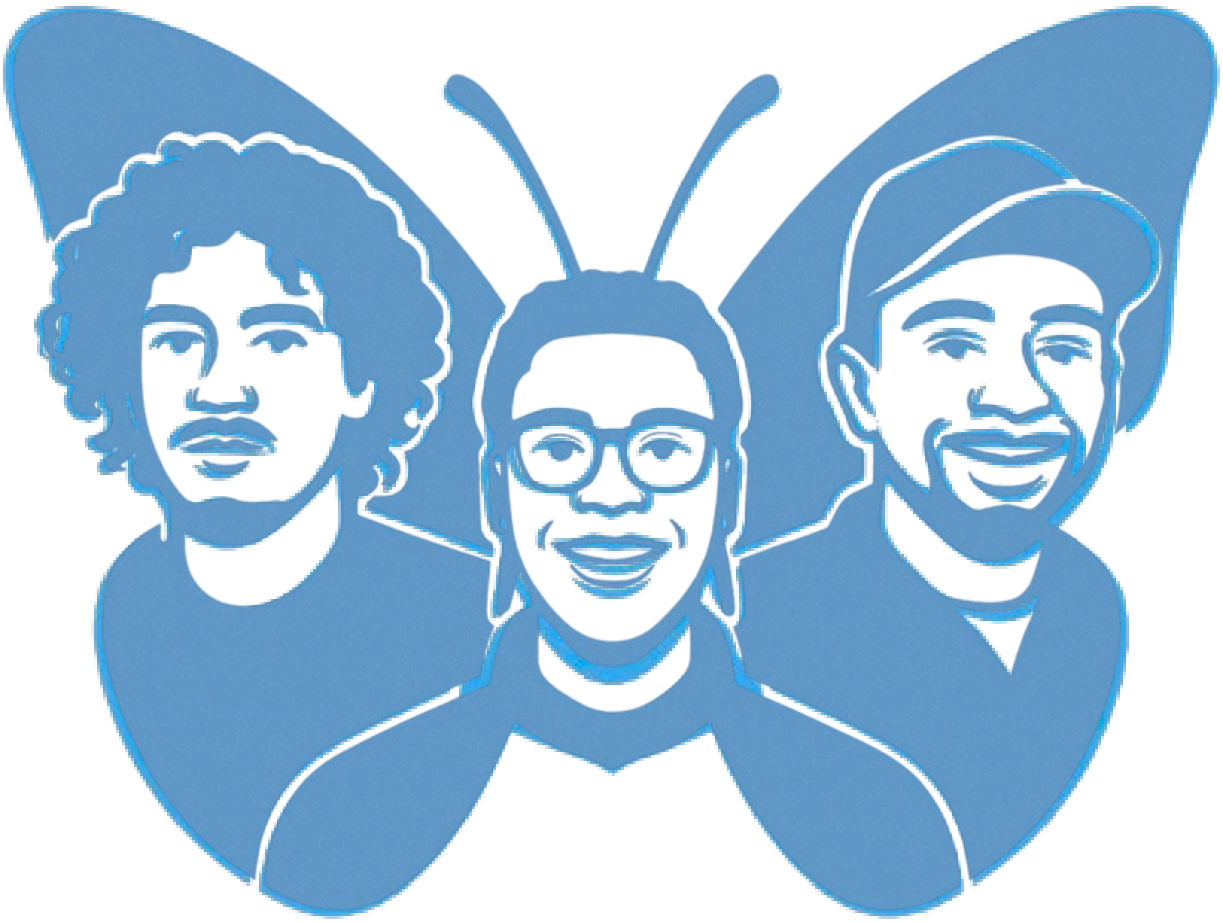
Rapport de projet

SAÉ 3.1 FI - Papillon

2025 - 2026

Aylane SEHL **Séri-khane YOLOU** **Jenson VAL**



UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL
IUT SENART FONTAINEBLEAU

TABLE DE MATIÈRES

I. INTRODUCTION

1.Présentation du projet et contexte

Le projet Papillon s'inscrit dans le cadre de la SAÉ 3.1 du BUT Informatique à l'IUT de Sénart-Fontainebleau.

L'objectif de cette SAÉ est de concevoir une application informatique complète, en appliquant les notions de programmation orientée objet, de gestion de projet et de conception d'interfaces graphiques apprises au cours du semestre.

Notre groupe a développer une application Java nommée "Papillon", qui prend la forme d'une To-Do List interactive. Elle permet à un utilisateur de créer, modifier et supprimer des rappels tout en les classant selon leur thème et leur niveau de priorité.

Nous avons eu des retours indiquant que le nom "Papillon" évoquait le comportement de l'insecte attiré par la lumière, se posant sur l'écran. Cette interprétation symbolique illustre bien notre volonté de concevoir une application vers laquelle l'utilisateur revient naturellement, guidé par sa clarté et sa simplicité d'utilisation.

Le projet s'inscrit dans une démarche d'autonomie et de réflexion. Nous avons cherché à concevoir une application à la fois fonctionnelle, intuitive et agréable à utiliser, en accordant une attention particulière à sa structure, sa logique et son ergonomie.

Ce projet a également constitué une étape importante dans notre apprentissage du développement Java, notamment en matière de gestion d'événements, de structuration du code et de manipulation d'interfaces graphiques.

Le travail a été mené en équipe, à l'aide de la plateforme Gitea, afin d'assurer le suivi du code, la répartition des tâches et la gestion des versions du projet.

La suite de ce rapport présentera les fonctionnalités principales de l'application, sa structure technique, ainsi que les choix de conception réalisés tout au long du développement.

I. INTRODUCTION

2.Objectifs du projet

Le principal objectif du projet Papillon est de développer une application simple, intuitive et efficace permettant à un utilisateur de gérer ses rappels et ses tâches quotidiennes. L'enjeu est de proposer un outil clair, visuel et agréable à utiliser, tout en assurant une bonne organisation et une navigation fluide entre les différentes fonctionnalités.

Nous avons voulu créer une application qui facilite la gestion du temps et des priorités. Chaque rappel peut être classé selon un thème (par exemple rouge, vert,bleu...) et un rang de priorité afin d'aider l'utilisateur à visualiser les tâches les plus urgentes. Cette approche permet d'obtenir une vue d'ensemble structurée et hiérarchisée de toutes les activités à réaliser.

Un autre objectif important du projet était de concevoir une interface ergonomique et accessible. L'utilisateur devait pouvoir ajouter, modifier ou supprimer un rappel en quelques clics, sans être confronté à des menus complexes ou des manipulations techniques. L'interface graphique a donc été pensée pour offrir une expérience simple et fluide, adaptée aussi bien à un usage régulier qu'occasionnel.

Enfin, le projet Papillon a également servi de support à l'apprentissage de bonnes pratiques de développement. Nous avons accordé une attention particulière à la structuration du code, à la clarté de la logique interne et à la stabilité de l'application.

Le projet nous a permis d'approfondir notre maîtrise du langage Java et de l'environnement Swing, tout en découvrant l'importance du travail collaboratif et du suivi de version via Gitea.

En résumé, Papillon vise à concilier simplicité d'utilisation, organisation intuitive et qualité de développement, tout en renforçant les compétences techniques et méthodologiques des membres du groupe.

3.Technologies employées

Le projet Papillon a été développé en Java, un langage orienté objet largement utilisé pour les applications de bureau.

Ce choix s'explique par la volonté d'exploiter un langage à la fois robuste, portable et structuré, offrant une bonne gestion des interfaces graphiques grâce à la bibliothèque Swing.
Java a permis de concevoir une application à la fois stable et performante, tout en respectant les principes de modularité et de réutilisabilité du code.

