

RAPPORT DE PROJET

Projet : Gestionnaire de listes de tâches



COUTABLE Guillaume, RULLIER Noémie
26 février 2013



Table des matières

1	Introduction	2
2	Les fonctionnalités	3
2.1	Fonctionnalités principales	3
2.2	Fonctionnalités secondaires	3
3	Storyboard, <i>paper prototyping</i> et Scénarios	4
3.1	Storyboard	4
3.2	Paper prototyping	5
3.2.1	Les maquettes	5
3.2.2	Choix réalisés	6
3.3	Scénarios	6
3.3.1	Scénario 1 - Crédit/Utilisation/Suppression de liste	6
3.3.2	Scénario 2 - Utilisation des templates	7
4	L'IHM	8
4.1	Le menu	8
4.2	Boîte à outils	9
4.3	Zone d'affichage de la liste	10
5	Le modèle	11
6	Limite de l'application	12
7	Conclusion générale	13

1 Introduction

L'objectif de ce projet fut de développer un gestionnaire avancé de tâches. Celui-ci devait permettre de créer des listes de tâches et de suivre l'avancement de celles-ci.

Afin de créer cette application que nous avons appelé *Taskinator*, nous avons établit plusieurs étapes dans l'avancement du projet. Ce rapport présentera ces étapes les unes après les autres.

2 Les fonctionnalités

La première étape fut d'analyser l'ensemble des fonctionnalités que notre application devait proposer.

2.1 Fonctionnalités principales

Voici dans un premier temps les fonctionnalités principales :

Créer une liste : cette fonctionnalité permet à l'utilisateur de créer une liste non ordonnée vide.

Créer une liste ordonnée : cette fonctionnalité permet à l'utilisateur de créer une liste ordonnée vide. L'ensemble des éléments de cette liste doivent être effectués dans un ordre précis.

Créer une tâche : cette fonctionnalité permet à l'utilisateur de créer une tâche.

Supprimer un élément : cette fonctionnalité permet de supprimer une tâche ou une liste (ordonnée ou non). Cette fonctionnalité est à manipuler avec précaution, en effet dans le cas d'une liste, la suppression de celle-ci implique aussi la suppression de tous ses éléments (listes ou tâches).

Enregistrer : cette fonctionnalité permet à l'utilisateur d'enregistrer sa liste dans un fichier sur son disque dur.

Enregistrer un template : cette fonctionnalité permet à l'utilisateur d'enregistrer la liste qu'il vient de créer comme un template afin que la structure de celle-ci soit réutilisable.

Ouvrir un template : cette fonctionnalité permet à l'utilisateur de créer une liste à partir d'un template enregistré. Il devra cependant renseigner le nom de cette liste ainsi que toutes les dates de tous les éléments. Il peut ensuite continuer à modifier cette liste.

2.2 Fonctionnalités secondaires

Voici les fonctionnalités secondaires :

Paramètre : cette fonctionnalité permet à l'utilisateur de modifier le type de l'élément sélectionné. Il pourra par exemple choisir de modifier une liste en liste ordonnée ou en une tâche. Cette fonctionnalité est à manipuler avec précaution, en effet si l'utilisateur décide de transformer une liste en tâche l'ensemble des éléments de la liste seront supprimés.

Monter / Descendre : cette fonctionnalité permet de monter ou descendre un élément dans l'arborescence de la liste. Dans le cas d'une liste, tous ces éléments sont aussi montés/descendus d'un rang. Si le changement se fait au sein d'une liste ordonnée l'ordre des éléments est aussi changé.

Aperçu : cette fonctionnalité permet à l'utilisateur d'avoir un aperçu de sa liste, ce qui permettra l'affichage d'un nombre plus important de tâches.

Gérer ses templates : cette fonctionnalité permet à l'utilisateur d'ajouter et de supprimer des templates.

