ColoCat

Plateforme de gestion des colocations

Projet de Technologie Objet

Rapport

Encadrants : Judicaël Bedouet, Philippe Roussel

Année universitaire 2024–2025

Équipe CD04
ABAKHANE Achraf – BOULAALAM El-Mehdi – MOUACHA Bassou
EL-BOUBEKRI Oussama – ELLAIK Fadwa – DIOUANE Walid – RIFAI Aya

Table des matières

1 Introduction

Ce rapport présente le développement de *ColoCat*, une application conçue dans le cadre du projet de Technologie Objet pour faciliter la gestion des colocations. Elle propose des outils centralisés pour gérer les dépenses, événements, incidents et réservations d'espaces communs. Le rapport détaille les principales fonctionnalités, l'architecture, les choix de conception, l'organisation de l'équipe et l'utilisation des méthodes agiles.

2 Principales fonctionnalités et objectifs

2.1 Elevator Pitch

- **Pour :** Les colocataires souhaitant organiser leur vie commune de manière efficace
- **Qui souhaitent :** Faciliter la communication, l'organisation et le suivi des tâches entre colocataires
- **Notre produit est :** ColoCat, une plateforme innovante de gestion des colocations
- **Qui :** Centralise la gestion des finances communes, des plannings et des discussions
- **À la différence de :** Les solutions traditionnelles fragmentées ou les méthodes manuelles
- **Permet de :** Rendre la colocation plus harmonieuse et efficace en centralisant toutes les informations dans un espace unique

2.2 Charte produit

Vision produit	Notre vision est de développer une plateforme innovante pour la gestion des colocations, facilitant la communication, l'organisation et le suivi des tâches entre colocataires. ColoCat propose des outils pour aider à la gestion des finances communes, des plannings de tâches ménagères et des discussions entre colocataires. L'idée est de rendre la colocation plus harmonieuse et efficace en centralisant toutes les informations et interactions nécessaires dans un espace unique.
Clients/Utilisateurs	Les utilisateurs de ColoCat sont principalement des colocataires de tous âges souhaitant organiser leur vie commune de manière efficace. Cela inclut les étudiants, les jeunes professionnels, et toute personne vivant en colocation qui souhaite simplifier la gestion des aspects pratiques de la cohabitation grâce à des outils numériques intuitifs.
Valeur Métier	La valeur métier de ColoCat réside dans la centralisation et la simplification de la gestion de colocation. Elle permet de réduire les conflits liés aux dépenses en offrant un suivi transparent, d'améliorer la communication entre colocataires, et d'optimiser l'utilisation des espaces communs grâce au système de réservation. L'application favorise une meilleure organisation et une vie en colocation plus harmonieuse.

TABLE 1 – Charte produit de l'application

2.3 User Stories

2.3.1 Interface utilisateur et authentification

1. En tant qu'utilisateur, je veux pouvoir me connecter à ColoCat avec mes identifiants personnels pour accéder aux services de l'application.
2. En tant que nouvel utilisateur, je veux pouvoir créer une nouvelle colocation pour commencer à organiser la vie commune avec mes futurs colocataires.
3. En tant qu'utilisateur, je veux avoir accès à un tableau de bord centralisé pour naviguer facilement entre les différentes fonctionnalités.

1. En tant que colocataire, je veux pouvoir ajouter une dépense partagée pour que tous les membres de la colocation puissent la voir et participer au remboursement.
2. En tant que colocataire, je veux pouvoir voir le solde de chaque membre pour savoir qui doit de l'argent à qui.
3. En tant que colocataire, je veux pouvoir marquer une dépense comme réglée pour mettre à jour les soldes.
4. En tant que colocataire, je veux pouvoir voir les statistiques de dépenses par catégorie pour mieux comprendre nos habitudes de consommation.

2.3.2 Gestion des événements

1. En tant que colocataire, je veux pouvoir créer un événement (repas, soirée, tâche ménagère) pour organiser la vie de la colocation.
2. En tant que colocataire, je veux pouvoir voir tous les événements planifiés pour m'organiser en conséquence.
3. En tant que colocataire, je veux pouvoir modifier ou supprimer un événement que j'ai créé.

