SAE 3.01 Développement d'applications

Pour cette SAE nous avons dû réaliser un logiciel d'organisation de tâches. Le logiciel se base sur l'utilisation de tâches qui peuvent comprendre un nombre indéfini de sous-tâches qui peuvent elles même contenir un nombre indéfini de sous-tâches et ainsi de suite.

Dans le logiciel les fonctionnalités disponible sont les suivantes :

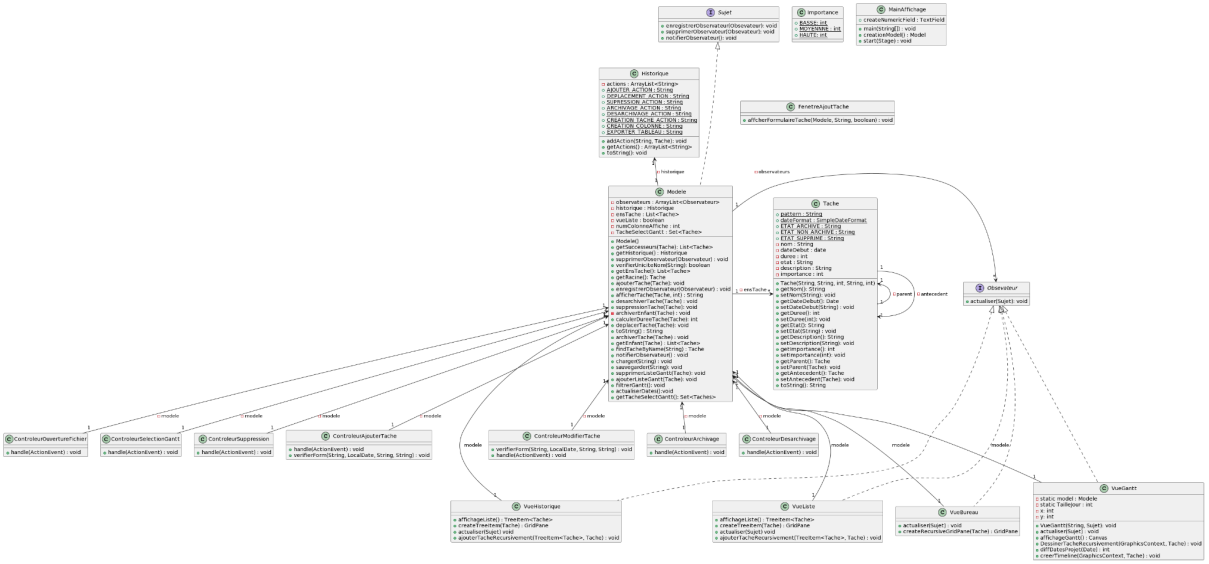
- créer une tâche
- déplacer une tâche dans une autre tâche
- modifier une tâche
- archiver une tâche (et donc ses sous-tâches)
- supprimer une tâche (et donc ses sous-tâches)
- enregistrer le projet dans un fichier
- ouvrir un projet enregistré sous forme d'un fichier
- Alterner entre différente vues :
    - vue en liste
    - vue bureau
    - vue d'un diagramme de gantt (où l'on peut choisir les tâches à afficher)
    - vue d'un historique des actions réalisées

Répartition du travail entre les membres du groupe :

Itération	DECHEZLEPRETRE Luc	DESPAQUIS Liam	FONTANEZ Antoine	SLIMANI Robin
1	Mise en place de la récursivité des tâches et affichage	Gestion des tâches et évolution des tâches parentes	Affichage en JavaFX	Tests unitaires, gestion de l'historique, quelques méthodes du modèle
2	Vue liste, Drag and Drop dans la vue liste	Création de la vue Bureau récursivement	Formulaire d'ajout de tâches en JavaFX	Mise à jour du diagramme de classe, vue Bureau

	Mise à jour du diagramme de classe, vue Bureau			
3	Implémentation d'un TabPane, aide pour modification et Drag and Drop dans la vue Bureau	Implémentation du Drag and Drop dans la vue Bureau	Archivage des tâches, affichage dans l'onglet dédié	Modification d'une tâche, mise à jour des diagrammes de classe et de séquence
4	Création du Gantt, implémentation du scrollPane	Création du Gantt en majorité, vérification des antécédents	Désarchivage, suppression des tâches, ajout de valeurs par défaut	Mise à jour du diagramme de classe, tests, diagrammes de séquence
5	Mise en place de la sauvegarde et du chargement de projet trellol en fichier.	correction du Gantt et d'autres erreurs dans le projet	Mise en place d'un meilleur interface graphique à l'aide de CSS	Mise en place du scénario proposé et début du support de la soutenance.