Pour la partie interface graphique, nous avons utilisé Swing, une bibliothèque native de Java permettant de créer des fenêtres, des boutons, des champs de texte et d'autres composants interactifs.
Son principal avantage est la souplesse de personnalisation et la compatibilité multiplateforme, ce qui garantit un rendu similaire sur différents systèmes d'exploitation.
L'utilisation de Swing nous a permis de construire une interface intuitive et dynamique, adaptée à une application de gestion de rappels.

La base de données du projet repose sur une solution locale simple, permettant de stocker les informations relatives aux rappels : titre, contenu, thème et rang de priorité.
Cette approche facilite la manipulation des données et assure un accès rapide aux informations nécessaires sans dépendre d'un serveur externe.
L'interconnexion entre le code Java et la base de données s'effectue via des requêtes SQL gérées par le programme.

Pour la gestion de version et le travail collaboratif, nous avons utilisé la plateforme Grond (Git), mise à disposition par l'IUT de Fontainebleau.
Cet outil nous a permis de synchroniser nos fichiers, de suivre les modifications et de travailler simultanément sur différentes parties du projet.
Grâce à Grond (Git), chaque membre du groupe a pu apporter ses contributions tout en conservant une traçabilité complète du développement.

Enfin, pour la rédaction et la mise en page du rapport, nous avons utilisé Canva pour le design visuel et Word pour la mise en forme du contenu avant exportation en PDF. Ces outils nous ont permis de produire un document soigné, lisible et professionnel, en cohérence avec l'identité visuelle du projet Papillon.

3.Technologies employées

Voici un petit schéma illustrant le descriptif cité précédemment :



Pour versionner les codes et collaborer en équipe efficacement via Git tout au long du développement



Pour rédiger, compiler et exécuter le code source de l'application, tout en bénéficiant d'un environnement de développement clair et efficace.



Pour gérer la base de données locale de l'application, stockant les informations liées aux rappels (titre, contenu, rang, thème).




Pour rédiger, mettre en page et illustrer le rapport de projet, en assurant une présentation claire et professionnelle du travail réalisé.

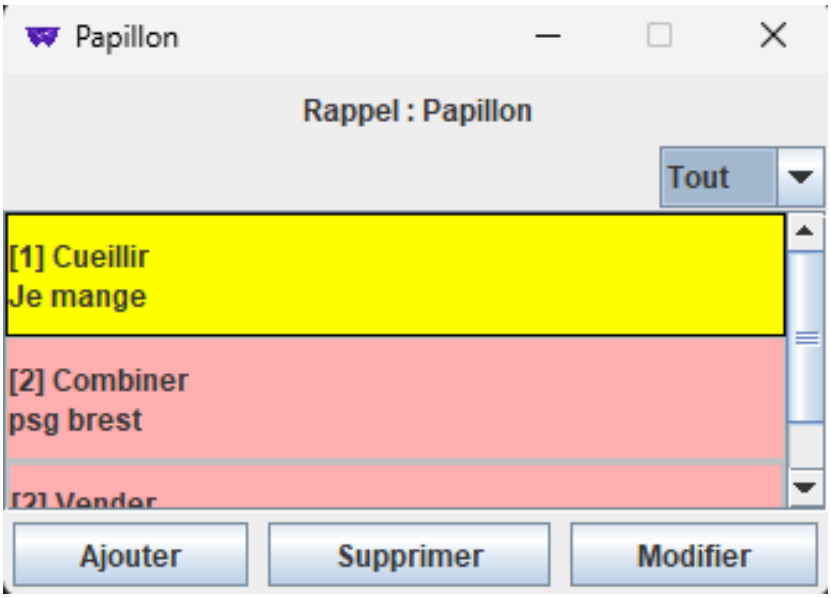


Afin de modéliser les différentes structures du projet, notamment le diagramme de classes et les schémas de conception, nous avons principalement utilisé les outils suivants.

II. PRÉSENTATION GÉNÉRALE ET FONCTIONNEMENT

1.Fonctionnalités principales

Implémentation des fonctionnalités basiques (CRUD)



- **C**reate (Créer/Ajouter un rappel)

Lorsque l'utilisateur clique sur le bouton ajouter une page s'ouvre lui permettant d'entrée les informations (Titre, Contenu, Rang, Thème) de son rappel et de le créer

- **R**ead (Lire un rappel)

Lorsque l'utilisateur clique droit et appuie sur le bouton ouvrir ou double-clique sur un rappel une page s'ouvre lui permettant de voir les informations (Titre, Contenu, Rang, Thème) de son rappel.

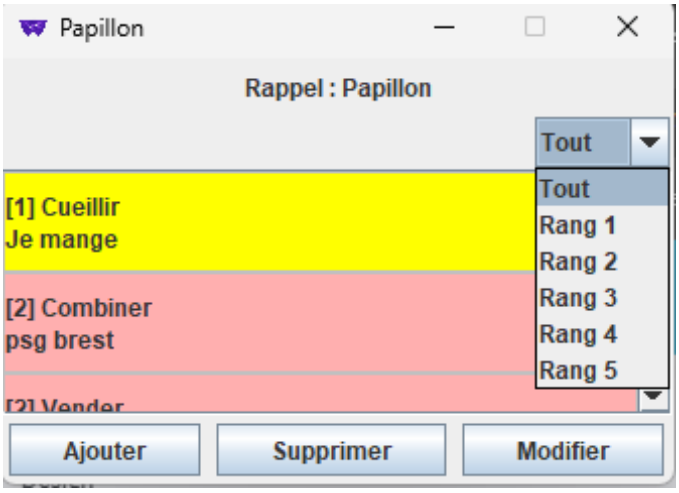
- **U**pdate (Modifier un rappel)

Lorsque l'utilisateur clique sur le bouton modifier, il doit d'abord sélectionner uniquement un rappel et une page s'ouvre lui permettant de modifier les informations (Titre, Contenu, Rang, Thème) de son rappel.

- **D**elete (Supprimer un rappel)

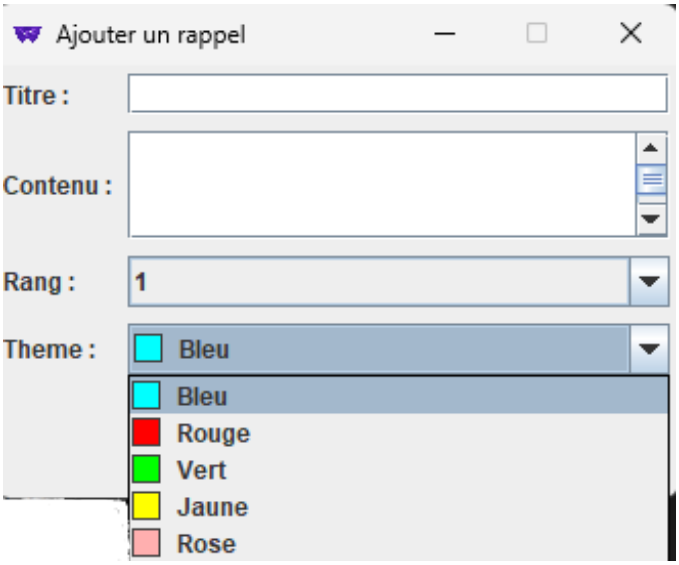
Lorsque l'utilisateur clique sur le bouton supprimer, il doit sélectionner un ou plusieurs rappels qui seront supprimer.

2.Fonctionnalités complémentaires



- **T**rier

L'utilisateur clique sur la liste déroulante et sélectionne une option.
"Tout" : permet d'afficher tout les rappels
"Rang N" : permet d'afficher uniquement les rappels de rang N.



