**Documentation technique – aTodo**

**Eléments utilisés**

1. **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.

1. **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.

1. **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.

1. **Bibliothèques utilisées**

Plusieurs bibliothèques ont été utilisées dans l’application : *ExAssert*, *Material*, *LocalizationActivity*, etbeaucoup 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](https://github.com/AntoineJT/ExAssert/commit/bc1b0a1a3fad20fd5e869de6f8b4fa50b52032e1))

*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](https://github.com/akexorcist/Localization/commit/f49420b634afd012a899c9278685a50cfd78b132))

*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 entre eux avec des contraintes.

*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.

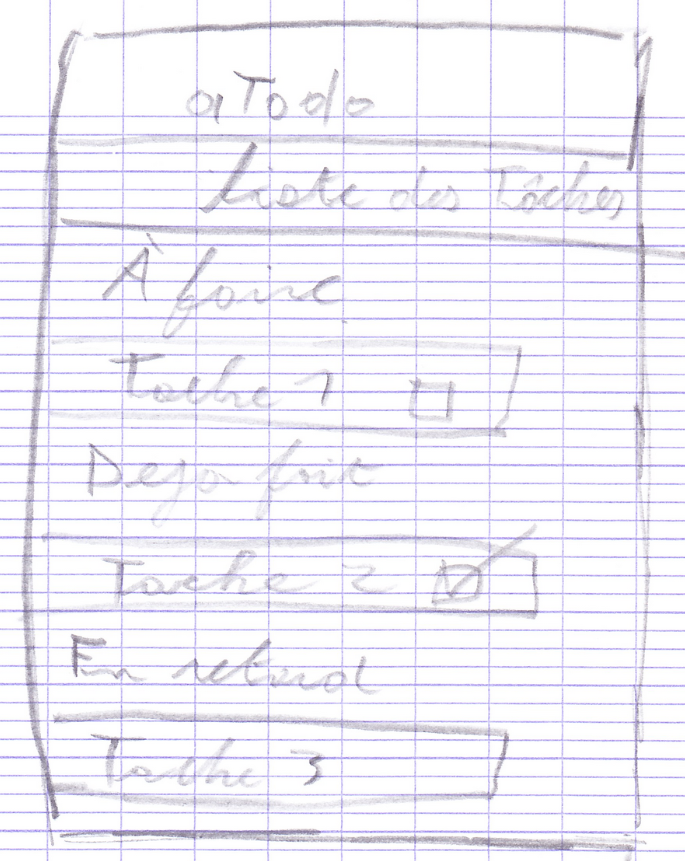
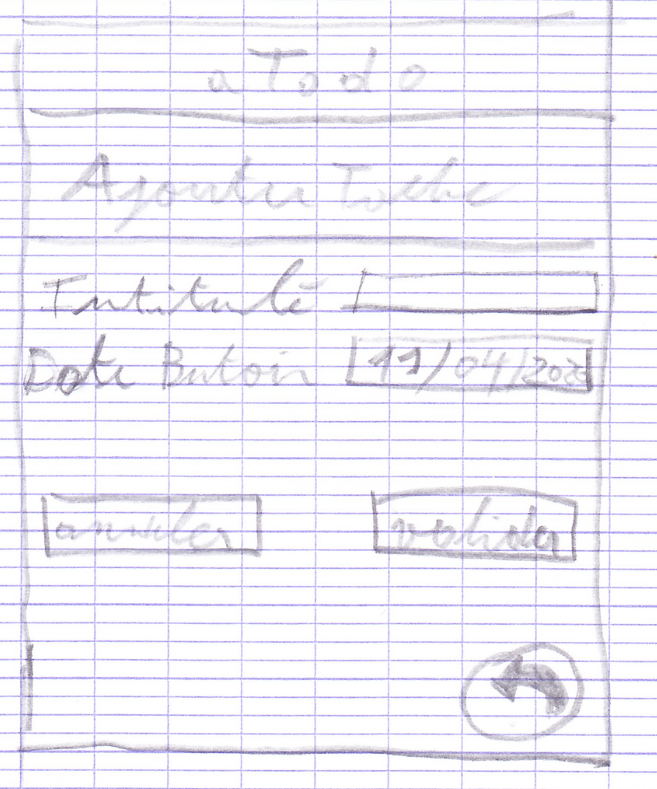
*RecyclerView* est une bibliothèque permettant de faire une liste plus efficace, pouvant être rafraichie efficacement. Cependant, nous n’utilisons pas cette fonctionnalité, nous l’utilisons afin de faire une liste de tâche assez attrayante.