2.3.3 Gestion des incidents

1. En tant que colocataire, je veux pouvoir signaler un incident (panne, problème de voisinage, litige) pour informer les autres membres.
2. En tant que colocataire, je veux pouvoir suivre le statut des incidents pour savoir s'ils sont en cours de traitement ou résolus.
3. En tant que colocataire, je veux pouvoir résoudre un incident avec une solution pour clôturer le problème.

2.3.4 Système de réservation

1. En tant que colocataire, je veux pouvoir réserver un espace commun (salle de bain, machine à laver, salon, cuisine) pour éviter les conflits d'usage.
2. En tant que colocataire, je veux pouvoir voir les réservations existantes pour planifier mes propres réservations.
3. En tant que colocataire, je veux pouvoir annuler mes réservations si mes plans changent.

3 Architecture de l'application

3.1 Découpage en packages

L'application est organisée en plusieurs packages selon les fonctionnalités :
— `login` : Gestion de l'authentification et des colocations
— `evenements` : Gestion des événements de la colocation
— `incidents` : Gestion des incidents et problèmes
— `reservation` : Système de réservation des espaces communs
— `gui` : Interfaces graphiques utilisateur
— `main` : Point d'entrée de l'application

3.2 Classes principales

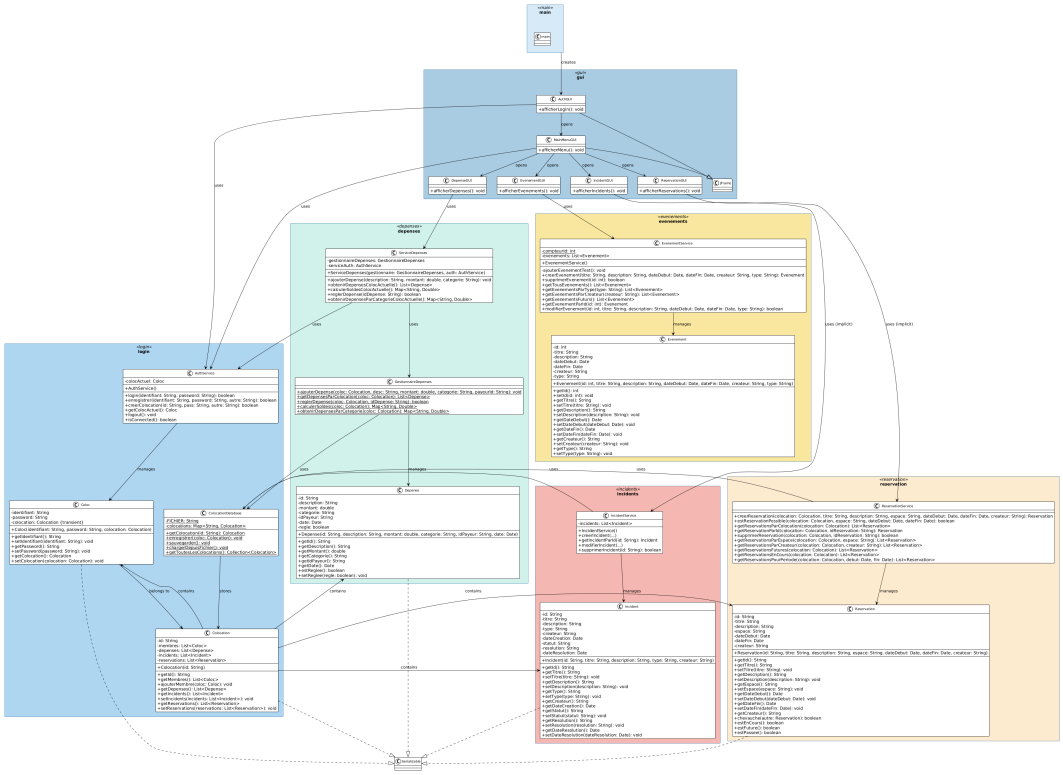

FIGURE 1 – Diagramme UML détaillant les classes

3.2.1 Package login

Classe Colocation
— Représente une colocation avec ses membres, dépenses, incidents et réservations
— Méthodes principales : `ajouterMembre()`, getters pour les listes
Classe Coloc
— Représente un colocataire avec identifiant et mot de passe
— Relation avec la colocation via un attribut `colocation`
Classe AuthService
— Service d'authentification gérant la connexion/déconnexion

