SAE 3.01 – DEVELOPPEMENT

-

LOGICIEL D'ORGANISATION DE TACHES PERSONNELLES

Rapport final

Alann MONNIER, Fabien TOUBHANS, Farid MARI

12/01/2024

Table des matières

1. INTRODUCTION

1.1 Contexte et objectifs du projet

Ce document final présente le processus complet de conception et de développement que nous avons suivi pour notre projet de fin de module, lequel avait pour but de créer un logiciel d'organisation de tâches personnelles. Dans le cadre de la SAE 3.01, notre équipe s'est attelée à la tâche complexe de concevoir un système répondant à divers besoins de gestion des tâches dans un contexte personnel.

Nous décrivons ici la création d'une application qui se mesure aux outils de gestion de projet, tels que Trello, tout en apportant des innovations en termes de fonctionnalités. Notre approche a mis un point d'honneur sur l'identification précise des besoins des utilisateurs et l'intégration de ceux-ci dans une conception fonctionnelle.

Ce rapport, détaille notre analyse, nos choix de conception, et les stratégies de développement que nous avons adoptées. Nous avons implémenté une architecture MVC en JavaFx, tout en suivant scrupuleusement les principes de conception que nous avons acquis, ainsi que ceux que nous avons découvert en cours de développement, afin de garantir la création d'un produit final répondant aux attentes établies.

En conclusion, ce document final reflète l'aboutissement de notre projet, et nous sommes fiers du résultat que nous avons obtenu.

Nous tenons à exprimer notre sincère gratitude envers toute l'équipe pédagogique pour son accompagnement tout au long de ce projet. Finalement, ce que nous avons pu atteindre est le fruit de notre collaboration et de notre détermination collective.

2. EVOLUTION DU PROJET PAR RAPPORT A L'ETUDE PREALABLE

Lors de la conception initiale, nous avions prévu d'utiliser plusieurs patrons de conception pour structurer notre application. Cependant, au fil de notre progression dans le projet, certaines de ces décisions ont été revues et adaptées en fonction des besoins et des contraintes.

1. Patron Active Record

Initialement, nous avions envisagé d'utiliser le patron Active Record pour gérer la sauvegarde de nos tâches, colonnes et lignes, en les stockant dans une base de données et en les récupérant à chaque redémarrage de l'application. Cependant, nous avons opté pour une approche de sérialisation de notre modèle de données (classe ModeleMenu), ce qui nous permet de restaurer facilement les données précédemment créées sans recourir à une base de données externe.

2. Patron Fabrique

Nous avions initialement prévu d'utiliser le patron Fabrique pour créer différents types de tâches, à savoir les tâches simples, les tâches mères et les tâches filles. Cependant, nous avons simplifié notre modèle de données en n'introduisant pas de distinction stricte entre ces types de tâches, ce qui a rendu le patron Fabrique inutile dans notre contexte.

3. Patron Stratégie et Patron Itérateur

Nous avions envisagé d'appliquer le patron Stratégie et le patron Itérateur pour gérer les différents types de tri de nos colonnes ou lignes. Finalement, nous avons opté pour une approche plus directe en implémentant quelques méthodes dans notre classe ModeleMenu pour effectuer les tris, éliminant ainsi le besoin de ces patrons.

4. Patron Stratégie pour le changement de Vue

Nous avions également envisagé d'utiliser le patron Stratégie pour gérer les changements de Vue, mais en raison de la complexité de notre architecture JavaFX actuelle, nous n'avons pas pu implémenter ce patron de manière efficace dans le temps imparti.

5. Patron Composite

Dans nos premières itérations, nous avions utilisé le patron Composite pour gérer les dépendances et les sous-tâches. Cependant, au fur et à mesure de l'évolution du projet, nous avons opté pour d'autres solutions, notamment la gestion des dépendances et l'utilisation d'une structure de données de type Map, ce qui a rendu le patron Composite superflu.

