Documentation technique – aTodo

Eléments utilisés

0. Introduction

Cette application consiste en notre première expérience de développement d'application Android native, ainsi, nous ne pouvons pas expliquer les détails spécifiques de fonctionnement du framework Android, et ne connaissons pas sur le bout des doigts les fonctionnalités proposées par les API du système.
Nous comprenons cependant comment interagissent entre eux les différents éléments de l'application que nous avons développés.

1. Logiciels

Afin de développer l'application mobile, l'IDE *Android Studio* ainsi que le plugin *SonarLint* ont été utilisés.

Android Studio est un fork de Google d'*IntelliJ Community* par Jetbrains. Il est l'**IDE officiel** pour développer des applications natives Android.
Il était ainsi un choix évident.

SonarLint quant à lui, est une **extension** faisant de l'analyse statique de code source d'un programme, dans notre IDE ou éditeur de texte, afin d'améliorer la qualité de code, de signaler de potentiels bugs ou mauvaises pratiques dans une liste d'avertissements, etc.

2. Langage utilisé

Etant donné que nous désirions développer une application native, nous avions le choix entre réaliser l'application en *Java* ou en *Kotlin*.

Kotlin est devenu, depuis Mai 2017, le langage officiel pour développer des applications Android, prenant la place de *Java* jusqu'alors.

Cependant, n'ayant aucune expérience en *Kotlin*, nous avons choisi de faire l'application en *Java*, étant donné que nous en avions déjà fait en cours, et que l'un de nous deux en a beaucoup fait.

3. Version d'Android

Nous avons choisi une version assez récente du SDK Android, la version *29*, correspondant à *Android 9*.

Nous avons fait ce choix puisque nous avons sur nos téléphones des versions récentes d'Android (Android 9 ou 10) et que nous ne voulions pas développer l'application sur des versions dépassées d'Android : nous voulions développer avec des API récentes.

4. Bibliothèques utilisées

Plusieurs bibliothèques ont été utilisées dans l'application : *ExAssert*, *Material*, *LocalizationActivity*, et beaucoup d'autres bibliothèques *AndroidX*.
N'ayant fait aucun test unitaire, nous ne parlerons pas des dépendances liées aux tests.

ExAssert est une bibliothèque développée par l'un de nous deux, implémentant des assertions avec un système d'exceptions catchées en interne en cas d'assertion fausse.
Sur Android, les assertions built-in (le mot-clé assert) sont toujours désactivées, contrairement à la JVM pour les applications Desktop, à laquelle on peut passer le flag -ea pour les activer.
Un support dédié à Android n'est pas implémenté, du moins pour le moment : il faudrait prendre en compte la constante *BuildConfig.DEBUG* pour désactiver les assertions pour les releases.
Code source : https://github.com/AntoineJT/ExAssert (bc1b0a1)

Material est une bibliothèque Android officielle, permettant d'utiliser les *SnackBar* dans l'application.

LocalizationActivity est une bibliothèque Android non officielle, développée par akexorcist en Kotlin, permettant de changer la langue de l'application facilement au runtime.
Code source : https://github.com/akexorcist/Localization (f49420b)

AndroidX est un ensemble de bibliothèques officielles Android consistant en des versions refactorées (réécrites) des API Android n'étant pas fournies avec le système. Elles en consistent donc un remplacement.

Parmi les bibliothèques *AndroidX* utilisées se trouvent : *ConstraintLayout*, *AppCompat*, *NavigationFragment*, *NavigationUI*, *RecyclerView*.

ConstraintLayout est un type de layout utilisé dans l'*activité ChangeLang* où les éléments sont placés relativement à des contraintes entre eux.

AppCompat est une bibliothèque permettant aux anciennes versions d'Android d'utiliser des versions de l'API plus récentes. Cela n'est peut-être pas nécessaire, mais le projet exemple généré par Android Studio les utilisaient. Nous nous pencherons davantage là-dessus si nous décidons de poursuivre le développement de l'application après le partiel.