3 Storyboard, paper prototyping et Scénarios

3.1 Storyboard

Le storyboard permet de montrer à quoi sert l'application. Il utilise les star people de Bill VerPlank. A la fin du storyboard, le personnage atteint son but et est satisfait. Nous avons donc ici créé notre storyboard pour notre application :

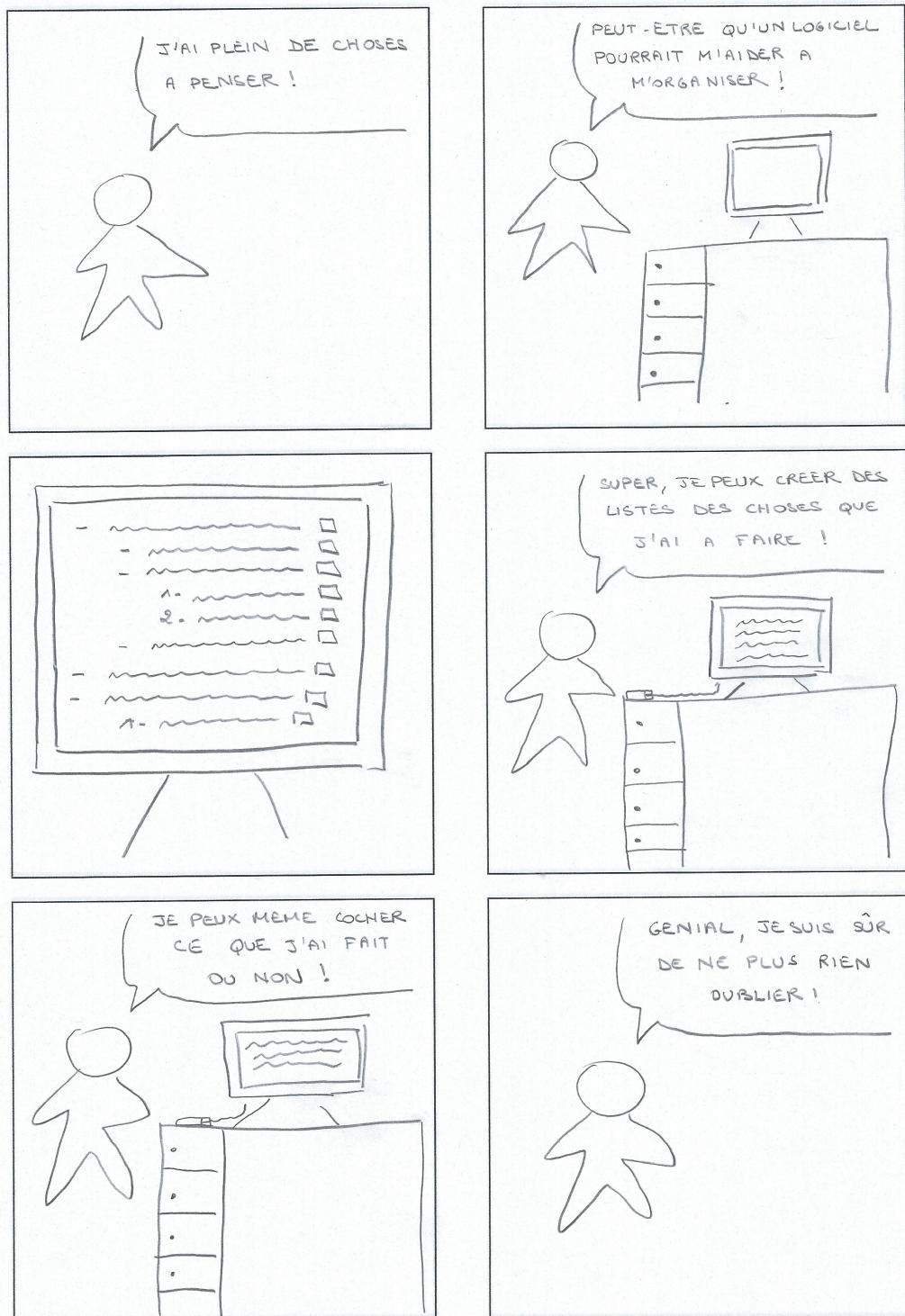


FIGURE 1 – Le storyboard

3.2 Paper prototyping

3.2.1 Les maquettes

Voici quelques unes des maquettes réalisées pour le paper prototyping. Les maquettes ici présentent sont celles qui ont été utilisées pour la réalisation de la vidéo du scénario 1 présenté ci-dessous.