— Méthodes : `login()`, `enregistrer()`, `creerColocation()`

Classe Depense

— Représente une dépense avec montant, catégorie, payeur et statut
— Attributs : id, description, montant, catégorie, idPayeur, date, regle

Classe GestionnaireDepenses

— Service statique pour la gestion des dépenses
— Méthodes : `ajouterDepense()`, `calculerSoldes()`, `reglerDepense()`

3.2.2 Package evenements

Classe Evenement

— Représente un événement avec titre, description, dates et type
— Types : REPAS, SOIREE, TACHE_MENAGERE, etc.

Classe EvenementService

— Service de gestion des événements avec compteur d'ID
— Méthodes : `creerEvenement()`, `supprimerEvenement()`, filtres par type/créateur

3.2.3 Package incidents

Classe Incident

— Représente un incident avec statuts : SIGNALE, EN_COURS, RESOLU
— Types : PANNE, VOISINAGE, LITIGE, AUTRE
— Méthodes métier : `marquerEnCours()`, `resoudre()`

Classe IncidentService

— Service statique pour la gestion des incidents
— Méthodes de filtrage par statut, type, créateur

3.2.4 Package reservation

Classe Reservation

— Représente une réservation d'espace avec dates de début/fin
— Espaces : SALLE_DE_BAIN, MACHINE_A_LAVER, SALON, CUISINE
— Méthode `chevauche()` pour détecter les conflits

Classe ReservationService

— Service statique avec vérification de disponibilité
— Méthode `estReservationPossible()` pour éviter les chevauchements

3.3 Persistance des données

Classe ColocationDatabase

— Gestion de la persistance via sérialisation Java
— Sauvegarde automatique dans le fichier `colocation.ser`
— Méthodes : `sauvegarder()`, `chargerDepuisFichier()`

3.4 Diagrammes de classe

L'architecture suit le pattern MVC avec :
— **Modèle** : Classes métier (Colocation, Depense, Evenement, etc.)
— **Vue** : Classes GUI (MainMenuGUI, DepenseGUI, etc.)

— **Contrôleur** : Services (AuthService, GestionnaireDepenses, etc.)
Les relations principales sont :
— Colocation 1—* Coloc (composition)
— Colocation 1—* Depense (composition)
— Colocation 1—* Incident (composition)
— Colocation 1—* Reservation (composition)

4 Choix de conception et réalisation

4.1 Choix technologiques

4.1.1 Langage et framework

Nous avons choisi Java avec Swing pour l'interface graphique car :
— Maîtrise du langage par l'équipe
— Swing permet de créer rapidement des interfaces fonctionnelles
— Portabilité multiplateforme
— Intégration native avec les design patterns orientés objet

4.1.2 Persistance

La sérialisation Java a été choisie pour :
— Simplicité d'implémentation
— Pas besoin de base de données externe
— Sauvegarde automatique de l'état complet de l'application

4.1.3 Architecture

Une architecture en couches avec :
— Couche présentation (GUI)
— Couche service (logique métier)
— Couche modèle (entités)
— Couche persistance (ColocationDatabase)

4.2 Patterns de conception utilisés

4.2.1 Singleton

La classe `ColocationDatabase` implémente un singleton pour garantir une unique source de vérité pour les données.

4.2.2 Service Layer

Les classes de service (`AuthService`, `GestionnaireDepenses`, etc.) encapsulent la logique métier et fournissent une API claire.

4.2.3 DTO (Data Transfer Object)

Les entités comme `Depense`, `Evenement` servent de containers de données avec getters/setters.