- **T**hème

L'utilisateur clique sur la liste déroulante lors de la modification ou l'ajout d'un rappel et sélectionne une option.
Il a le choix entre 5 couleurs, elles serviront de couleur pour le fond du rappel.

III.STRUCTURE DU PROGRAMME

1.Structure et logique du code

Lors de la conception de notre application Papillon, nous avons initialement suivi une approche inspirée du modèle MVC (Modèle – Vue – Contrôleur) afin d'organiser clairement notre code.
L'objectif était de séparer les responsabilités :

- le modèle pour la gestion des données et de la base,
- la vue pour l'affichage et l'interface graphique,
- le contrôleur pour coordonner les interactions entre les deux.

Cependant, cette approche, bien que logique, impliquait de nombreuses importations de packages entre les différents répertoires (par exemple utiliser une classe du dossier modèle dans une classe du dossier vue). Cela rendait la compilation et la maintenance plus complexes pour un projet de taille modeste.

Après plusieurs essais sur différentes branches Git (modèle-test, contrôleur-test, etc.), nous avons décidé de fusionner toutes les classes dans un seul package commun :
package fr.iutfbleau.papillon;
Cette simplification respecte les conseils de notre enseignant, puisque notre application ne comporte pas un grand nombre de fichiers.
Ainsi, la structure est plus légère et lisible, tout en gardant une logique de séparation fonctionnelle à travers le nom et le rôle de chaque classe.

2. Justification du choix architectural

Le choix d'un modèle orienté objet structuré mais simplifié découle d'un souci d'efficacité. Notre application manipule différents types d'objets (utilisateurs, rappels, base de données, interface graphique) qui interagissent entre eux de manière claire.

Nous avons privilégié une architecture centrée sur la classe GestionRappel, qui fait le lien entre la couche de données (RappelBD, UtilisateurBD) et la partie interface (Main).

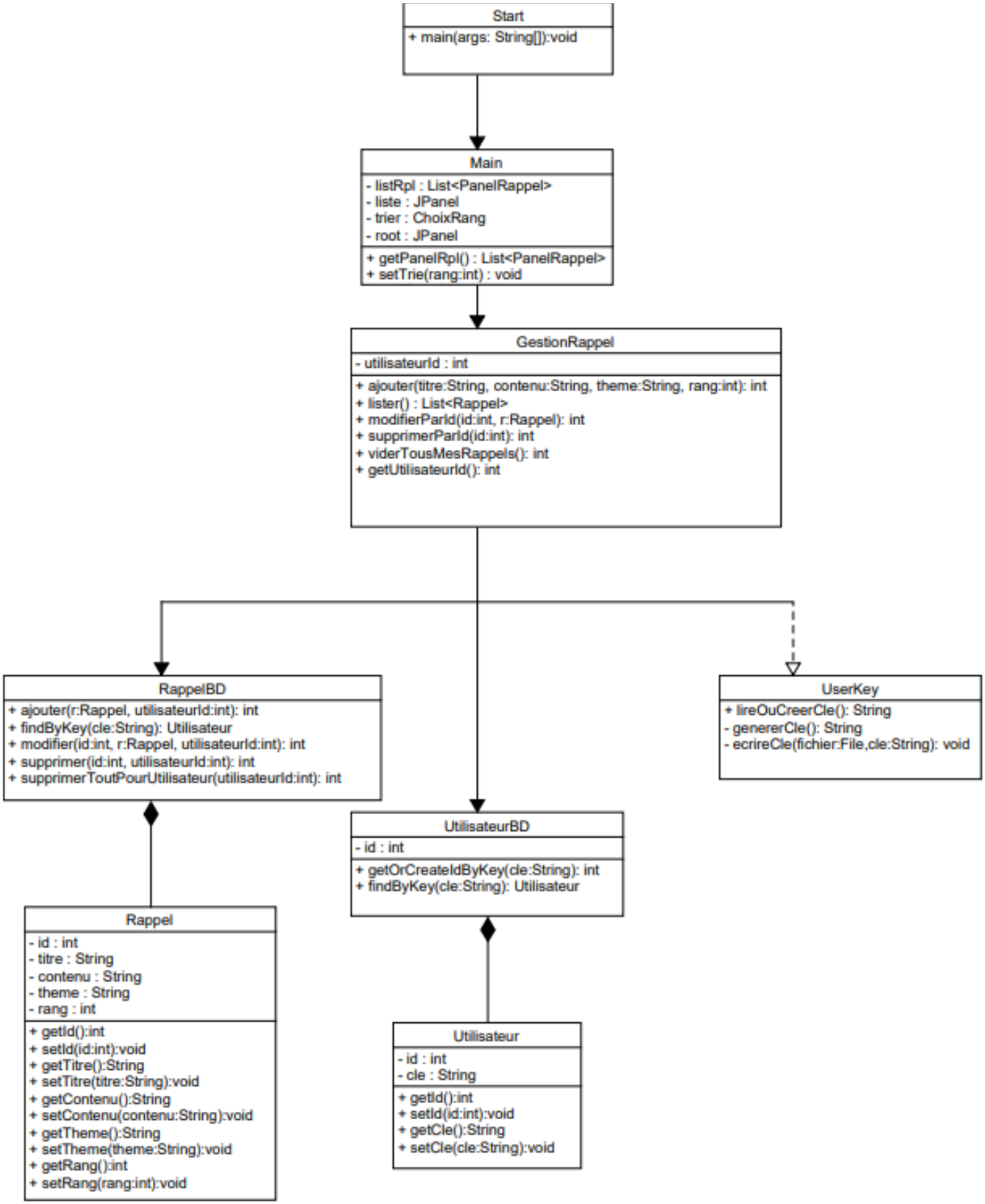
Ce choix garantit :

- une bonne lisibilité du code,
- une responsabilité claire pour chaque classe,
- et une facilité d'évolution en cas d'ajout de nouvelles fonctionnalités (par exemple la gestion de plusieurs utilisateurs ou la synchronisation).

En résumé, même sans structure MVC stricte, notre architecture conserve ses principes fondamentaux : séparation des rôles, modularité et cohérence.

III.STRUCTURE DU PROGRAMME

3.1Diagramme de classe simplifié



III.STRUCTURE DU PROGRAMME

3.2Diagramme de classe simplifié

Le diagramme de classes ci-dessous illustre la structure globale du programme et les principales relations entre les composants :

- Utilisateur : représente un utilisateur unique, identifié par un id et une clé.
- Il peut créer plusieurs Rappels (relation 1 ↔ 0..*).
- Rappel : contient les informations principales d'un rappel (titre, contenu, thème, rang).
- Ces objets sont manipulés par RappelBD pour être enregistrés ou modifiés dans la base.
- RappelBD : gère les opérations CRUD (ajout, suppression, modification, lecture) sur les rappels associés à un utilisateur.
- UtilisateurBD : permet de retrouver ou de créer un utilisateur à partir de sa clé unique.
- UserKey : gère la génération et la lecture d'une clé locale identifiant l'utilisateur.
- GestionRappel : fait le lien entre l'interface et la base de données.
- Elle orchestre les interactions entre Main, RappelBD et UtilisateurBD.
- Main : correspond à la fenêtre principale de l'application.
- Elle appelle GestionRappel pour afficher, trier ou mettre à jour les rappels selon les actions de l'utilisateur.

