INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

Projet POOA GM4
2023/2024

Coloc'App

Rapport

Étudiants : Constance BAU, Kenza BENNANI, Paul MÉHAUD, Ines SILHADI

Encadrants : Mathieu BOURGAIS et Cecilia ZANNI-MERK

Table des matières

1. Introduction **3**
I. Objectifs clés . . . 3
II. Fonctionnalités principales . . . 4
a. Propriétaire . . . 4
b. Colocataire . . . 4

2. Modélisation **6**
I. Diagramme de cas d'utilisation . . . 6
a. ...du loueur . . . 6
b. ...du colocataire . . . 7
II. Navigation dans les menus . . . 8
III. Diagrammes de classes . . . 11
a. Version 1 . . . 11
b. Version 2 . . . 12
IV. Diagramme d'architecture réseau . . . 15

3. Exemples d'utilisation de Coloc'APP **17**

4. Organisation du projet **24**
I. Répartition du travail . . . 24
II. Rétrospective . . . 24

5. Conclusion **26**

1. Introduction

Le but de ce projet est d'implémenter une application de gestion de colocation. Cette application sera utilisable par les différents acteurs de la colocation, à savoir : le propriétaire et les différents locataires. D'une part, le propriétaire aura accès aux dossiers de ses locataires ; d'autre part, les colocataires profiteront de fonctionnalités de gestion de tâches ménagères et d'organisation de sorties. Le cœur de ce projet se concentrera sur la fonctionnalité réseau. En effet, le logiciel devra permettre une utilisation simultanée par plusieurs utilisateurs, avec une partie réservée au propriétaire pour suivre les paiements mensuels de chaque locataire, et une autre partie destinée à chaque locataire d'un appartement partagé pour organiser les tâches ménagères et les sorties en ville, notamment dans les bars. Ceci donne une idée de nos objectifs dans leur globalité, rentrons à présent dans le vif du sujet.

Dans ce rapport, nous explorerons les objectifs clés qui sous-tendent notre démarche. Nous détaillerons ensuite l'organisation du projet. Les fonctions principales de l'application seront détaillées, décrivant comment elle offre une solution complète pour la gestion des appartements partagés, ceci étant accompagné de multiples diagrammes qui illustreront notre démarche. Enfin, nous jetterons un regard sur les futures améliorations envisagées et possibles expansions de l'application.

I. Objectifs clés

Le logiciel de gestion d'appartements partagés comportera plusieurs fonctions principales destinées à simplifier la gestion tant pour le propriétaire que pour les colocataires. Nos objectifs s'articulent en deux axes principaux : la gestion de la partie concernant le propriétaire et la gestion de la partie concernant les colocataires. Du côté du propriétaire, les principales fonctions seront axées sur la gestion des paiements mensuels des locataires. Cela inclura la possibilité de générer des rapports de paiement, et de gérer l'ajout et la suppression des locataires. Pour les colocataires, le logiciel proposera deux grandes catégories de fonctionnalités. La première concernera la gestion des

tâches ménagères, permettant aux colocataires de répartir les responsabilités et de suivre l'avancement des différentes tâches. La seconde catégorie sera dédiée à l'organisation des sorties dans les bars de Rouen. Cette dernière fera par ailleurs appel à l'algorithmique des graphes pour calculer le plus court chemin.

II. Fonctionnalités principales

Avant toutes choses, un premier enjeu commun aux deux acteurs est la gestion de l'authentification. Un identifiant et un mot de passe seront générés pour chaque utilisateur avec des combinaisons classiques : nom-prénom-chiffres, et seront sauvegardés dans des fichiers. Un second enjeu commun est la prise en charge de la communication entre les différents utilisateurs. C'est d'ailleurs là que réside l'aspect réseau du projet. Chaque utilisateur a le choix d'envoyer une annonce à l'un ou à tous les autres utilisateurs qu'il sélectionne dans un menu déroulant et de consulter les annonces envoyées et reçues. L'idée est exactement la même que pour une messagerie classique à l'exception de l'affichage de toutes les annonces confondues.