FIGURE 2 – paper prototyping

3.2.2 Choix réalisés

Dans cette section, nous expliquons les choix effectués, mis en avant dans le *paper prototyping* et pourquoi nous avons choisi de mettre en place ces solutions :

- **Fonctionnalité de création (liste ordonnée ou non et tâche)** : nous avons décidé d'adopter un système de sélection de l'élément "fils". Expliquons ce système. Lorsque l'utilisateur crée une liste, un élément vide de cette liste apparaît. L'utilisateur doit alors sélectionner cet élément et définir de quel type l'élément sera. Lorsque cet élément à un type, un nouvel élément vide apparaît ; l'utilisateur peut alors définir un autre élément et ainsi de suite. Si l'utilisateur ne souhaite pas continuer à remplir cette liste, il ne tient plus compte des éléments vides.
- **Fonctionnalité Monter et Descendre** : nous avons décidé de placer ces fonctionnalités dans la boîte à outils plutôt que sur l'élément à déplacer pour un gain de temps au niveau de l'utilisation. En effet, si nous avions fait le choix de mettre les boutons *monter* et *descendre* sur l'élément, lorsque l'utilisateur aurait déplacé l'élément, il aurait perdu le focus de la souris sur ces boutons. Attention, lors de l'utilisation de cette fonctionnalité dans une liste ordonnée ; si on monte ou descend un élément de tel façon qu'un élément coché se trouve être en dessous d'un élément non coché dans la liste, alors le déplacement ne sera pas activé. (Cette dernière précaution n'a pas été implémentée)
- **Fonctionnalité Supprimer** : nous avons choisi d'ajouter cette possibilité de suppression sur chaque élément pour que l'utilisateur puisse supprimer plus rapidement. De plus nous avons décidé de faire apparaître une fenêtre d'avertissement car la suppression d'une liste sélectionnée peut entraîner la suppression d'éléments. Pour plus de sécurité, le focus de la validation de la suppression est par défaut sur *Non*.
- **Fonctionnalité Paramètre** : nous avons décidé pour cette fonctionnalité (qui peut entraîner la suppression de différents éléments) de faire, également, apparaître une fenêtre d'avertissement. Dans cette fenêtre le focus de la validation du changement est mise à *Non*. On perd cependant un peu de vitesse d'exécution (si l'utilisateur souhaite effectivement bien effectuer ce changement) au profit de plus de sécurité.

3.3 Scénarios

Afin de tester notre paper Prototype, nous avons créé plusieurs scénarios.

3.3.1 Scénario 1 - Crédation/Utilisation/Suppression de liste

Ce premier scénario a été utilisé pour tester le *paper prototyping*. Il permet de créer une liste non ordonnée et d'y ajouter une liste ordonnée avec ses propres tâches et des tâches.