Ces relations peuvent se résumer ainsi :

Relation	Description
Utilisateur (1) ↔ Rappel (0..*)	Un utilisateur peut créer plusieurs rappels, mais chaque rappel
GestionRappel → RappelBD / UtilisateurBD / UserKey	GestionRappel coordonne les interactions entre les classes métiers
RappelBD ↔ Rappel	RappelBD manipule les objets Rappel pour les insérer ou les récupérer
Main ↔ GestionRappel	Main fait appel à GestionRappel pour lier l'interface à la logique de gestion.

Cette organisation reflète une hiérarchie simple et intuitive, garantissant une bonne communication entre les différentes couches sans les complexités d'un MVC complet.

IV. BASE DE DONNÉES

1.Présentation de la table de données

La base de données de l'application Papillon a été conçue pour être à la fois simple, fiable et extensible.
L'objectif était de permettre à chaque utilisateur d'avoir ses propres rappels personnels, sans avoir besoin d'un système d'authentification complexe.

1. Logique de conception

Au départ, l'équipe souhaitait concevoir une seule table contenant tous les rappels.
Cependant, lors du développement, nous avons introduit un système de clé unique locale pour identifier chaque utilisateur.

Cette évolution a rendu nécessaire la création d'une seconde table :

- une table pour les utilisateurs (utilisateur),
- une table pour leurs rappels (rappel).

Ainsi, chaque rappel est désormais lié à un utilisateur spécifique grâce à une clé étrangère (utilisateur_id).
Cette structure renforce la cohérence des données et permet à l'application de gérer plusieurs utilisateurs de manière transparente.

2. Structure des tables

Table **utilisateur**

Cette table contient les informations permettant d'identifier un utilisateur de manière unique.

- **id** : identifiant numérique auto-incrémenté (clé primaire)
- **cle** : clé unique générée localement et sauvegardée dans un fichier (.papillon_id) sur la machine de l'utilisateur

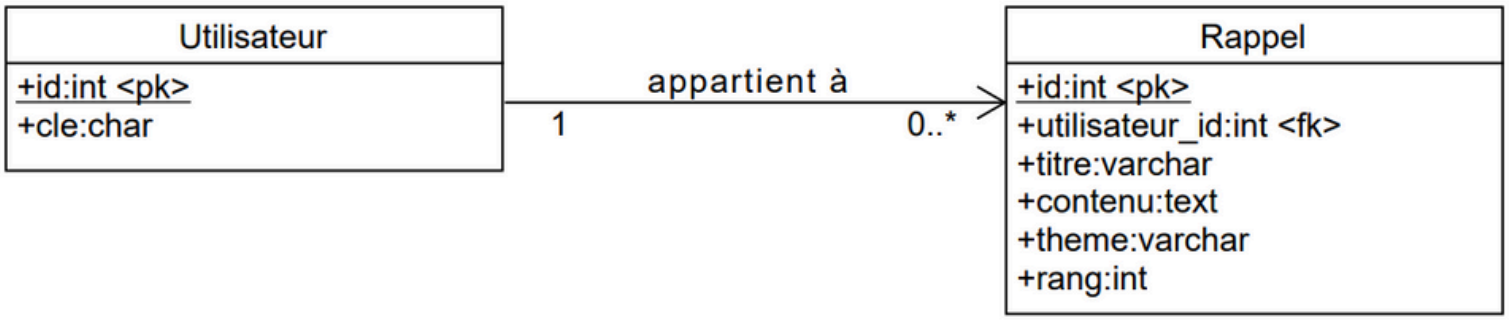
Elle sert uniquement à relier chaque clé générée à un identifiant interne en base.
Table **rappel**

Cette table stocke les informations des rappels créés par les utilisateurs.

- id : identifiant du rappel (clé primaire)
- utilisateur_id : clé étrangère pointant vers l'utilisateur propriétaire
- titre : titre du rappel
- contenu : texte du rappel
- theme : thème choisi (ex : personnel, travail, étude...)
- rang : niveau de priorité (valeur entière)

La relation entre les deux tables est une relation 1-n (un utilisateur peut avoir plusieurs rappels), assurée par une contrainte de clé étrangère avec suppression en cascade (ON DELETE CASCADE).
Ainsi, si un utilisateur est supprimé, tous ses rappels sont également effacés automatiquement.

2.Diagramme de classe complet



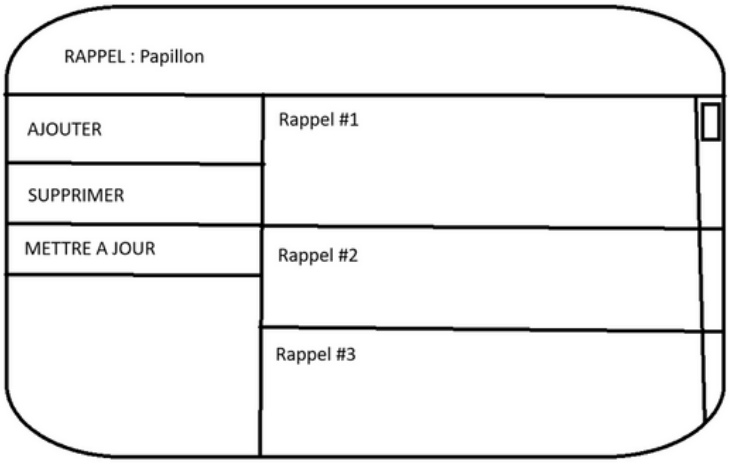
3.Lien entre la base et le programme

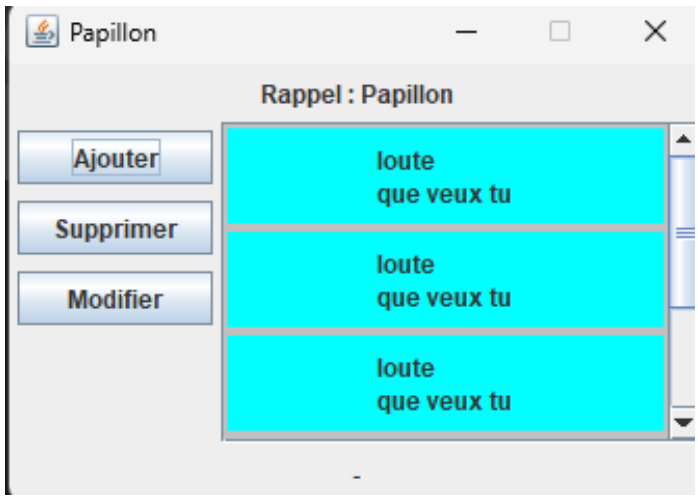
Le programme génère automatiquement une clé unique lors de sa première exécution. Cette clé est enregistrée dans un fichier local et également insérée dans la base de données. Elle permet d'identifier l'utilisateur et d'associer ses rappels à son profil.

Lors des prochaines ouvertures, si le fichier contenant la clé existe déjà, le programme récupère cette même clé pour reconnaître l'utilisateur. Cela lui permet de charger automatiquement les rappels qui lui sont associés dans la base de données, sans nécessiter de nouvelle identification.

