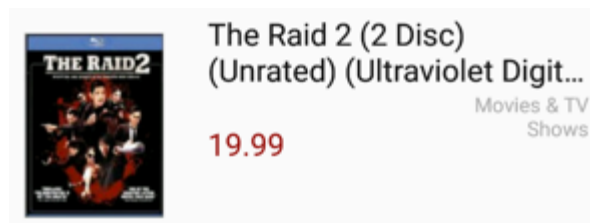
Appliquez ces critères en utilisant les attributs des `TextView` et de `ImageView`.

Votre interface devrait ressembler à quelque-chose comme ça :



Gérer les attributs qui peuvent être null

Vous l'avez peut-être déjà remarqué, mais tous les objets de notre dataset ne sont pas aussi précis : certains produits n'ont pas de fabricant indiqué, et d'autres n'ont pas de catégories.



Produit dont l'attribut manufacturer est null

L'absence du fabricant déséquilibre le layout en faisant flotter le prix. On va éviter ça en créant une `TextView` spécialisée : il suffit alors de surcharger `setText()` pour vérifier d'abord si l'attribut est `null`, et si oui de cacher la vue en lui mettant une visibilité `GONE`.

Créez une classe `NotNullTextView` comme décrit ci-dessus, puis modifiez votre layout en utilisant votre `NotNullTextView` pour l'attribut *manufacturer*.

Ajouter des statistiques

Votre application ressemble désormais à une vraie liste de produits, et un utilisateur pourrait commencer à s'en servir. Pour autant, il manque un indicateur essentiel d'une interface de recherche : le **nombre de résultats** pour la requête en cours.

InstantSearch fournit un widget qui va vous servir à afficher des informations sur les résultats affichés (ou sur l'erreur reçue).

Ajoutez dans le layout de votre activité un nouveau widget : Stats.

```
<com.algolia.instantsearch.ui.views.Stats
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

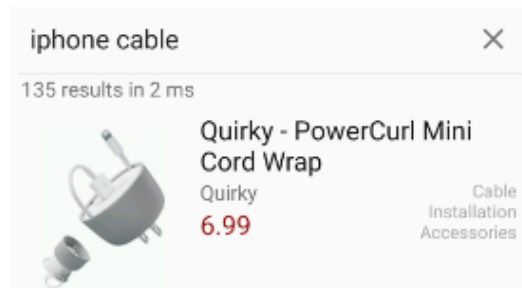
Vous pouvez configurer ce qu'il affiche avec l'attribut *algolia:resultTemplate*. Ce template s'affichera pour chaque résultat en remplaçant les placeholders suivants par les valeurs appropriées :

- *{nbHits}* -> nombre total de résultats
- *{hitsPerPage}* -> nombre de résultats par page
- *{nbPages}* -> nombre de pages
- *{page}* -> numéro de la page actuelle
- *{query}* -> texte de la requête

Choisissez ce que vous voulez afficher en recevant des résultats : quelles informations seraient utiles à l'utilisateur ?

Étape 3 : Expliquer les résultats

Quand on cherche quelque-chose, c'est toujours désagréable d'obtenir des résultats qui n'ont rien à voir avec ce qu'on voudrait trouver :



Exemple de résultat surprenant : quel est le rapport entre ce “Quirky” et ma requête ?

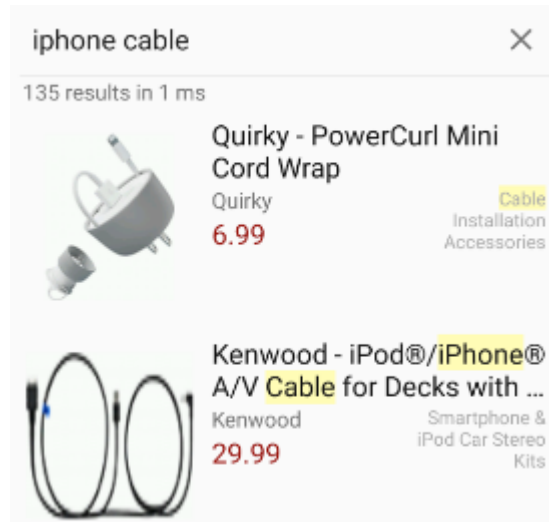
Pourtant ce produit est pertinent : il s'agit d'un enrouleur de câble qui contient “iPhone” dans sa description, et “cable” dans sa catégorie. Comment peut-on expliquer ça à notre utilisateur ?

Comme on n'affiche pas la description dans notre interface, on ne va pas pouvoir faire grand chose pour ce premier attribut.

En revanche, on pourrait souligner que le terme “cable” se retrouve dans la catégorie : c'est le rôle du **Highlighting**, qui consiste à surligner les éléments de la requête qu'on retrouve dans le résultat.

Activer le highlighting dans le layout des hits

Avec InstantSearch, il vous suffit d'ajouter un attribut `algolia:highlighted='@{true}'` à une vue de votre `hits_template` pour qu'elle soit surlignée à chaque requête :



Vous pouvez choisir la couleur avec l'attribut `algolia:highlightingColor`:

```
algolia:highlightingColor='@{"@android:color/holo_red_dark"}'/>
```

Vous pouvez l'utiliser comme ci-dessus avec une couleur système, ou utiliser une couleur définie dans votre `colors.xml` :

```
algolia:highlightingColor='@{"@color/colorHighlight"}'/>
```

Choisissez les attributs que vous voulez surligner: utilisez une couleur particulière si elle vous paraît plus appropriée que celle par défaut.

Relancez l'application et faites une recherche : les résultats sont beaucoup plus compréhensibles pour vos utilisateurs !

Étape 4 : Des résultats plus pertinents

Pendant les étapes précédentes, vous avez peut-être remarqué que les résultats s'affichent dans l'ordre alphabétique. C'est rarement la meilleure option : si on affiche les produits triés par meilleure vente, ou en excluant ceux qui ne sont pas en stock, on augmente les chances que l'utilisateur trouve ce qu'il cherche : c'est donc important de configurer son moteur de recherche pour lui expliquer **ce qu'est un résultat pertinent**.

On va configurer Algolia en allant dans l'[explorateur d'index](#) pour changer quelques réglages :


Dire au moteur comment il doit chercher

Le réglage [Searchable Attributes](#) définit les attributs où le moteur va chercher ce que tape l'utilisateur. Ils lui permettent de répondre à deux questions :

- Quels attributs de nos objets pourraient contenir les termes tapés par l'utilisateur ?
- Si deux produits contiennent le terme cherché mais dans des attributs différents, quel produit est le plus pertinent ?


Choisissez les attributs que vos utilisateurs pourront chercher ; ajoutez les dans *Searchable Attributes*.


Basic Settings



Searchable & Ranking Attributes

Configure the list of attributes you want to search in and the one you want to use for the ranking.

**Searchable Attributes**
(ordered by importance)

**Not Configured**
By default, all records attributes are considered as searchable.

➕ ADD A SEARCHABLE ATTRIBUTE

Ordered list of string attributes you want to search in. You need to sort them by order of importance.

Réglage des Searchable Attributes

Triez ces attributs en mettant les plus importants d'abord : un produit contenant le mot clé dans *name* est-il plus pertinent qu'un produit contenant le mot clé dans *category* ?

Si la position du mot clé dans l'attribut est importante, marquez le *Ordered* : un match au début du nom est-il plus pertinent que vers la fin ? Est-ce pareil pour le fabricant ?

Classer les résultats


Les réglages qu'on vient de faire vont permettre de trouver les résultats pertinents pour une requête donnée. Mais dans quel ordre veut-on afficher tous ces résultats ?