Diagramme de classe final :



Présentation d'un élément original :

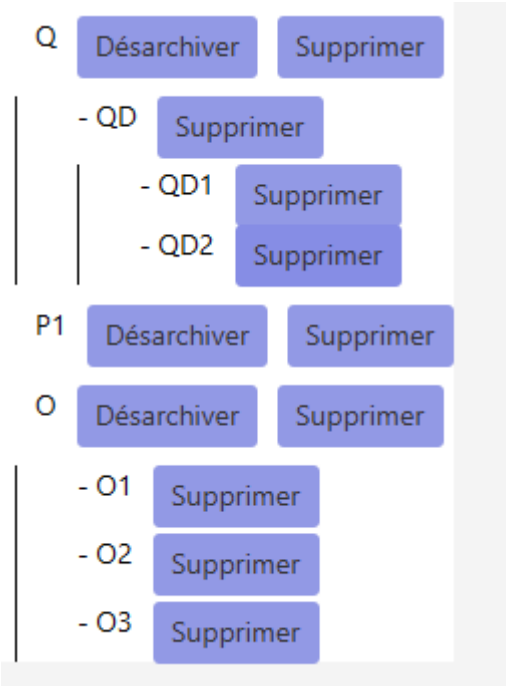
Antoine :

En ce qui me concerne, la fonctionnalité dont je suis le plus fier est la gestion de l'archive. Lorsque l'on est sur la fenêtre de modification d'une tâche on peut archiver une tâche, ce qui va la faire complètement disparaître des affichages Bureau, Liste et Gantt. La méthode d'archivage du modèle met la tâche courante en état archivé puis lance une méthode récursive recherchant toutes ses sous-tâches et les archive.

```
/**
 * Méthode permettant l'achivage dans le modèle de la tâche passé en paramètre
 * @param tache à archiver
 */
4 usages   RobinSlimani <robinslimani> +2
public void archiverTache(Tache tache) {
    tache.setEtat(Tache.ETAT_ARCHIVE);
    this.archiverEnfants(tache);
    this.getHistorique().addAction(Historique.ARCHIVAGE_ACTION, tache.getNom());
    this.notifierObservateurs();
}
```

```java
/**
 * Méthode recursive permettant l'achivage des enfants d'une tâche
 * @param tache parent des enfants à archiver
 */
2 usages  Akip2
private void archiverEnfants(Tache tache){
    ArrayList<Tache> enfants= (ArrayList)this.getEnfant(tache);
    for(Tache t : enfants){
        t.setEtat(Tache.ETAT_PARENT_ARCHIVE);
        this.archiverEnfants(t);
    }
}
```

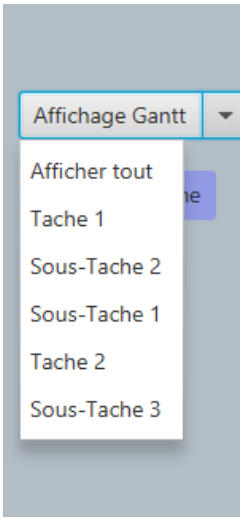
On peut accéder à toutes les tâches archivées dans l'archive. On voit les sous-tâches d'une tâche archivée sous forme d'arborescence. On peut supprimer une sous tâches archivées mais on peut seulement désarchiver la tâche parent et en la desarchivant toutes ses sous taches sont désarchivées avec.



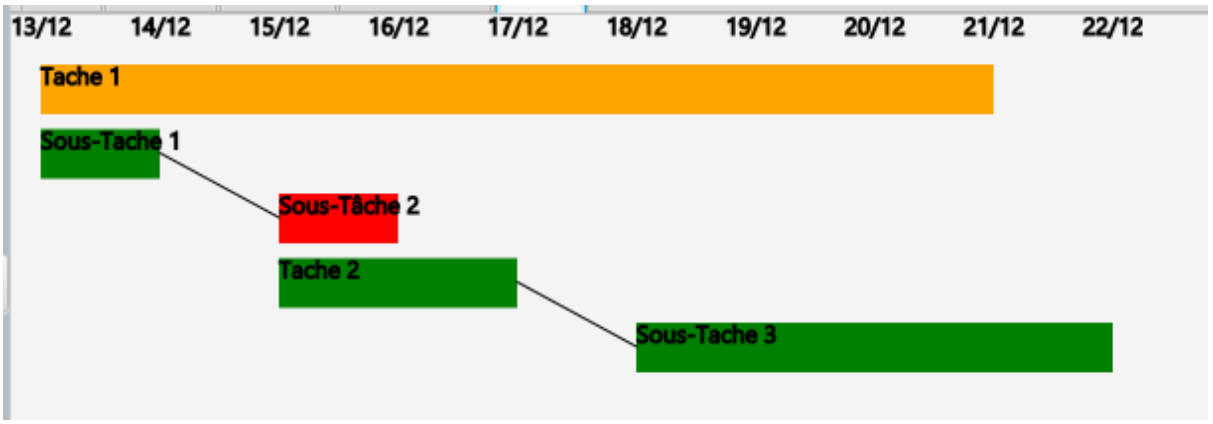
La méthode désarchiver du modèle est récursive et recherche tous les enfants de la tâche à désarchiver pour leur redonner l'état initial. Cependant, le modèle ne notifie pas les