a. Propriétaire

Concernant le propriétaire, la problématique principale de cet acteur est de lui permettre un suivi des paiements de ses locataires par mois. Pour ce faire, il aura la possibilité d'ajouter ou de supprimer des locataires. Il aura ensuite accès au tableau des paiements de chacun, ainsi que ses informations principales. Il est donc nécessaire d'enregistrer de nouveaux loyers en entrant le montant, la période et le statut (Payé ou Non Payé).

b. Colocataire

Comme dit précédemment, le colocataire aura la possibilité de gérer les tâches ménagère mais aussi organiser des sorties. L'implémentation de cette partie est la plus fastidieuse compte tenu des nombreuses contraintes à respecter. Dans un premier temps, l'application offre la possibilité au locataire d'accéder à sa to-do-list, qu'il modifie lui même. Comme nous avons pu le mentionner plus tôt, les colocataires pourront générer des sorties dans des bars à Rouen en optimisant le temps de trajet. Il y a deux possibilités pour les sorties dans les bars :

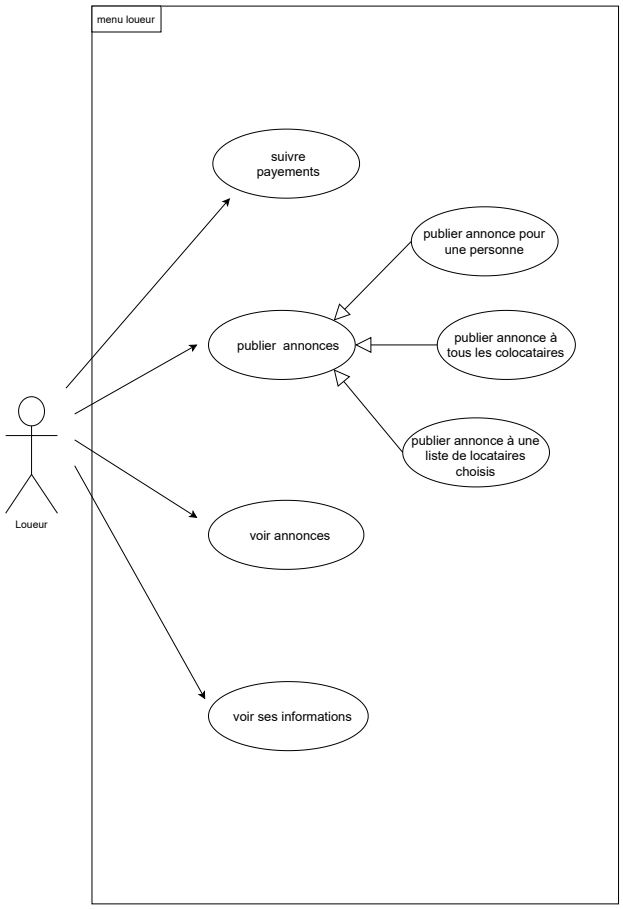
- Les colocataires souhaitent faire la tournée des bars de Rouen. Pour ce faire ils choisissent le bar duquel ils souhaitent partir et l'algorithme leur donne l'ordre dans lequel parcourir tous les bars pour que le temps de trajet total soit le moins long possible.
- Une autre possibilité est d'offrir aux colocataires le choix de sélectionner le bar duquel ils veulent partir et celui auquel ils veulent arriver et l'algorithme leur donne le plus court chemin entre ces deux bars sous forme d'un chemin de bars.