La réponse va dépendre de votre cas d'utilisation. Un site de vente de vêtements en ligne affichera par exemple les produits en stock d'abord, avant de les trier de la meilleure vente à celui qui se vend le moins. Une application qui recommande des restaurants locaux va plutôt trier les résultats par proximité géographique. À chaque fois, il faut trouver la donnée qui va permettre de classer les résultats : elle dépend souvent des **objectifs métiers** de l'application (vendre des vêtements, aider à trouver un restaurant à côté, ...)

Pour cette application, notre objectif est de faire acheter ces produits. Quel attribut de nos produits peut aider à les classer en mettant les meilleures ventes d'abord ?

Ajoutez cet attribut dans *Custom Ranking*. Choisissez si vous voulez classer par ordre croissant ou décroissant.

Custom Ranking Attributes
(ordered by importance)

**Not Configured**
The ranking doesn't currently include any popularity (business) metric, you should include one here.

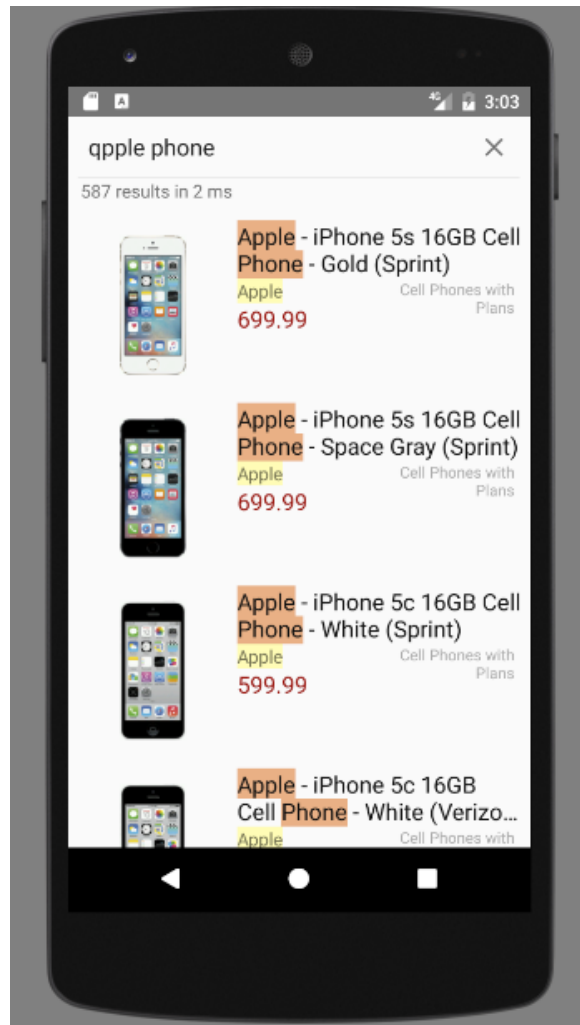
ADD A RANKING ATTRIBUTE

Specify here the attributes that indicate the popularity of each record (e.g. number of likes, views or sales) to configure the custom ranking criteria.

Réglage du Custom Ranking

5. Pour aller plus loin

On arrive à la fin de cet atelier, et le résultat final est assez satisfaisant :



Notre application utilise un moteur de recherche très performant, les résultats qu'elle affiche sont pertinents et son interface est réfléchie pour optimiser l'expérience de notre utilisateur : c'est déjà mieux que la plupart des interfaces qu'on rencontre sur internet ou dans des applications !

Pour autant, on peut toujours mieux faire : en réfléchissant à ce qui manque à vos utilisateurs, à ce qui pourrait les satisfaire davantage ou les inciter à acheter vos produits, vous aurez sûrement des idées de points à améliorer où de choses à rajouter.

Voilà déjà quelques pistes pour aller plus loin avec cette interface :

- Le prix n'a pas d'unité. On peut faire une *TextView* spécialisée pour afficher un dollar avant la valeur de *salePrice* !
- Certains produits n'ont pas de *category* définie. Et si on affichait leur *type* à la place ?
- Certains produits ont une promotion (*promoPrice*). On pourrait afficher comme Amazon la promotion suivie du prix original rayé : **18\$** ~~24\$~~
- La promotion serait encore plus motivante si on indique combien l'utilisateur économise sur une autre ligne: Amazon indique "You save **\$12.00** (32%)"
- Quand la requête ne donne aucun résultat, l'écran est vide. En utilisant sur nos Hits le même mécanisme d'[emptyView](#) qu'avec une *AdapterView*, on pourrait afficher une vue qui incite l'utilisateur à chercher autre-chose : on pourrait lui afficher des idées de requêtes, décrire nos produits, mettre un message amusant qui incite à découvrir autre-chose... À vous d'être créatifs :)