NavigationFragment et *NavigationUI* sont des bibliothèques liées à la navigation, elles ont été ajoutées par Android Studio directement dans le projet exemple, nous ne savons pas exactement à quoi elles servent. *NavigationFragment* du moins est utilisée pour inclure le *fragment TaskList* dans l'activité principale.

RecyclerView est une bibliothèque permettant de faire une liste plus efficace, pouvant être rafraichie efficacement. Cependant, nous n'utilisons pas cette fonctionnalité, nous l'utilisons afin de faire une liste de tâche assez attrayante.

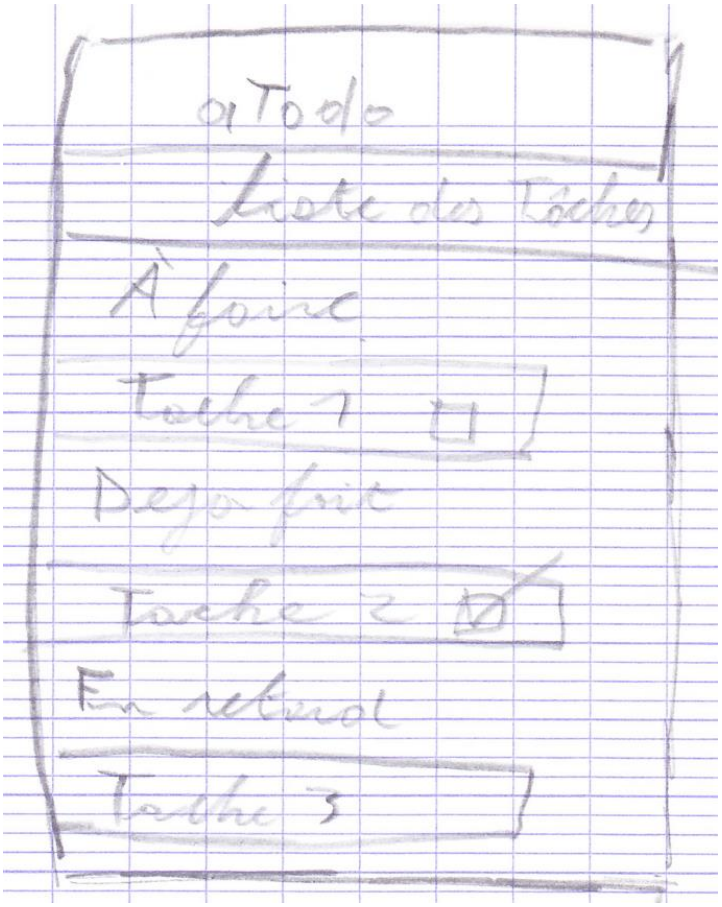
5. Langues

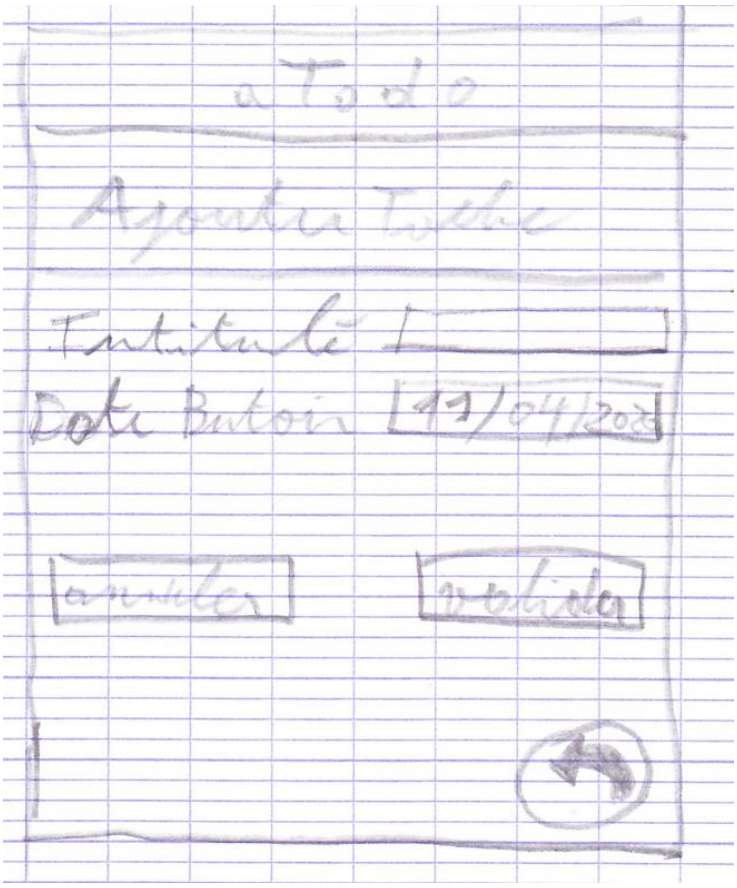
Nous avons utilisé le système d'internationalisation (i18n) inclut dans Android afin de proposer 3 langues distinctes : l'anglais, le français et l'espagnol.