1.Choix esthétiques et principes d'interface

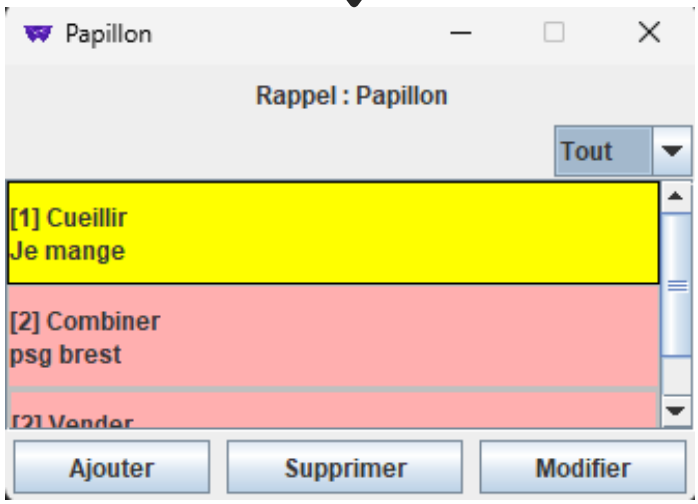
Pour concevoir notre interface graphique, nous avons d'abord réalisé une maquette Mid-Fidelity.
Cette étape nous a permis de visualiser le rendu final attendu et de mieux imaginer la structure générale de l'application avant de passer au développement.
La maquette nous a aussi aidés à définir les zones principales de l'interface (liste des rappels, boutons d'action, champs de saisie, etc.) et à anticiper la manière dont nous allions coder l'affichage.





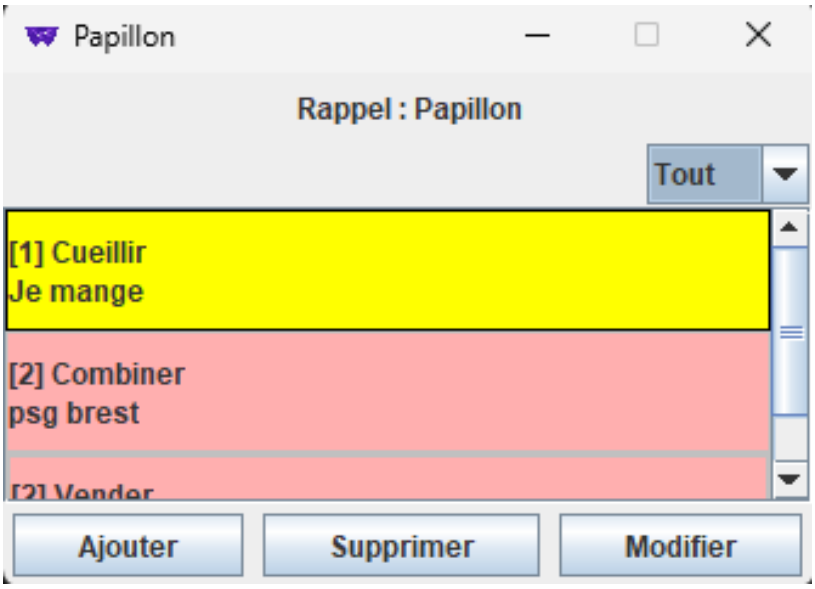
Dans un premier temps, nous avons réussi à obtenir un résultat assez fidèle à la maquette réalisée précédemment.

Par la suite, pour éviter certains problèmes d'affichage et rendre l'interface plus ergonomique, nous avons choisi cette disposition, en plaçant les rappels au centre de l'attention.

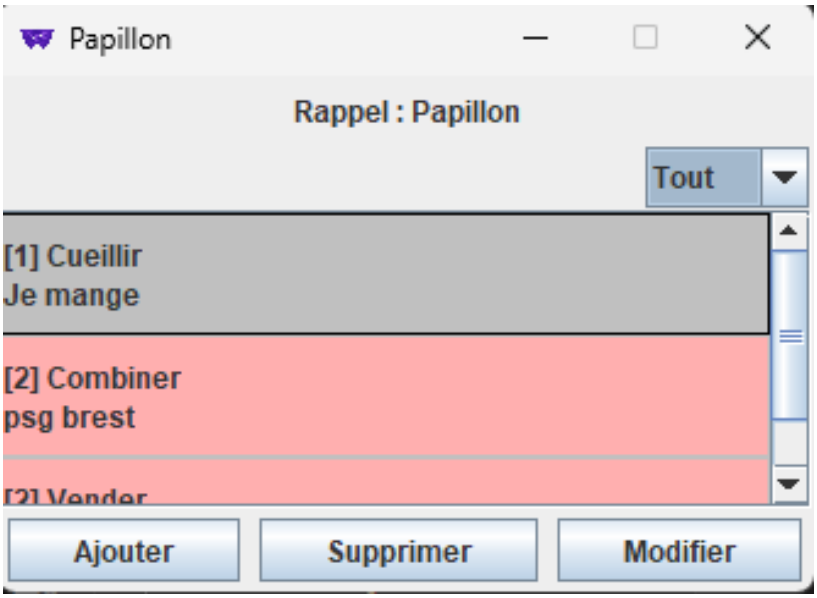


2.1Améliorations ergonomiques et interface

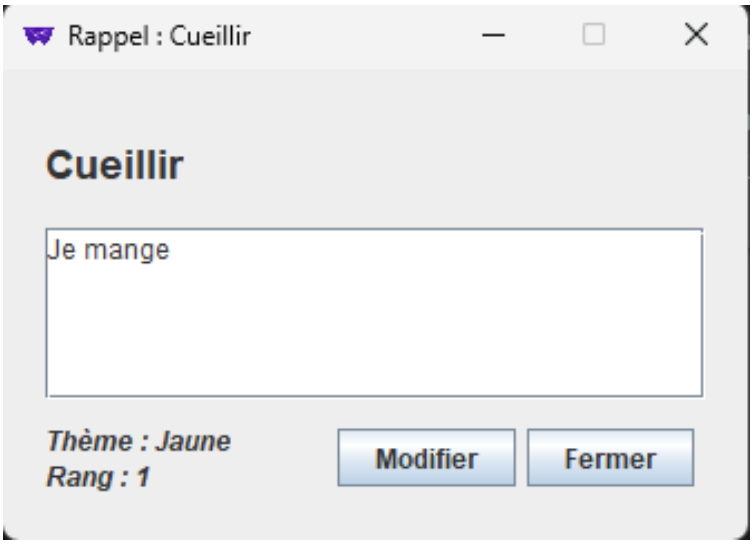
Lorsque l'utilisateur passe la souris sur un rappel, la bordure de ce dernier devient noire, ce qui permet de mieux indiquer l'élément survolé et d'inciter au clic.



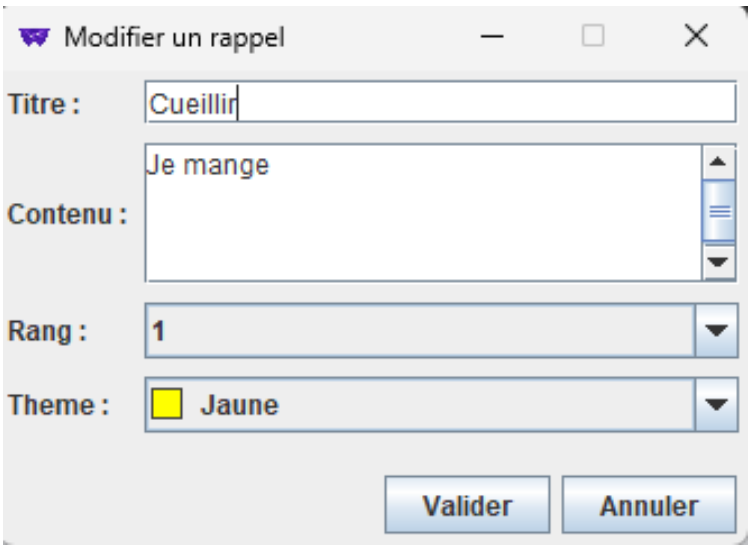
Lorsqu'un utilisateur clique sur un rappel, celui-ci est sélectionné et son fond devient gris. Un second clic permet de le désélectionner, en lui rendant sa couleur d'origine.