observateurs à chaque appel, elle ne le fait qu'au premier afin d'optimiser la méthode et de ne lancer l'actualisation seulement pour l'affichage final.

Liam : La partie de mon travail dont je suis le plus fier est la génération du diagramme de Gantt : dessiné sur un canvas, le diagramme de Gantt est généré en fonction des tâches sélectionné par l'utilisateur via un menu



Le Gantt a été assez important dans ce projet car il a permis de souligner les problèmes dans notre gestion des dates. Le Gantt a été généré récursivement pour qu'un antécédent soit à côté de ses successeurs et les tâches ont été préalablement trié pour être affiché par ordre chronologique. Bien que le diagramme n'affiche pas clairement les lien de parenté, il affiche les liens antécédents/successeurs avec une ligne reliant les deux tâches



Robin :

- L'élément le plus original que j'ai pû faire est l'historique des actions réalisées. Cette fonction utilise une classe **Historique** ayant une liste d'actions en **String**.

```
public class Historique {
    /**
     * Attribut liste représentant la totalité des actions
     */
    8 usages
    private LinkedHashMap<String, String> actions;

    /**
     * Attribut String représentant l'action de modification d'une tâche
     */
    1 usage
    public static String MODIFICATION_ACTION = "La tâche {objet} a été modifiée";
    /**
     * Attribut String représentant l'action de déplacement d'une tâche
     */
    1 usage
    public static String DEPLACEMENT_ACTION = "La tâche {objet} a été déplacée";
    /**
     * Attribut String représentant l'action de suppression d'une tâche
```

- Cette classe possède des actions prédéfinies, utilisées dans le modèle lors des actions faites.

```
     */
    2 usages   RobinSlimani <robinslimani> +1
    private VBox affichageHistorique() {

        VBox vb = new VBox( v: 10);
        vb.getStyleClass().add("boiteTexte");
        vb.setPadding(new Insets( v: 10, v1: 10, v2: 10, v3: 20));
        for (String action : model.getHistorique().getActions()) {
            Label label = new Label(action);
            vb.getChildren().add( index: 0,label);
        }
        return vb;

    }
```

- L'ajout d'une action se fait grâce à la méthode **addAction(Action prédéfinie, la tâche concernée)**.

```
6 usages   despaqui9u
public void addAction(String action,String objet){
    this.actions.add(action.replace( target: "{objet}",objet));
}
```

Luc :
Pour ma part, l'élément dont je suis le plus fier est la création d'un objet **TreeView** pour la vue liste. Le **TreeView** est créé de façon récursive à l'aide de deux méthodes :

- La première est seulement une méthode qui crée un **TreeItem** possédant une tâche

```
private TreeItem<Tache> createTreeItem(Tache tache) {
    TreeItem<Tache> treeItem = new TreeItem<>(tache);

    return treeItem;
}
```

- La seconde prend en paramètre en paramètre le **TreeItem** auquel ajouter les enfants et la tâche, tâche justement prise en paramètre. Donc pour construire la suite du **TreeView** la méthode récupère à partir du modèle la liste des enfants de la tâche, construit un **TreeItem** pour l'enfant puis rappelle la méthode pour ses enfants. Jusqu'à ce qu'une tâche n'ai pas d'enfants.

```
private void ajouterTacheRecursivement(TreeItem<Tache> parentItem, Tache tache){
    if(this.model.getEnfant(tache).size() == 0){
        return;
    }
    for(Tache enfant : this.model.getEnfant(tache)){
        if(enfant.getEtat()!=Tache.ETAT_ARCHIVE) {
            // Crée un nouvel élément de TreeItem pour la tâche
            TreeItem<Tache> childItem = createTreeItem(enfant);

            // Ajoute l'élément enfant à l'élément parent
            parentItem.getChildren().add(childItem);

            // Appelle récursivement la fonction pour ajouter des sous-tâches à cet élément enfant
            ajouterTacheRecursivement(childItem, enfant);
        }
    }
}
```

