

Melvin Campos Casares  
Maxime Liber  
Hubert Van De Walle

**RAPPORT DE PROJET**

TodoList – Groupe 30



# GitHub

*Lien du repository :* <https://github.com/melvinmajor/TodoList>

*Lien du Wiki :* <https://github.com/melvinmajor/TodoList/wiki>

# Cahier des charges - descriptif

## Quel est notre projet ?

Notre projet est une liste de tâches à faire.

L'intérêt de réaliser ce projet est qu'il peut être utilisé au quotidien pour divers tâches/choses dans divers domaines : les cours, les projets personnels comme professionnels, etc.

Cette première analyse nous à par ailleurs donné l'idée d'implémenter des catégories non spécifiées afin de laisser l'utilisateur de les créer.

Comme demandé, ce projet fonctionnera sous le modèle MVC. Nous aurons un affichage console et un affichage graphique qui sera "user-friendly".

Nous travaillerons également sous l'aspect client-serveur-socket.

L'interface graphique sera écrite sous JavaFX ou Swing.

Notre contrainte est que la todolist devra fonctionner sous Java 10 au lieu de Java 8 (celui vu au cours pratique) et que nous créerons des tests unitaires JUnit 5.

## Ce que notre projet doit absolument faire

La première chose qu'il doit pouvoir faire est de créer de nouvelles tâches, de pouvoir les compléter ainsi que de les supprimer.

En d'autres termes, il faut qu'il puisse gérer les différentes tâches et les enregistrer localement.

## Ce que notre projet serait capable de faire

* Une partie du contenu des tâches pourra être modifié par la suite par l'utilisateur (tout sauf la date de création),
* Trier par importance, date, etc.
* Filtrer par utilisateur, catégorie ou période spécifique de temps.

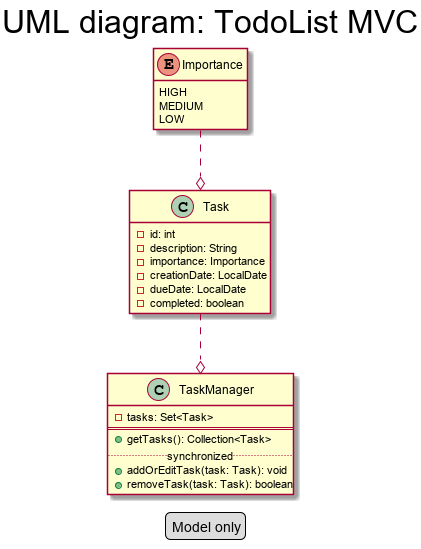
## Quelques idées d'implémentations

Parce que nous souhaitons avoir quelque chose de réellement complet, nous pourrions implémenter certaines des idées suivantes :

* Une nouvelle variable qui stockera s'il s'agit d'un travail de groupe ou non,
* D'autres options de tri ou filtre,
* Une barre de progression pour des éléments précis comme la date d'échéance ou l'importance,
* Une API JSON pour l'intégration du programme dans diverses applications par la suite (site internet, etc.).

# Diagramme UML

Ce diagramme UML est celui du modèle MVC (partie modèle uniquement).



# Choix d'implémentation effectués

## Interface graphique redimensionnable

Nous avons considéré cela comme secondaire et n'y avons prêté attention que lorsque notre affichage principal était terminé et que nous nous occupions de l'intégration des options permettant la gestion des tâches directement via l'interface graphique.

Cela a entrainé un changement au niveau de l'aspect général de l'interface graphique étant donné les contraintes liées au Grid Bag Layout, le seul layout compatible avec notre idée d'affichage après plusieurs essais infructueux avec d'autres options.

En utilisant le Grid Bag Layout, nous avons dû réadapter nos boutons en haut de la fenêtre principale pour qu'ils soient intégrés dans un menu horizontal.

# Difficultés rencontrées

## Utilisation de 2 IDE différents avec *Maven*

L'utilisation de deux Integrated Development Environment (IDE) différents n'était peut-être pas la meilleure option en soi, mais nous y avons trouvé un certain intérêt.

[IntelliJ IDEA Community Edition](https://www.jetbrains.com/idea/) est un IDE très performant et bien meilleur qu'[Eclipse](https://www.eclipse.org/).

Étonnamment, notre gestionnaire de dépendance [*"Maven"*](https://maven.apache.org/) fonctionnait bien mieux et de façon bien plus transparente avec IntelliJ qu'Eclipse.

Maven est pourtant conçu par les développeurs derrière Eclipse.

Il ne fut pas rare de devoir reconstruire notre workspace (lors de la création du serveur et puis de l'interface graphique) ou de devoir effectuer un refresh du workspace à la moindre synchronisation avec Git.

## Java 10 et 11

L'utilisation d'une version plus récente de Java n'a pas été chose facile.

En effet, Oracle ayant changé les conditions d'utilisations de Java, il nous a fallu suivre [OpenJDK](https://openjdk.java.net/) pour récupérer un Java Development Kit (JDK) pouvant être librement utilisé.

Grâce à cela, nous avons profité d'un JDK avec Java Runtime Environment (JRE) intégré, ce qui nous a évité d'autres contraintes d'implémentation avec Eclipse.

## JavaFX ou Swing ?

Au départ, nous voulions créer l'interface graphique en JavaFX et non Swing.

Cette décision était prise étant donné les problèmes majeurs liés aux DPI des écrans (Windows et Linux impacté contrairement à macOS), entre autres choses.

Malheureusement, on a très vite remarqué un manque de temps crucial et nous avons préféré utiliser Swing avec Window Builder comme point de départ.

Cela a occasionné un code très lourd et difficile à séparer, ce qui nous a permis de comprendre l'intérêt minime des systèmes WYSIWYG (What You See Is What You Get).

## Gestion du planning

Au niveau de la gestion du temps, cela n'a pas été notre point fort.

Nous pensions avoir suffisamment de temps au départ mais c'était sans compter le projet en langage C du cours d'OS ainsi que l'examen de réseau en fin de quadrimestre.

Tout cela a entrainé une certaine inconsistance au niveau du planning et une avancée ralentie de ce projet.

# Pistes d'amélioration éventuelles

## Localizer

Une détection de la langue de l'environnement dans lequel le programme fonctionne pour automatiquement s'adapter en fonction de sa base de données et à défaut, être disponible en anglais serait une amélioration envisageable à moyen terme.

Il faudra réadapter une partie du programme pour que cela soit possible et de manière aisée.

## Support des écrans HiDPI

Supporter les résolutions plus fortes pour une même taille d’écran, nécessitant un changement de DPI, pourrait être intéressant afin d’avoir la même interface quel que soit l’environnement sur lequel notre programme fonctionnerait.

Cela aiderait également à la lisibilité de l’affichage sur les écrans HiDPI.

# Conclusion

## Melvin Campos Casares

## Maxime Liber

## Hubert Van De Walle