4.3 Organisation de l'équipe et mise en œuvre des méthodes agiles

4.3.1 Méthodologie

Nous avons adopté une approche agile inspirée de Scrum avec :
— Définition des user stories
— Développement itératif par fonctionnalités
— Réunions de coordination régulières
— Tests continus des fonctionnalités développées

4.3.2 Organisation

L'équipe s'est organisée avec :
— Un Product Owner pour définir les besoins
— Un Scrum Master pour faciliter le processus
— Des développeurs spécialisés par module

4.3.3 Outils de collaboration

— Git pour le versionnement du code.
— GitLab pour le dépôt et la gestion collaborative du code source.
— GitHub pour la gestion des tâches et des issues.
— WhatsApp : création d'un groupe pour assurer une communication rapide entre les membres.
— Google Meet : réunions organisées tous les 15 jours afin de faire le point sur l'avancement du projet et de répartir les tâches de l'itération suivante.
— Partage d'écran pour faciliter le développement en pair programming.
— Trello

4.4 Problèmes rencontrés et solutions apportées

4.4.1 Gestion de la persistance

Problème : Relations bidirectionnelles causant des problèmes de sérialisation.
Solution : Utilisation du mot-clé `transient` et reconstruction des liens au chargement.

4.4.2 Gestion des dates

Problème : Complexité de la gestion des dates pour les réservations.
Solution : Implémentation d'une méthode `chevauche()` pour détecter les conflits.

4.4.3 Interface utilisateur

Problème : Synchronisation des vues après modification des données.
Solution : Méthodes de rechargement des données dans chaque interface.

4.4.4 Architecture évolutive

Problème : Ajout de nouvelles fonctionnalités sans casser l'existant.
Solution : Services statiques et interfaces découplées permettant l'extension.

5 Fonctionnalités implémentées

5.1 Authentification et gestion des colocations

— Création de nouvelle colocation avec premier membre administrateur
— Connexion/inscription des colocataires existants
— Gestion sécurisée des sessions utilisateur

5.2 Gestion des dépenses

— Ajout de dépenses avec catégorisation
— Calcul automatique des soldes entre colocataires
— Marquage des dépenses comme réglées
— Statistiques par catégorie de dépenses

5.3 Planification d'événements

— Création d'événements typés (repas, soirées, tâches)
— Visualisation du calendrier des événements
— Filtrage par type et créateur
— Gestion des événements futurs

5.4 Système de signalement d'incidents

— Signalement d'incidents avec typologie
— Suivi du statut (signalé, en cours, résolu)
— Historique des résolutions
— Filtrage par statut et type

5.5 Réservation d'espaces communs

— Réservation avec vérification de disponibilité
— Gestion des conflits de réservation
— Visualisation du planning des réservations
— Annulation de réservations

6 Tests et validation

6.1 Tests fonctionnels

Chaque fonctionnalité a été testée avec :
— Cas nominaux de fonctionnement
— Cas d'erreur et de validation
— Tests d'intégration entre modules

6.2 Tests d'interface

— Vérification de l'ergonomie des interfaces
— Tests de responsivité des boutons et actions

— Validation des messages d'erreur et de succès

7 Conclusion

Ce projet nous a permis de mettre en pratique les concepts de la programmation orientée objet dans un contexte concret. *ColoCat*, l'application développée, répond aux besoins de gestion de colocation en proposant une solution intégrée et ergonomique qui facilite la communication, l'organisation et le suivi des tâches entre colocataires.

Points forts du projet

— Architecture modulaire et extensible, facilitant l'ajout de nouvelles fonctionnalités.
— Interface utilisateur intuitive avec des écrans d'accueil clairs (connexion / création de colocation).
— Gestion complète du cycle de vie des données avec persistance automatique.
— Système de réservation innovant basé sur un calendrier interactif.
— Respect des bonnes pratiques de développement orienté objet.

Pistes d'amélioration

— Intégration d'un système de notifications push entre colocataires.
— Ajout d'un module de messagerie pour améliorer la communication interne.
— Développement d'une version mobile pour un accès simplifié.
— Mise en place de rappels automatiques pour les paiements et les tâches.
— Intégration avec des services de paiement en ligne.

Conclusion

Cette expérience de développement de *ColoCat* nous a permis d'acquérir des compétences précieuses en développement logiciel, gestion de projet agile et travail collaboratif. Notre équipe CD04 a su relever le défi de créer une plateforme complète, optimisant la gestion de la vie en colocation et garantissant une cohabitation fluide et harmonieuse.