2.2Améliorations ergonomiques et interface



Lorsqu'un utilisateur ouvre un rappel, une fenêtre s'affiche pour lui présenter ses informations détaillées : le titre, le contenu, le thème et le rang. Il peut ensuite choisir de modifier ce rappel ou simplement de fermer la fenêtre pour revenir à la liste principale.



Lorsqu'il clique sur le bouton Modifier, une nouvelle fenêtre s'ouvre, reprenant les champs du rappel. L'utilisateur peut alors changer le titre, le contenu, le rang ou le thème, puis valider ses modifications ou annuler pour conserver la version précédente.

VI. DIFFICULTÉS RENCONTRÉES ET SOLUTIONS

1.Contraintes techniques

3 problème majeurs rencontrés :

Pendant la réalisation du projet Papillon, nous avons été confrontés à plusieurs difficultés techniques qui nous ont parfois fait perdre du temps, mais qui nous ont aussi beaucoup appris.
Voici les trois problèmes principaux que nous avons dû surmonter :

- **Gérer l'agencement des éléments affichés**

L'un des premiers obstacles a été de réussir à bien organiser les éléments de l'interface graphique. Le positionnement des boutons, des champs de texte et des panneaux n'était pas toujours simple à gérer (surtout avec le système de GridBagLayout).
Nous avons souvent dû refaire la mise en page pour obtenir un rendu clair et harmonieux, sans que les composants se chevauchent ou s'alignent mal. Ce travail a pris du temps, mais il nous a permis de mieux comprendre le fonctionnement des contraintes d'affichage et d'obtenir une interface propre et agréable à utiliser.

- **Implémentation de Maria DB (Base de donnée)**

La partie la plus difficile a été la configuration et l'intégration de MariaDB dans le projet.
On a rencontré plusieurs erreurs liées au pilote JDBC ("No suitable driver found"), dues au chemin d'accès et à la gestion du classpath entre le .jar et le dossier org/mariadb/jdbc.

Un autre problème a été la restructuration de la base de données : au lieu d'une seule table, on a dû en créer une seconde pour gérer les utilisateurs et relier chaque rappel à un identifiant unique.

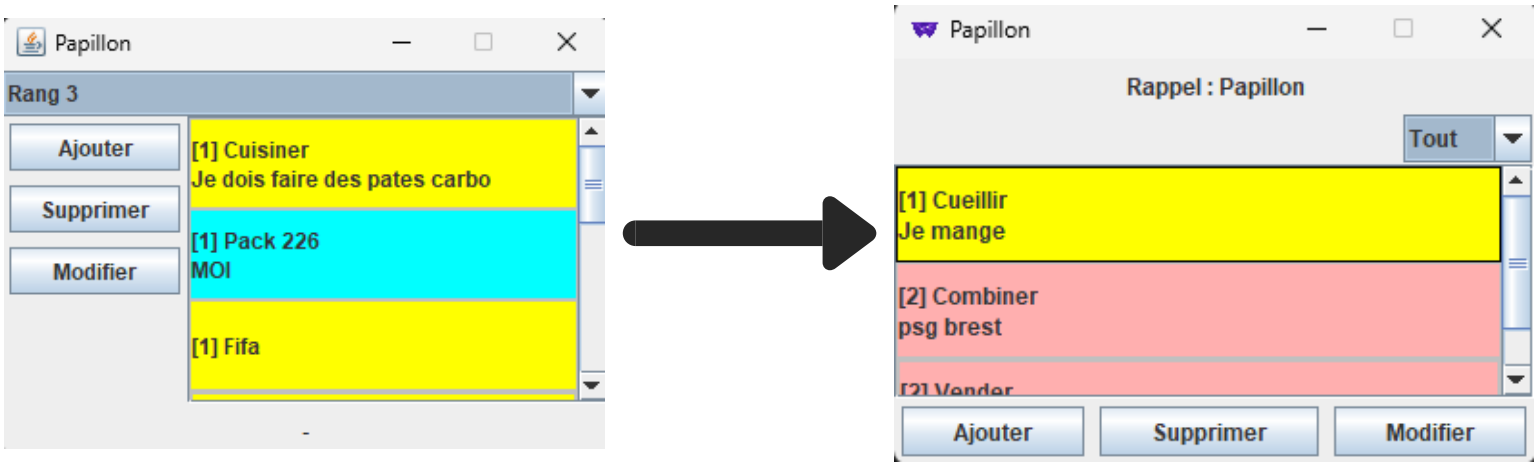
- **Affichage d'une couleur et d'un texte dans une liste déroulante**

Enfin, l'un des points les plus délicats a été d'ajouter à notre liste déroulante (la JComboBox) un affichage combinant à la fois du texte et une icon de couleur correspondant au thème du rappel. Par défaut, les listes déroulantes ne permettent pas ce genre d'affichage personnalisé. Nous avons donc dû apprendre à créer un renderer spécifique pour redessiner chaque élément de la liste. En Java, un renderer (ou cell renderer) est un composant spécial qui détermine comment un élément graphique est affiché à l'écran, sans forcément changer sa logique interne.
Même si cela nous a demandé plusieurs essais, le résultat final rend l'application beaucoup plus claire visuellement et agréable à utiliser.

2.1Choix de conception & Solutions

Gérer l'agencement des éléments affichés

Pour résoudre certains problèmes d'affichage, nous avons dû réorganiser entièrement la mise en page de la fenêtre principal de l'application. En effet, la manière dont l'on utilisait la méthode pour gérer l'affichage des composants entrait parfois en conflit avec la disposition initiale des éléments. Cela provoquait des chevauchements ou des alignements incorrects, rendant l'interface moins lisible.



Ce changement a non seulement permis de résoudre les problèmes d'affichage, mais aussi d'obtenir une interface plus équilibrée et agréable à utiliser.

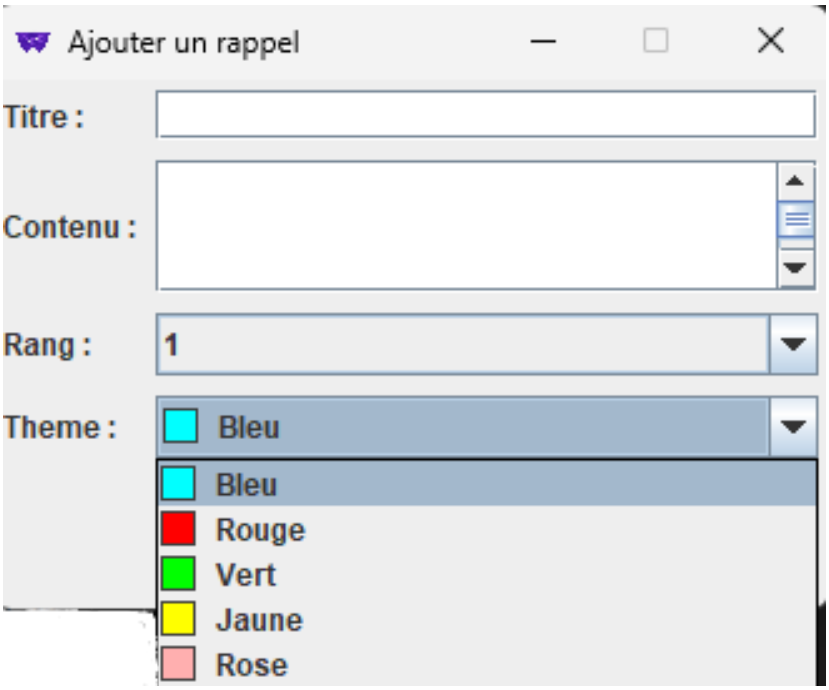
Ce réagencement a marqué une étape importante dans notre progression, car il nous a forcés à mieux comprendre le fonctionnement des gestionnaires de disposition en Java et à adapter notre code pour rendre l'affichage plus stable et modulable.