1. L'utilisateur choisit quel type de liste il souhaite créer, il choisit ici une liste non ordonnée. Il lui donne un nom et une date de fin. (Pour notre scénario cette liste sera appelée la liste mère)
2. Il crée ensuite une liste ordonnée (qui est le premier élément de la liste mère). Il lui donne un nom et une date. (Pour notre scénario cette liste sera appelée la liste 1)
3. Il crée ensuite une tâche qui sera le premier élément de la liste 1. Il lui donne un nom et une date. (Pour notre scénario cette tâche sera appelée la tâche 1.1)
4. Il crée ensuite une tâche qui sera le deuxième élément de la liste 1. Il lui donne un nom et une date. (Pour notre scénario cette tâche sera appelée la tâche 1.2)
5. Il souhaite maintenant échanger les tâches 1.1 et 1.2, il sélectionne donc la tâche 1.1 et la place au dessous de la tâche 1.2
6. Une popup peut apparaître et informe l'utilisateur que l'action qu'il vient d'effectuer provoque un conflit de date. Le champs date concerné par le conflit prend une couleur orange d'avertissement.
7. L'utilisateur change le type de la liste 1 en une liste non ordonnée.
8. L'utilisateur supprime la tâche 1.2

9. L'utilisateur supprime la tâche 1. Une popup apparaît pour l'avertir que cette suppression supprimera aussi tous les éléments de la liste.

3.3.2 Scénario 2 - Utilisation des templates

Ce scénario permet de créer une liste à partir d'un template. De modifier cette liste et de la réenregistrer comme un nouveau template.

1. L'utilisateur choisit de créer une liste à partir d'un template. Il choisit ici le template qui correspond à la préparation d'un cours, puis donne un nom et une date à sa liste.
2. Il va ensuite pour tous les éléments de ce template donner une date.
3. L'utilisateur va ensuite modifier cette liste en y ajoutant une tâche à la liste mère (il lui donne un nom et une date).
4. Il va ensuite vouloir enregistrer cette nouvelle liste comme un nouveau template.

4 L'IHM

Afin de créer une interface la plus simple et explicite possible pour l'utilisateur nous avons effectué différents choix que nous allons expliquer ci-dessous.

La fenêtre principale est tout d'abord composée d'un menu, d'une boîte à outils et d'une zone réservée à l'affichage de la liste en cours de création.

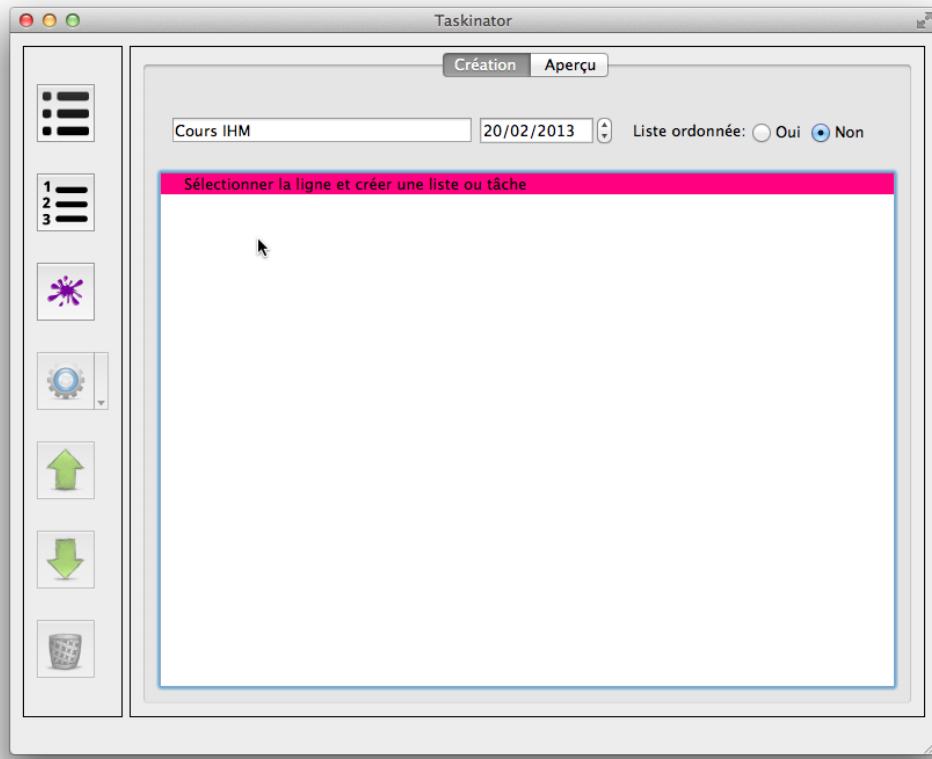


FIGURE 3 – La fenêtre principale

4.1 Le menu

Le menu permet de regrouper différentes fonctionnalités sous un thème/rubrique. On retrouvera les fonctionnalités suivantes :