- Et donc la création du **TreeView** se fait comme suit en créant une **TreeView** depuis la racine du modèle :

```
// Crée le TreeView avec l'élément racine
TreeItem<Tache> racine = createTreeItem(this.model.getRacine());
this.ajouterTacheRecursivement(racine, this.model.getRacine());
//Création du TreeView à partir du TreeItem racine
TreeView<Tache> treeView = new TreeView<>(racine);
```

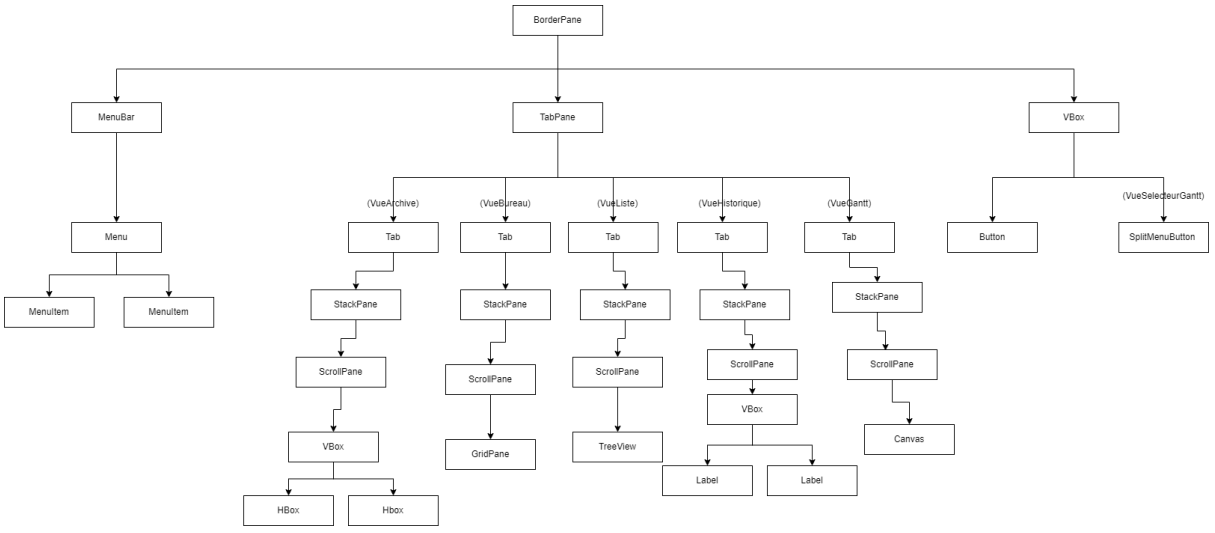
Comparaison avec l'étude préalable :

Dans l'étude préalable nous sommes partis sur un logiciel différent qui utilisait une base de données et gérait en plus des utilisateurs et leur connexion. Les tâches pouvaient avoir des personnes assignés à leur réalisation.
De plus, la conception des tâches et de leurs sous-tâches à été mal exécutée. En effet on utilisait une classe ListeTache permettant de représenter des colonnes avec des tâches. Mais finalement lors de la conception finale nous avons préféré faire cela à l'aide du patron de conception composite, qui permet de gérer les sous tâches.

Patron de conception et d'architecture :

Dans notre application l'architecture est celle du modèle de conception MVC. Il nous est très utile pour la gestion des données par l'utilisateur.
Et les tâches ont été implémentées en utilisant le patron composite. Ce patron a été implémenté en JAVA de la manière suivante : chaque objet Tache possède un attribut Tache nommé parent, ce parent possède donc des enfants qui sont des tâches qui le composent et les enfants peuvent devenir parent et ainsi de suite. Donc dans notre modèle la méthode la plus importante est certainement celle qui permet de récupérer tous les enfants d'une tâche. Ce qui nous permet une utilisation importante de la récursivité dans notre code.

Graphe de scène de l'application :



L'interface de notre application est organisé de la manière suivante :

- La base de l'interface est un BorderPane qui contient :
    - au centre un TabPane qui permet de switcher entre les différentes vues
    - au dessus une MenuBar qui permet l'ouvreture/l'enregistrement de projet
    - et enfin à gauche un Button qui permet d'ajouter une tâche et un menu déroulant qui permet de choisir les tâches à afficher dans le Gantt.

Conclusion :

Pour conclure nous sommes assez fiers du rendu final de notre application et son développement a été très enrichissant que ce soit du point de vue de la conception ou même de l'implémentation. Ce projet nous a permis de mettre en pratique des concepts vu dans différentes ressources ce qui nous fait comprendre l'importance de ces dernières.