2.2Choix de conception & Solutions

Affichage d'une icon de couleur et d'un texte dans une liste déroulante

L'un des points d'amélioration de l'interface concernait la liste déroulante permettant de choisir le thème d'un rappel. L'idée était de permettre à l'utilisateur d'associer chaque thème à une petite icône colorée visible directement dans la liste. Par exemple, un carré bleu pour "Bleu", rouge pour "Rouge", vert pour "Vert", etc.

L'objectif était de rendre la sélection plus visuelle et intuitive : l'utilisateur peut ainsi repérer immédiatement la couleur associée à chaque thème sans avoir à lire tous les intitulés.



En revanche, les méthodes utilisés ne permettaient pas ce genre de personnalisation par défaut. Pour obtenir ce rendu, nous avons donc créé un renderer personnalisé pour la liste déroulante. Ce composant redéfinit la manière dont chaque élément est affiché : il associe une petite case colorée à chaque élément de la liste déroulante à gauche du texte, créant un repère visuel clair et esthétique.

Mettre cela en place n'a pas été simple au début : il a fallu tester plusieurs manières d'afficher les icônes et ajuster les espacements pour que tout reste bien aligné. Mais le résultat final répond à nos attentes : la liste est à la fois plus esthétique, plus ergonomique et surtout beaucoup plus agréable à utiliser

3.Organisation et travail d'équipe

Tout au long du projet, chacun d'entre nous a contribué à l'ensemble du développement, mais avec des domaines de spécialisation plus marqués. On peut dire que chaque membre a travaillé environ 50 % sur des tâches communes, et 50 % sur un domaine particulier où il était plus à l'aise.

- **Séri-Khane : Base de donnée**

Séri-Khane s'est principalement occupé de la base de données, en veillant à la bonne communication entre MariaDB et l'application**.**

- **Jenson : Agencement du code**

Jenson s'est concentré sur l'agencement du code et la structure générale du programme, notamment la mise en place des différentes classes et de l'interface.

- **Aylane : Description du code**

Aylane principalement a pris en charge la description et la documentation du code, afin de garder une trace claire du fonctionnement de chaque partie du projet.

Même si chacun avait sa spécialité, nous avancions toujours ensemble. Dès qu'un problème majeur survenait, nous mettions nos efforts en commun pour le résoudre. Cette entraide constante nous a permis de progresser au même rythme, d'apprendre les uns des autres et de maintenir une bonne cohésion d'équipe tout au long du développement.

VII. CONCLUSIONS

1.1Conclusion personnelle de chaque membre

Jenson VAL

Ce projet m'a beaucoup appris, notamment sur l'utilisation de Git. Nous avons parfois fait des erreurs et rencontré quelques bugs, mais ces difficultés nous ont permis de mieux comprendre comment gérer les différentes versions du code et de renforcer notre manière de travailler ensemble.

Au début, j'avais du mal à comprendre la gestion de l'affichage des éléments. Les contraintes du GridBagLayout ne m'ont pas paru simples à maîtriser et le positionnement des éléments me paraissait confus. Avec la pratique et en échangeant avec mes coéquipiers, j'ai fini par bien comprendre comment tout s'articulait, ce qui m'a permis de construire une interface plus claire et plus cohérente.

Un autre point sur lequel j'ai rencontré des difficultés concerne la mise en place de la fonction de tri des rappels. Au départ, je comprenais mal la logique à adopter pour classer les rappels selon leur niveau d'importance. Il a fallu plusieurs essais avant que la méthode fonctionne correctement, notamment pour que l'affichage se mette bien à jour après le tri.
Ce travail m'a permis de mieux comprendre le lien entre la logique du code et l'affichage graphique, et de voir comment une simple modification dans la structure du programme pouvait influencer directement l'interface. Même si cette partie m'a demandé du temps, elle m'a beaucoup aidé à progresser et à mieux maîtriser la gestion dynamique des composants dans l'application.

Au-delà de la technique, ce projet m'a aussi montré une fois de plus l'importance d'une bonne organisation d'équipe. Nous avons réussi à bien répartir les tâches, à rester à jour sur nos avancées et à nous entraider dès qu'un problème se présentait. Cette bonne coordination a vraiment facilité le développement.

Avec le recul, je trouve cette expérience très enrichissante, car elle m'a permis d'améliorer à la fois mes compétences techniques et ma façon de travailler en équipe.

VII. CONCLUSIONS

1.2Conclusion personnelle de chaque membre

Aylane SEHL

Cette SAÉ a été une expérience vraiment formatrice pour moi. Au départ, j'ai rencontré pas mal de difficultés, notamment avec la gestion de l'affichage et les contraintes du GridBagLayout. J'avais du mal à comprendre pourquoi certains éléments ne s'affichaient pas comme prévu, mais ces problèmes m'ont forcé à creuser et à mieux comprendre le fonctionnement des contraintes. Finalement, c'est justement en corrigeant ces erreurs que j'ai le plus progressé : j'ai appris à être plus rigoureux et à anticiper la disposition des composants.

J'ai aussi beaucoup amélioré ma façon de structurer le code. J'ai bien assimilé l'importance de séparer les responsabilités : une classe = un rôle. Ça paraît simple, mais en pratique, ça change tout dans la clarté et la maintenance du projet.
Un autre point sur lequel j'ai pris du plaisir, c'est la documentation. J'ai rédigé le README avec soin, en essayant d'expliquer clairement chaque commande et chaque étape. J'ai compris à quel point un bon fichier README et MAKEFILE peut faciliter la compréhension du projet pour les autres ; c'est un peu la "porte d'entrée" du code, et j'ai voulu qu'il soit propre et compréhensible.
Au sein de l'équipe, chacun avait sa spécialité, Séri-Khane s'occupait surtout de la base de données, Jenson de l'agencement du code mais on échangeait sur Discord souvent et on s'aidait mutuellement quand un problème bloquait. Cette entraide a été essentielle, et c'est aussi ce qui rend le travail d'équipe agréable : on apprend les uns des autres.

En résumé, ce projet m'a permis d'approfondir mes connaissances techniques tout en développant des compétences de collaboration et de communication. J'en ressors plus à l'aise avec Java, les interfaces graphiques et la conception orientée objet, mais aussi avec une meilleure méthode de travail et de documentation.

1.3Conclusion personnelle de chaque membre

Seri-Khane YOLOU

Cette SAÉ m'a beaucoup apporté, autant sur le plan technique que personnel. J'ai beaucoup apprécié la liberté laissée dans la conception du projet : elle m'a permis d'expérimenter, de tester mes idées et de progresser par la pratique. Travailler sur l'intégration de la base de données avec Java m'a aidé à approfondir ma compréhension de java.sql, de MariaDB et de la structure globale d'une application orientée objet. J'ai également pu mettre en place un système d'identification par clé unique avec la classe UUID, une solution simple et sécurisée que j'ai découverte au fil de mes recherches.

Sur le plan du travail d'équipe, cette SAÉ m'a rappelé la logique de la méthode Scrum. Nous avons organisé notre progression sous forme de sprints, avec des objectifs clairs à chaque itération et des réunions de suivi pour évaluer nos avancées. Ces moments de rétrospective nous ont permis d'ajuster notre manière de travailler, de mieux répartir les responsabilités et d'améliorer la communication entre les membres du groupe. Cette dynamique collective a créé une ambiance motivante, où chacun contribuait à la réussite commune du projet.