En plus de ces modifications architecturales, nous avons ajouté plusieurs fonctionnalités importantes. Nous avons implémenté la sauvegarde des données grâce à la sérialisation de notre modèle, ce qui permet de restaurer les données lors du redémarrage de l'application. De plus, nous avons introduit la possibilité de déplacer les colonnes par le biais du glisser-déposer (drag and drop). Enfin, nous avons mis en place des vérifications lors de l'ajout de sous-tâches ou de dépendances pour nous assurer que les dates entrées sont cohérentes par rapport à la tâche parente, améliorant ainsi la qualité globale de l'application.

3. LISTE DES FONCTIONNALITES

Création de tâches :	Cette fonctionnalité permet aux utilisateurs de créer des tâches.
Créer de nouvelles colonnes (3 de bases)	Les utilisateurs pourront créer de nouvelles colonnes en plus des trois colonnes de base fournies. Ces colonnes peuvent être personnalisées pour refléter différents stades ou aspects du workflow des tâches, telles que 'À faire', 'En cours', et 'Terminée', offrant ainsi une flexibilité dans l'organisation des tâches.
Supprimer une tâche	Une option pour supprimer les tâches inutiles ou terminées permet de maintenir l'espace de travail clair et organisé. Les tâches peuvent être supprimées individuellement ou en groupe, en fonction des besoins de l'utilisateur.
Archiver une tâche	Les tâches complétées peuvent être archivées pour conserver un historique des travaux accomplis sans encombrer l'espace de travail actif. L'archivage contribue à une meilleure visibilité des tâches en cours et aide à suivre les progrès tout en conservant les données pour des références futures. On peut bien sûr désarchiver ces tâches. Elles apparaissent dans un onglet prédisposé, appelé Archive
Dépendances entre tâches	Il est possible d'établir des liens de dépendance entre les tâches. Cette fonctionnalité est indispensable pour la génération du Gantt.
Visualisation en liste et sous-listes	Les utilisateurs peuvent visualiser les tâches sous forme de listes imbriquées, ce qui est particulièrement utile pour les projets complexes avec de multiples sous-tâches. Cette structure hiérarchique aide à comprendre la relation entre les tâches et les différentes étapes nécessaires à leur achèvement.
Génération de diagrammes de Gantt	Cette fonctionnalité avancée permet aux utilisateurs de sélectionner des tâches spécifiques et de générer un diagramme de Gantt, facilitant une visualisation graphique de la chronologie et de la durée de chaque tâche. Les utilisateurs peuvent ainsi anticiper les interactions entre les tâches, gérer les délais et optimiser la planification globale du projet.

Tri des tâches (alphabétique)	Les utilisateurs ont la possibilité de trier leurs tâches par ordre alphabétique, ce qui facilitera la recherche et la navigation.
Tri des tâches selon l'urgence (3 niveaux)	Cette fonctionnalité permet de trier les tâches en fonction de leur niveau d'urgence, en proposant trois niveaux différents. Cela aidera à hiérarchiser les tâches en fonction de leur importance et de leur priorité.
Tri des tâches selon la date de création	Les utilisateurs pourront trier leurs tâches en fonction de leur date de création, ce qui permettra de suivre l'ordre chronologique des tâches et de mieux gérer les nouvelles tâches par rapport aux anciennes.
Déplacer une tâche via le menu	Donne la possibilité aux utilisateurs de déplacer une tâche d'une colonne à une autre en utilisant l'option déplacer tâche dans le menu déroulant, offrant ainsi une manière pratique de réorganiser les tâches.
Drag and drop	Les utilisateurs ont la possibilité de déplacer les tâches d'une colonne à une autre ou de réorganiser l'ordre des tâches au sein d'une colonne en utilisant la fonction de glisser-déposer dans la visualisation du tableau.

4. MODELISATION UML

4.1 Diagramme de classe

voir dossier → Diagramme_de_classe.

4.2 Graphe de Scene

voir dossier → Graphe_de_Scene.

5. PATRONS DE CONCEPTION

Dans le cadre de notre projet Trello, nous avons mis en place plusieurs patrons de conception et d'architecture pour garantir une structure robuste et extensible de notre application. Voici les principaux patrons que nous avons utilisés :

1. **Patron Singleton** :

Nous avons adopté le patron Singleton dans la gestion de notre classe "Archive" pour nous assurer qu'une seule instance de cette classe puisse exister. Cette décision repose sur le fait que la gestion des tâches archivées doit rester centralisée et que l'existence de plusieurs instances pourrait entraîner des incohérences dans les données.