- Fichier
 - **Nouveau** : qui permet de créer une nouvelle liste vide ou à partir d'un template.
 - **Ouvrir** : qui permet d'ouvrir une liste au format .tor enregistrer sur le disque dur de l'utilisateur.
 - **Ouvrir récent** : qui permet à l'utilisateur d'ouvrir l'une des cinq dernières listes ouvertes dans cette application.
 - **Enregistrer** : qui permet à l'utilisateur d'enregistrer sa liste en cours à chemin déjà spécifié.
 - **Enregistrer sous** : qui permet à l'utilisateur de choisir le chemin auquel il souhaite enregistrer sa liste et de l'enregistrer.
 - **Exporter** : qui permet à l'utilisateur d'exporter sa liste au format PDF. (Cette fonctionnalité n'a pas été implémentée)
 - **Quitter** : qui permet à l'utilisateur de quitter l'application.



FIGURE 4 – Le menu Fichier

- Édition
- **Annuler** : qui permet à l'utilisateur d'annuler une action qu'il vient d'effectuer. (Cette fonctionnalité n'a pas été implémentée)
- **Rétablissement** : qui permet à l'utilisateur de rétablir une action qu'il vient d'annuler. (Cette fonctionnalité n'a pas été implémentée)



FIGURE 5 – Le menu Fichier

- Outils
- **Enregistrer template** : qui permet à l'utilisateur d'enregistrer sa liste en tant que template au format .ulk.
- **Option** : qui permet à l'utilisateur de gérer ses templates. Il peut choisir le chemin vers lequel il souhaite enregistrer ses templates. La liste des templates présents est affichée. On peut sélectionner un template et afficher sa structure. On peut de plus importer ou supprimer un template .



FIGURE 6 – Le menu Fichier

- Aide
- **A propos** : permet à l'utilisateur d'avoir plus d'information sur l'application *Taskinator*.

4.2 Boîte à outils

Cette boîte à outils permet de rassembler toutes les fonctionnalités que l'utilisateur sera amené à utiliser fréquemment. Nous avons décidé de mettre cette boîte à outils à gauche, puisque dans notre culture le sens de lecture est de gauche à droite. Toutes les fonctionnalités présentent dans cette boîte à outils seront appliquées sur l'élément sélectionné dans la zone d'affichage de la liste. Cette boîte à outils contient les fonctionnalités suivantes :

- **Créer une liste non ordonnée (☰)** : elle permet à l'utilisateur de créer une nouvelle liste non ordonnée.
- **Créer une liste ordonnée (☷)** : elle permet à l'utilisateur de créer une nouvelle liste ordonnée.
- **Créer une tâche (*)** : elle permet à l'utilisateur de créer une nouvelle tâche.
- **Paramètre (⚙)** : elle permet à l'utilisateur de changer le type d'un élément.
- **Monter (⬆)** : elle permet à l'utilisateur de monter un élément.
- **Descendre (⬇)** : elle permet à l'utilisateur de descendre un élément.
- **Supprimer (☒)** : elle permet à l'utilisateur de supprimer un élément.

Tous les boutons permettant de réaliser ces fonctionnalités ont été implémentés grâce à des *QToolButton*. Nous avons ensuite utilisé des signaux et des slots afin de connecter l'action à réaliser aux boutons.

