Séance nº 2 : *Todo* liste

1 Objectif

L'objectif de ce TP est de se familiariser avec les listes sous Android. Il s'agit principalement de mettre en pratique les exemples du cours, qu'il faut donc lire **consciencieusement**. Des indications sont fournies afin de faciliter certains passages chronophages n'apportant rien sur le fond. L'écran final de l'application doit ressembler à l'écran présenté Fig. 1.

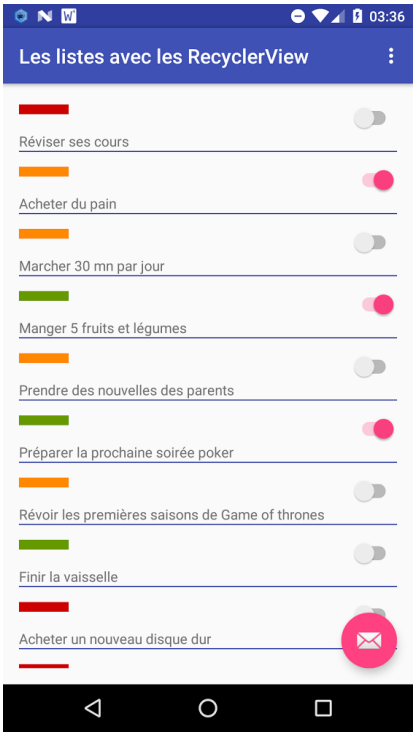

Figure 1 : Page principale

2 Développement

1. Créer une nouvelle application en commençant par une activité de type *Basic Activity*.
2. Utiliser le menu « VCS » pour demander le versionnage de l'application *via* un dépôt Git. Choisir l'option *Enable Version Control Integration*. Par défaut, seul un dépôt local est créé (ce qui peut être suffisant suivant les besoins). Aller dans *File / settings / Version Control* pour compléter la configuration s'il faut synchroniser

ce dépôt avec un dépôt distant [1] . Penser à faire des commits fréquents (`Ctrl+K` en principe), idéalement un à chaque fin de question.

3. Exécuter l'application générée pour vérifier qu'elle fonctionne normalement.
4. Créer une classe `TodoItem` permettant de gérer un item d'une *todo* liste. Cette liste doit être dotée :
   — d'un label (chaîne),
   — d'un tag (énumération sur les valeurs `Faible`, `Normal`, `Important`),
   — d'un booléen permettant de savoir si la tâche est réalisée (initialisé à faux).
   La classe doit contenir un constructeur (acceptant le label et un tag), ainsi que tous les *getters* et *setters* possibles. Afin de la définir rapidement, utiliser le raccourci `Alt+Inser` sous Android Studio.
5. Créer un attribut correspondant à une liste (`ArrayList<>`) de `TodoItem` dans l'activité principale.
6. Écrire une méthode `getItems()` permettant de « peupler » la liste. Un exemple de méthode pouvant servir à des tests est disponible sur l'ENT. Par la suite, cette méthode pourrait naturellement exploiter une base de données ou récupérer des données en ligne…
7. Suivre le cours pour l'implantation du `RecyclerView` (paragraphe « Les *`RecyclerView`* », partie 3 — Création). Quelques conseils :
   — utiliser un `TextView` pour le label,
   — utiliser un `ImageView` pour la couleur du tag (on conseille les dimensions 10dp×50dp),
   — utiliser un `Switch` pour symboliser l'état de l'item,
   — utiliser un simple `View` pour la ligne séparatrice (hauteur : 1dp),
   — jouer sur les *padding* du `ViewGroup` englobant pour gérer l'espacement entre les items,
   — utiliser la méthode `setBackgroundColor()` pour mettre à jour la couleur de l'`ImageView`. Il peut être nécessaire de créer un attribut `private Resources resources;` dans la classe `TodoHolder`, initialisé grâce à une instruction du type `resources = itemView.getResources();` dans le constructeur, pour pouvoir y accéder ensuite depuis la méthode `bindTodo`. L'utilisation finale ressemble alors à :
   `image.setBackgroundColor(resources.getColor(R.color.faible))`.

3 Terminé ?

Si la partie principale est terminée, introduire des *swipes* répondant aux consignes suivantes :
— un *swipe* à gauche doit cocher (dans tous les cas) l'item correspondant,
— un *swipe* à droite doit décocher (dans tous les cas) l'item correspondant.

[1] Le raccordement sur Github est natif. Pour BitBucket, il est nécessaire d'installer un plugin. Suivre la documentation sur `http://www.goprogramming.space/connecting-android-studio-project-with-bitbucket/`. Ceci étant, même avec le plugin, l'initialisation semble poser problème. Il vaut alors mieux réaliser cette opération directement en ligne de commandes (`git add .` + `git commit -m "Premier commit"` + création du dépôt sur BitBucket et suivre les indications.) Une fois ces initialisations réalisées, le reste des opérations se fait directement sous Android Studio.