2. **Patron Observateur (Modèle-Vue-Contrôleur - MVC) :**

Pour garantir un affichage dynamique et une architecture bien structurée, nous avons opté pour le patron Observateur et mis en place l'architecture Modèle-Vue-Contrôleur (MVC). Cette approche nous a permis de séparer clairement les responsabilités au sein de notre application. Le Modèle, représenté par la classe "ModeleMenu", contient toutes les données du Trello, gère les interactions utilisateur, et notifie les vues en cas de modifications. La Vue, quant à elle, est responsable de l'affichage des données et peut être adaptée pour différents modes tels que les listes, les colonnes, l'archivage ou le diagramme de Gantt. Les Contrôleurs interceptent les actions de l'utilisateur et mettent à jour le Modèle avec de nouvelles données, ce qui est ensuite reflété dans l'affichage.

Nous avions également envisagé d'utiliser le patron Stratégie pour gérer les différentes méthodes d'affichage et de tri des tâches, mais en raison de contraintes de temps et des modifications apportées au projet, nous n'avons pas pu le mettre en œuvre. Cependant, le choix des patrons Singleton et Observateur/MVC a grandement contribué à la solidité de notre application et à la maintenabilité de son code.

6. REPARTITION DU TRAVAIL DANS L'EQUIPE

Alann →

- Implémentation du Modele
- Création des gestions des dépendances
- Implémentation du Modele
- Conception des Vue liées aux colonnes en JavaFX
- Sauvegarde
- Contrôleurs
- Drag and drop des colonnes
- Architecture MVC
- Diagramme de GANTT
- Déploiement des tests
- Diagramme de classe

Fabien →

- Implémentation du Modele
- Gestion des tris
- Contrôleurs
- Déplacer / Supprimer tâche
- Archivage
- Sauvegarde
- Architecture MVC
- Diagramme de GANTT
- Diagramme de classe

Farid →

- Implémentation du Modele
- Tâche
- Contrôleurs
- Implémentations des tâches sous forme de listes et sous listes
- Gestion des alertes – logique de refus
- Architecture MVC
- Diagramme de GANTT
- Style de l'application (Colonne, Listes, Taches, Sous taches, Dépendance, Details)
- Diagramme de classe

7. ELEMENTS ORIGINAUX DONT NOUS SOMMES FIERS

Alann :

« *La fonctionnalité qui m'a rendu le plus fier est la fonctionnalité du drag and drop. Pouvoir déplacer une tache dans une autre colonne juste en la déplaçant est très intéressant et évite de devoir à passer par un menu puis une autre fenêtre et indiqué ou l'on veut déplacer la colonne. J'ai bien aimé cette fonctionnalité car je ne la connaissais pas et j'ai appris de nouvelle chose. De plus j'ai trouvé la fonctionnalité très logique et donc facile à implémenter.*»

Fabien :

« *Ce qui m'a rendu le plus fier dans notre projet Trello, c'est notre système de sauvegarde via sérialisation. Pouvoir créer un mécanisme qui permet de retrouver les données précédemment créées à chaque lancement de l'application a été une réalisation vraiment satisfaisante. Cela ajoute une dimension de convivialité et de continuité à l'expérience. J'ai passé beaucoup de temps à le mettre en place, à assurer la stabilité et la fiabilité de cette fonctionnalité.* »

Farid :

« *Ce qui m'a rendu le plus fier, c'est l'implémentation de la vue en liste et sous-liste. J'ai passé beaucoup de temps à élaborer une méthode récursive ingénieuse pour gérer les sous-tâches de manière visuellement intuitive. Voir les sous-tâches se déployer dans une liste hiérarchique lorsque l'utilisateur interagit avec notre application a été une réalisation vraiment gratifiante. Cela offre une manière claire et logique d'organiser les tâches complexes, en permettant aux utilisateurs de comprendre rapidement les relations entre les tâches principales et les sous-tâches associées. Je suis fier d'avoir pu apporter cette fonctionnalité.* »