4.3 Zone d'affichage de la liste

Cette zone est comme indiqué réservée à l'affichage de la liste en cours de création. On pourra donc voir l'arborescence de la liste, avec toutes ses informations. Cette zone sera organisée en une arborescence d'éléments. Pour cela, nous avons utilisé un *QTreeWidget*. Celui-ci sera composé d'éléments.

Nous avons créé notre propre élément (héritant de *QWidget*) composé d'un *QLineEdit* permettant de taper le nom de l'élément, d'un *QDateEdit* permettant de rentrer la date, d'un *QToolButton* permettant d'ajouter la fonctionnalité de suppression sur l'élément et d'une *QCheckBox* permettant de valider la réalisation de la tâche. Plusieurs cas sont possibles pour l'état de la checkbox :

- Dans le cas d'un élément d'une liste ordonnée, cette checkbox sera grisée si la tâche antérieure n'est pas cochée.
- Dans le cas où l'élément est une liste, tant que la checkbox de la liste est grise, tous les éléments fils ont des checkbox grisées.
- Dans le cas d'une liste, sa checkbox sera automatiquement cochée lorsque tous les éléments de la liste seront eux aussi cochés.

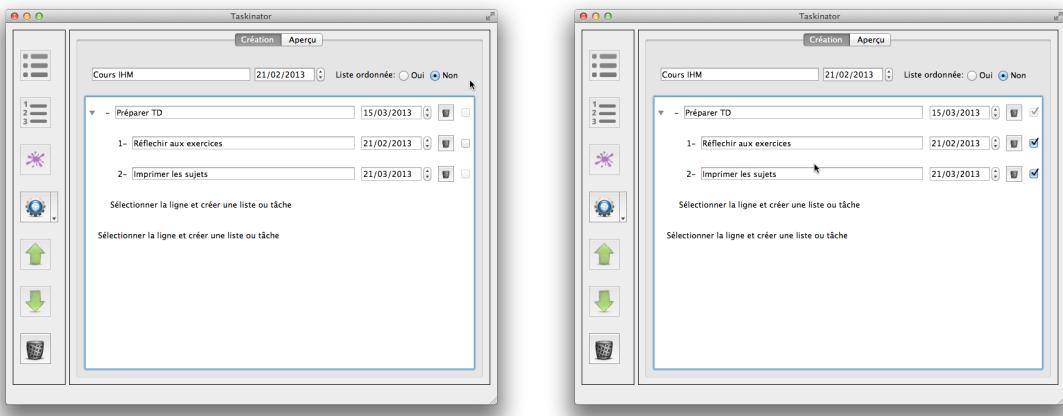


FIGURE 7 – Fonctionnement de l'activation/désactivation des cases à cocher

Nous aurons une information supplémentaire au début de chaque élément. Dans le cas d'une liste non ordonnées, ses éléments sont représentés avec un tiret devant les informations de la liste, tandis que les éléments des listes ordonnées sont représentés à l'aide de numéro.

Nous avons décidé pour plus de sécurité de griser tout menu, bouton inutilisable à un état de l'application afin que l'utilisateur n'exécute pas d'actions impossibles. Cela évitera qu'il se pose des questions s'il voit aucun changement effectué. De cette façon l'utilisateur sait à chaque instant que cette fonctionnalité est présente et n'a pas à chercher si une action particulière peut être effectuée ou pas.

5 Le modèle

Afin de réaliser cette application nous nous sommes basés sur le modèle suivant dont voici le diagramme de classes :