1. **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.

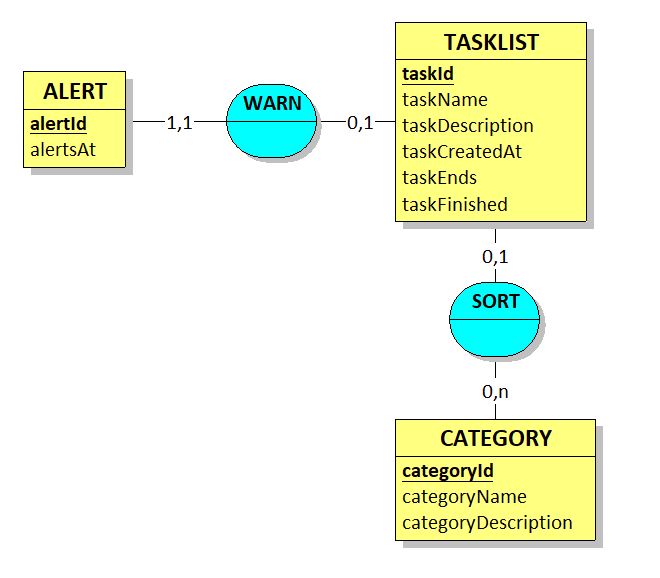
Au début, l’application était faite seulement en anglais, car une fois l’évaluation passée, le code source de cette dernière sera rendu public sur GitHub (sous licence GPLv3 ou MIT).

**Travail préparatoire**

1. **Maquettes**

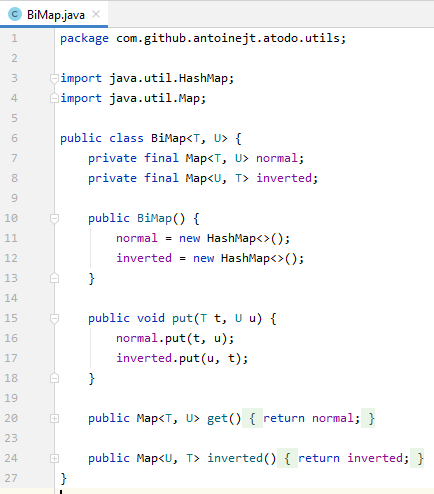


1. **Modèle Conceptuel de Données**

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*

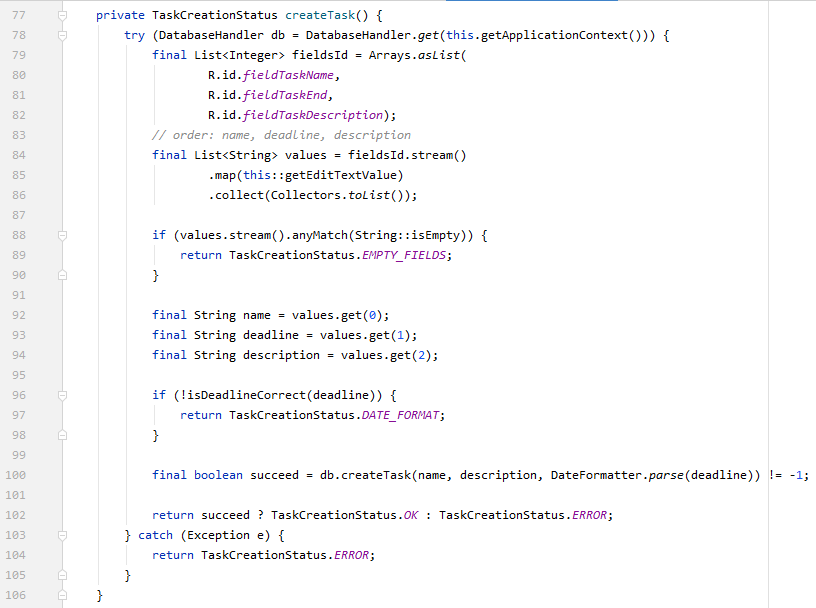
Une BiMap est une Map dans laquelle on peut accéder aux clés et aux valeurs dans les deux sens.

Cette implémentation est minimaliste et n’assure aucun contrat (voir [programmation par contrat](https://fr.wikipedia.org/wiki/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*



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.