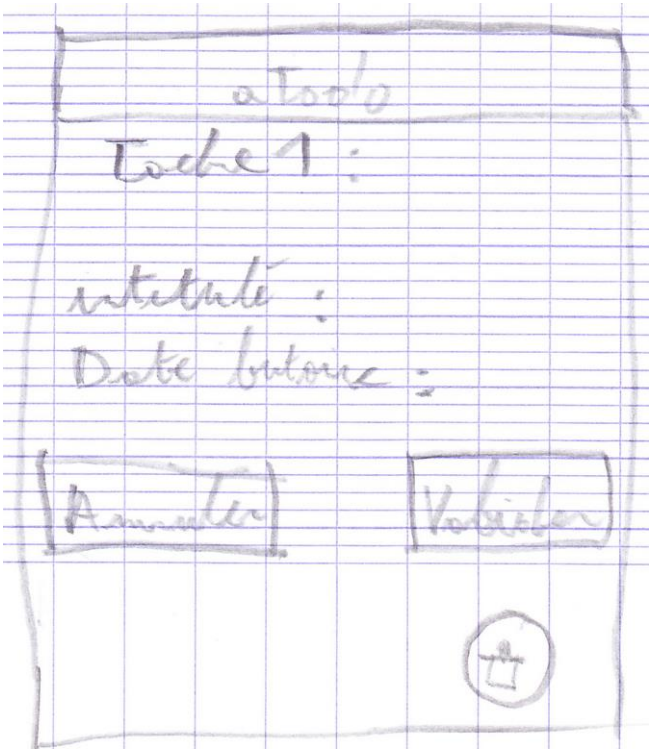
Au début, l'application était faite seulement en anglais, car une fois l'évaluation passée, le code source de cette dernière sera rendu public sur GitHub (sous licence GPLv3 ou MIT).

Travail préparatoire

1. Maquettes

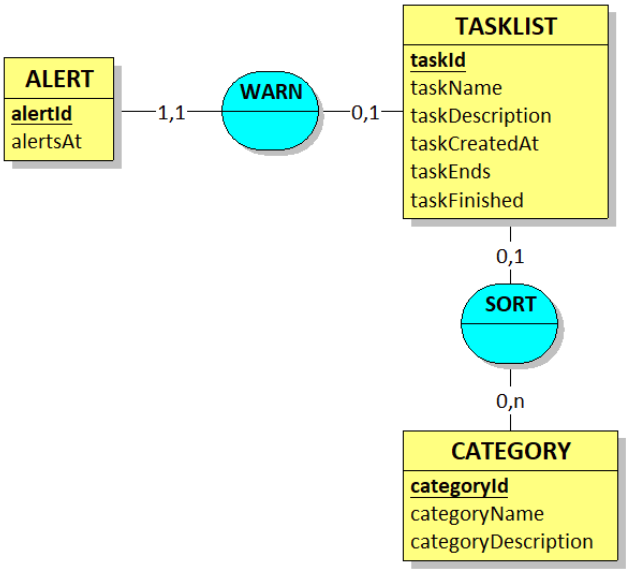






2. Modèle Conceptuel de Données

Le modèle conceptuel de données a été réalisé en anglais puisque nous comptons probablement continuer l'application après l'épreuve passée.