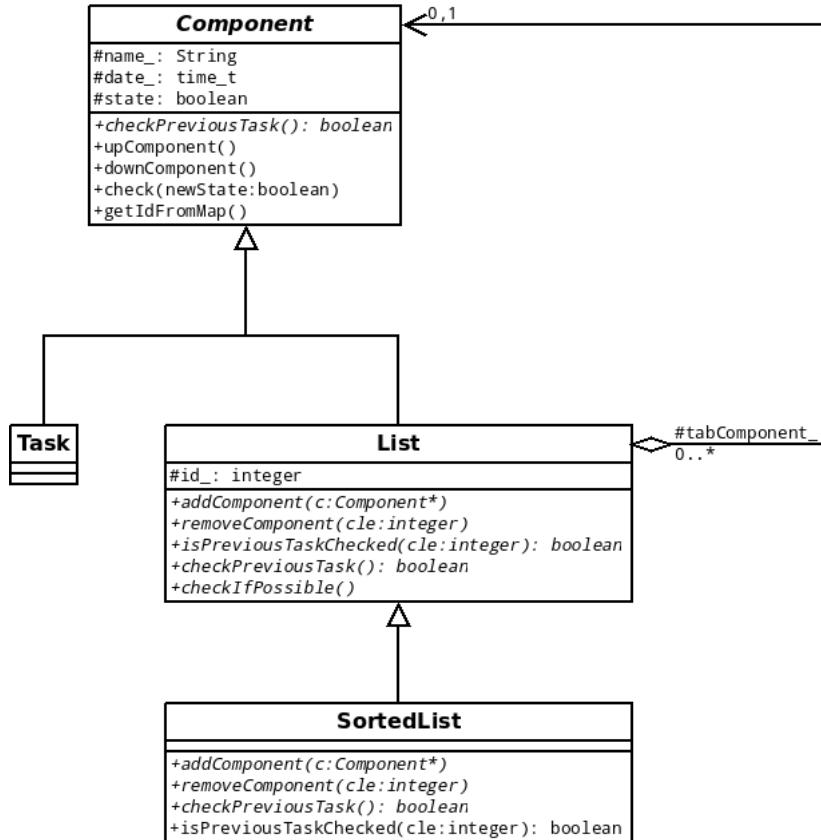


FIGURE 8 – Diagramme de classe du modèle

Le modèle de cette application est composé d'un pattern composite permettant d'implémenter une structure d'arbre et de composer les différents objets ensemble. Nous avons donc ici un *Component* qui peut être, soit une *Task* (une tâche), soit une *List* (une liste) qui elle-même peut ensuite être une *SortedList* (une liste ordonnée), celle-ci hérite de la classe *List*. Ce modèle permet donc comme expliqué ci-dessus de composer ces éléments et d'obtenir par exemple des listes de listes de tâches ... Nous pouvons donc gérer tous ces composants au sein de ce modèle et ainsi, ajouter un composant dans une liste, le supprimer, le déplacer d'un rang (dans les deux sens). Nous disposons aussi de toutes les fonctions permettant de déterminer si un composant est cochable ou non.

6 Limite de l'application

Notre application à ses propres limites. Elle ne permet pas d'éditer différentes listes en même temps (l'utilisateur peut créer/modifier une liste à la fois).

Elle ne gère pas non plus la cohérence des dates des composants. Effectivement dans le cas d'une liste ordonnée, il faudrait vérifier que la date d'une tâche soit inférieure à la date de la tâche suivante. Cette vérification devra se faire lors de la création mais aussi à chaque action effectuée.

Et ne peut pour le moment pas gérer un historique d'actions effectuées, ce qui aurait pu être très utile en cas d'erreur de manipulation (à la suite d'une suppression non souhaitée par exemple).

Il aurait été sympathique de proposer la création de la liste d'une part et de pouvoir cocher les tâches d'autre part, sur deux affichage différents. Que la modification d'un des deux affichages est un impact sur l'autre.

7 Conclusion générale

Ce projet nous a permis de réfléchir à la structure de l'IHM afin que celle-ci soit la plus intuitive possible pour l'utilisateur. Réfléchir à la position des boutons, le nom ou l'icône les représentant, comment l'ajout des composants se fera au sein de la liste, ... Et donc, d'orienter notre conception sur l'IHM, plus que sur le modèle, et ainsi expérimenter une autre approche de conception.