2. Modélisation

I. Diagramme de cas d'utilisation . . .

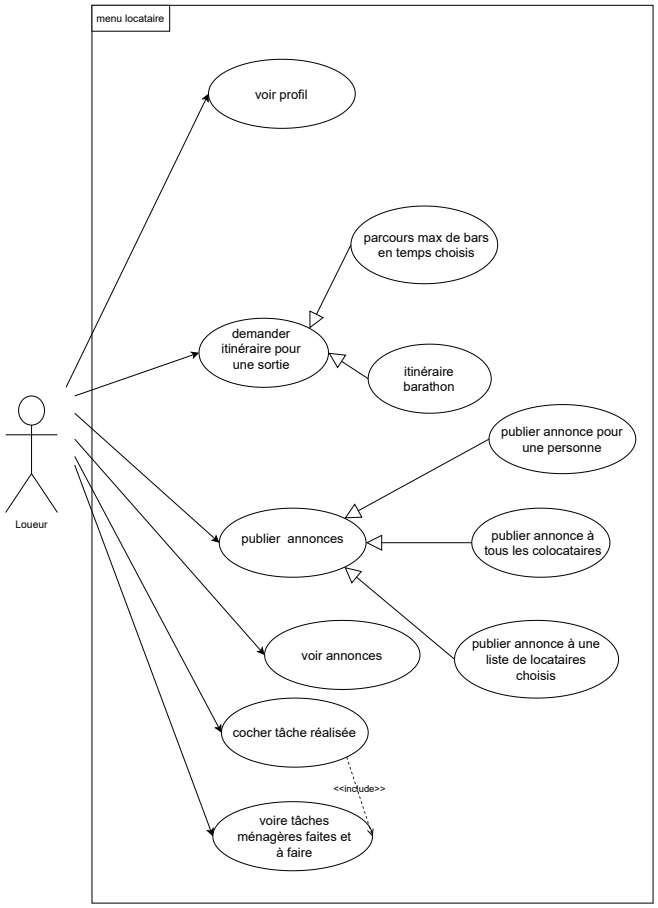
Une première étape de conception a été la réalisation d'un diagramme de cas d'utilisations afin d'avoir une vision globale des besoins.

a. . . .du loueur



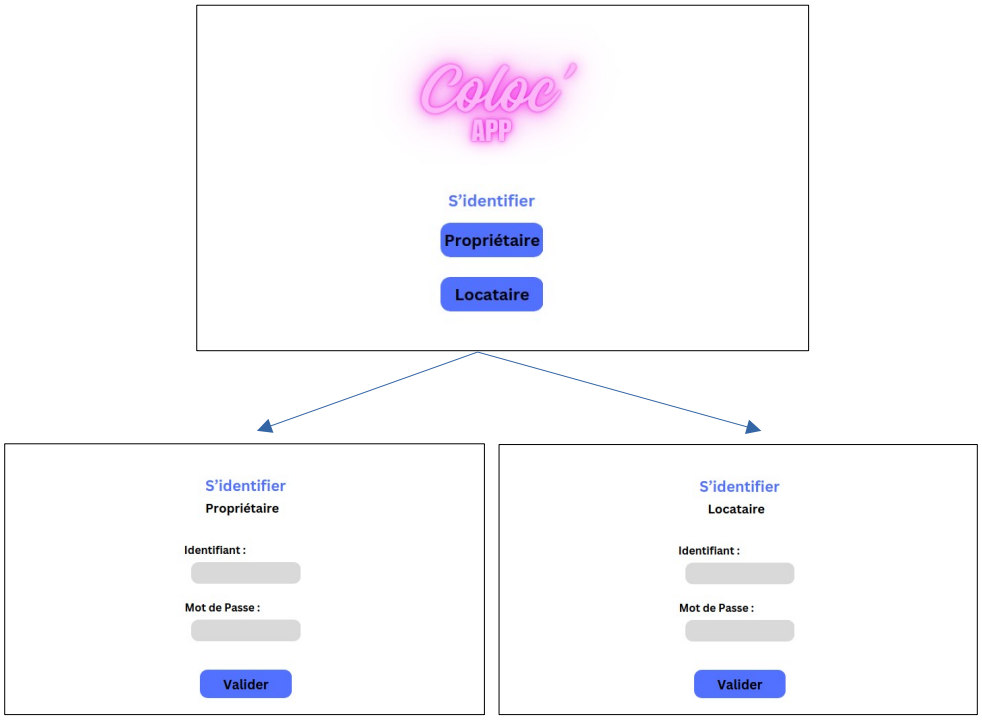
Toutes ces actions nécessitent que le loueur se soit préalablement connecté sur l'application et à tout moment le loueur peut se déconnecter. Nous ne l'avons pas mis sur le diagramme pour ne pas le surcharger.