La gestion du versionnage sur Git et Grond a aussi été un apprentissage important. J'ai mieux compris l'intérêt d'un suivi rigoureux des versions, de la création de branches dédiées (comme pour les tests ou les modèles), et de l'importance des messages de commit clairs. Cette rigueur m'a permis de mieux collaborer et d'éviter les erreurs de synchronisation.

En résumé, cette SAÉ m'a permis de renforcer mes compétences techniques, d'apprendre à travailler efficacement en équipe et de mieux structurer ma façon de développer un projet logiciel.

VII. CONCLUSIONS

2.Conclusion générale du groupe

La **SAÉ 3.1 – Application Papillon** nous a offert une expérience complète de développement logiciel, mais surtout une **mise en situation concrète** de ce qu'implique la conception d'un projet informatique du début à la fin.
Cette liberté pédagogique, que nous avons beaucoup appréciée, a aussi révélé la **complexité de l'autonomie en équipe** : il ne s'agissait pas seulement d'écrire du code, mais de réfléchir à la **cohérence d'un ensemble**, base de données, interface, logique, et maintenance.

Sur le plan technique, nous avons réalisé que les difficultés ne venaient pas uniquement du langage ou des outils, mais souvent des **interactions entre les différentes couches** du projet.
Par exemple, le lien entre **MariaDB** et **Java** nous a forcés à comprendre réellement le rôle du pilote JDBC, la gestion du classpath, et la communication entre objets métiers et tables SQL.
Ces obstacles, loin d'être de simples erreurs techniques, nous ont poussés à adopter une approche plus **rigoureuse et méthodique** du débogage : lire les logs, tester isolément chaque couche, et documenter les comportements inattendus.

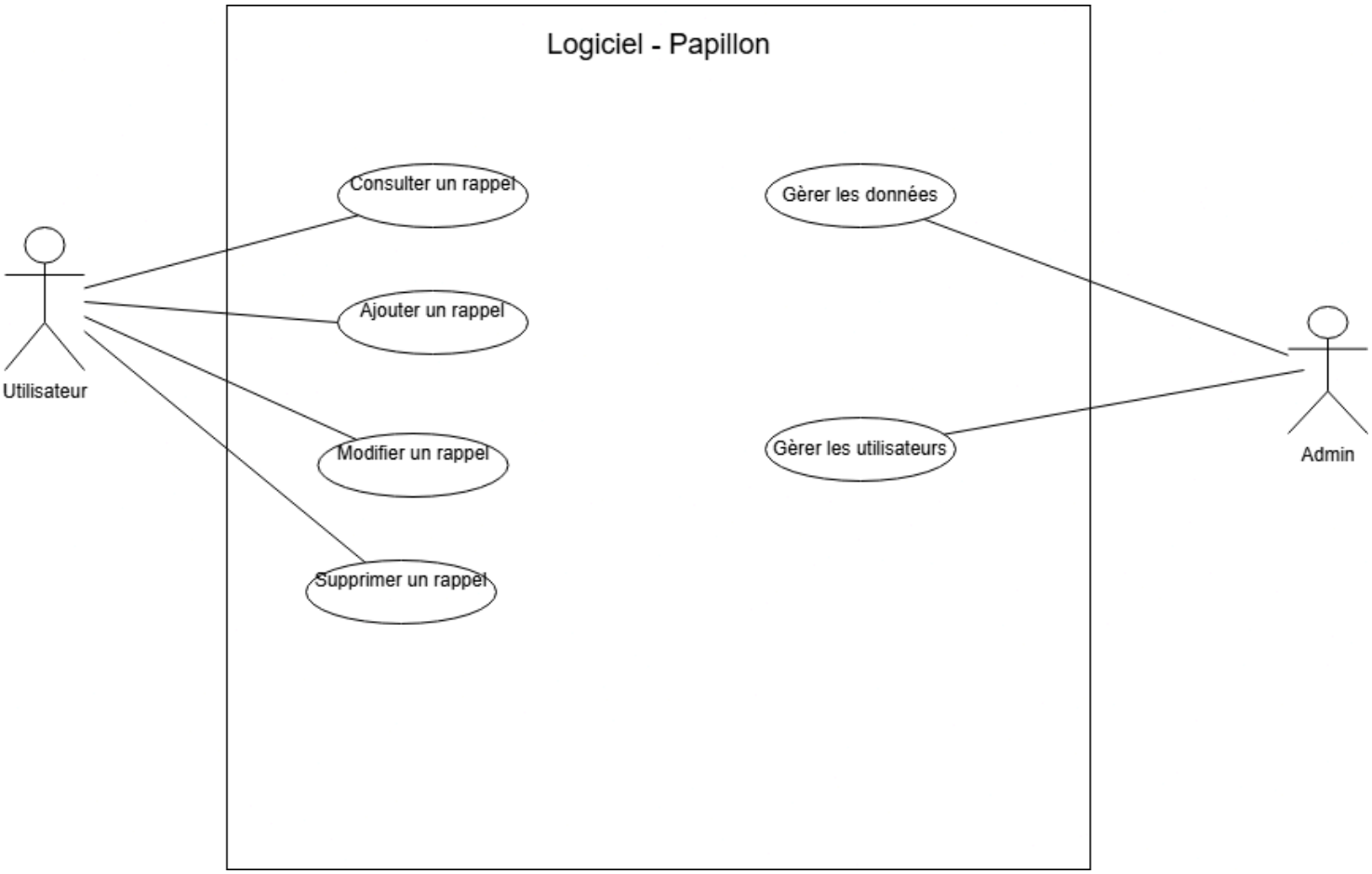
La structure du projet, pensée en **programmation orientée objet**, nous a également appris à raisonner en termes de **responsabilités** : chaque classe devait avoir une fonction claire, et chaque dépendance devait être justifiée.
Cette logique a profondément changé notre manière de coder : moins d'improvisation, plus de conception. Nous avons compris qu'un bon code n'est pas celui qui "fonctionne", mais celui qui peut être **repris**, **compris** et **amélioré** par quelqu'un d'autre.

D'un point de vue organisationnel, nous avons adopté une méthode proche du **Scrum**, avec des **sprints**, des **review**, et des **rétrospectives** régulières.
Ce cadre nous a aidés à rythmer notre progression, mais aussi à **prendre du recul** sur nos décisions : parfois, il valait mieux supprimer une fonctionnalité complexe plutôt que de compromettre la stabilité générale. Cette réflexion sur la **priorisation des tâches** nous a appris que dans un vrai projet, **le réalisme prime sur la perfection.**

Enfin, l'utilisation de **Git et Grond** nous a fait prendre conscience de la valeur du **travail collaboratif versionné** : la clarté des commits, la discipline du merge et la communication entre développeurs sont devenues des compétences à part entière. Chaque commit représentait une **petite victoire**, chaque fusion un pas de plus vers un travail cohérent et partagé.

En fin de compte, même si ce projet reste une SAÉ parmi d'autres, il a constitué une **expérience marquante**, aussi bien sur le plan technique que humain.
Il nous a appris que la programmation n'est pas qu'une affaire de code, mais une **discipline d'organisation,** de **réflexion** et de **communication**.
Papillon nous a donné une vision plus réaliste du développement logiciel : c'est un travail d'équipe où la technique doit toujours servir la compréhension, la clarté et la collaboration.

1.Diagrammes UML détaillés



Projet réalisé par

- Aylane SEHL

- Séri-Khane YOLOU

- Jenson VAL

Enseignant LUC HERNANDEZ