De ce modèle, nous n'utilisons dans l'application que la table TaskList, les autres servant à des fonctionnalités que nous n'avons pas eut le temps d'implémenter, tel que l'alerte par notification peu avant expiration de la tâche, ainsi que le système de classement des tâches par catégories.

Explication de certains snippets de code

BiMap.java

Une BiMap est une Map où on peut accéder aux clés et valeurs des deux sens.
Une clé est une valeur et une valeur est aussi une clé.

BiMap.java

```java
package com.github.antoinejt.atodo.utils;

import java.util.HashMap;
import java.util.Map;

public class BiMap<T, U> {
    private final Map<T, U> normal;
    private final Map<U, T> inverted;

    public BiMap() {
        normal = new HashMap<>();
        inverted = new HashMap<>();
    }

    public void put(T t, U u) {
        normal.put(t, u);
        inverted.put(u, t);
    }

    public Map<T, U> get() { return normal; }

    public Map<U, T> inverted() { return inverted; }
}
```

Cette implémentation est minimaliste et n'assure aucun contrat (voir programmation par contrat).
Ainsi, mal l'utiliser pourrait causer des bugs assez compliqués à dépister puis à résoudre si l'on n'a pas conscience de ses défauts manifestes.

Dans les contrats que la classe devrait garantir, il y a notamment le fait que les clés et les valeurs doivent être uniques.

En l'occurrence, si 2 valeurs sont identiques, alors l'insertion dans la seconde map, inverted, ne s'effectue logiquement pas (mais cela reste à vérifier).

CreateTaskActivity#createTask

```
77  private TaskCreationStatus createTask() {
78      try (DatabaseHandler db = DatabaseHandler.get(this.getApplicationContext())) {
79          final List<Integer> fieldsId = Arrays.asList(
80                  R.id.fieldTaskName,
81                  R.id.fieldTaskEnd,
82                  R.id.fieldTaskDescription);
83          // order: name, deadline, description
84          final List<String> values = fieldsId.stream()
85                  .map(this::getEditTextValue)
86                  .collect(Collectors.toList());
87
88          if (values.stream().anyMatch(String::isEmpty)) {
89              return TaskCreationStatus.EMPTY_FIELDS;
90          }
91
92          final String name = values.get(0);
93          final String deadline = values.get(1);
94          final String description = values.get(2);
95
96          if (!isDeadlineCorrect(deadline)) {
97              return TaskCreationStatus.DATE_FORMAT;
98          }
99
100         final boolean succeed = db.createTask(name, description, DateFormatter.parse(deadline)) != -1;
101
102         return succeed ? TaskCreationStatus.OK : TaskCreationStatus.ERROR;
103     } catch (Exception e) {
104         return TaskCreationStatus.ERROR;
105     }
106 }
```

Cette méthode permet de créer une tâche, tout en indiquant à l'utilisateur si l'opération réussi ou non, et en lui donnant des informations dans le cas d'un échec.

Tout d'abord, on récupère les 3 champs fieldTaskName, fieldTaskEnd, fieldTaskDescription.
Ensuite, grâce au stream map collect, on récupère la valeur contenue dans chaque champ.
On fait ensuite un stream anyMatch(String::isEmpty) permettant de vérifier que tous les champs sont remplis.
Dans le cas contraire, une erreur indiquant que les champs ne sont pas tous remplis est retournée à l'utilisateur.
On vérifie ensuite que la deadline est formatée correctement (en appelant une fonction imparfaite, mais qui grossièrement fait le travail).
Finalement, on tente de créer la tâche. En cas d'échec une erreur assez vague est retournée à l'utilisateur, et en cas de réussite, il est notifié du succès également.

L'indication de réussite ou d'échec est faite via l'affichage d'une SnackBar.