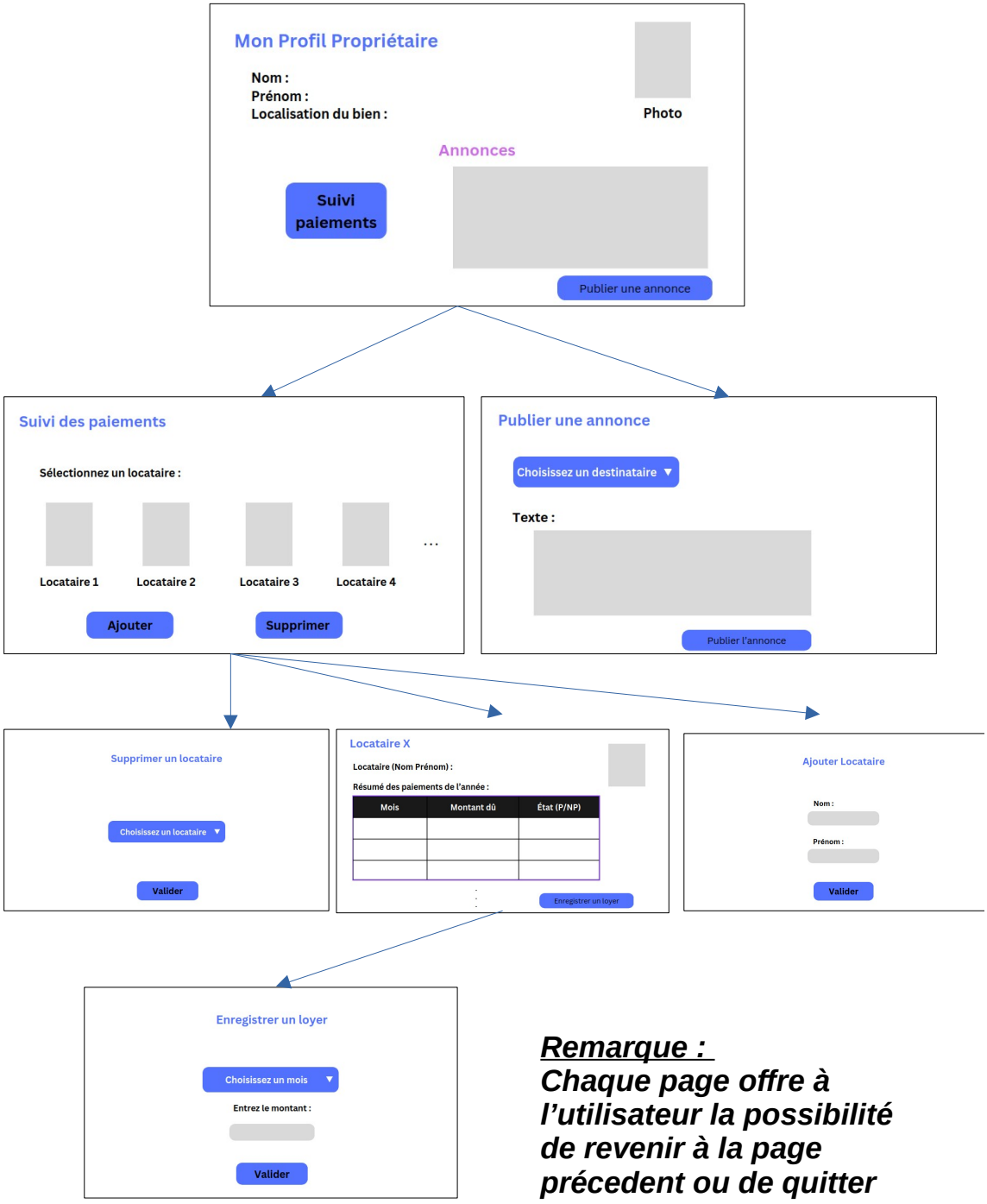
b. . . .du colocataire



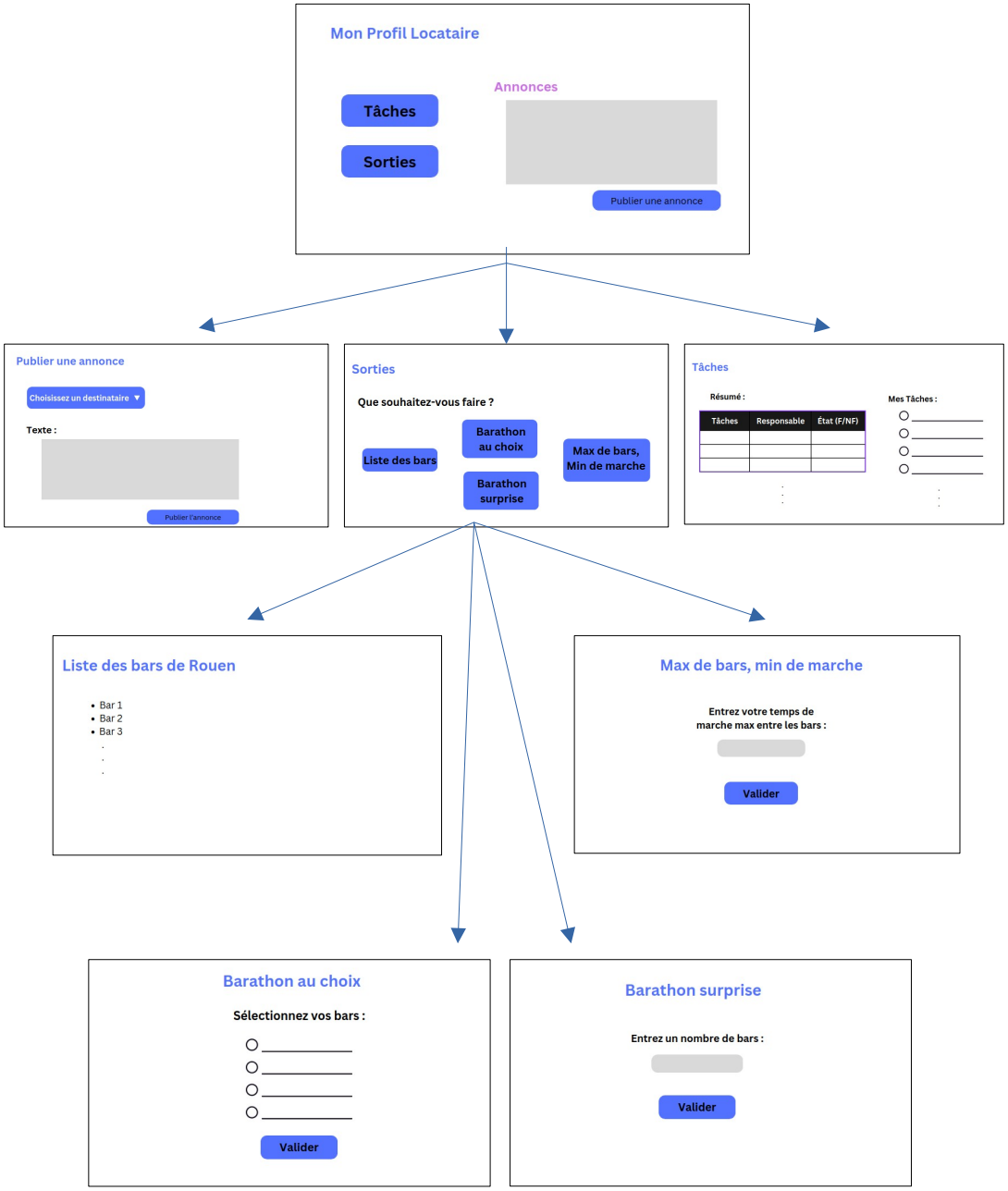
II. Navigation dans les menus

Le diagramme ci-dessous résume l'architecture de notre logiciel, avec les différents menus ainsi que les issues qu'ils proposent. Bien entendu, il est possible de quitter et revenir en arrière pour chacune des pages. Nous verrons cette fonctionnalité dans les captures d'écran de notre démonstration.





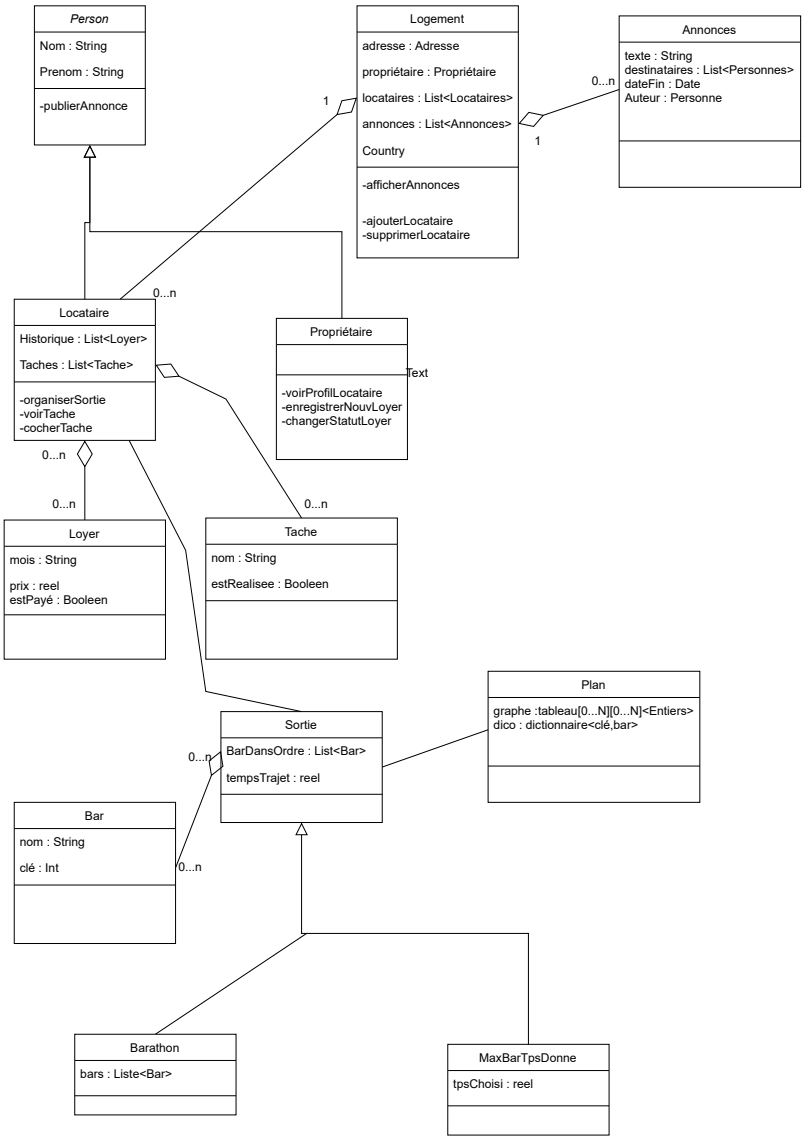
<u>Remarque :</u>
Chaque page offre à l'utilisateur la possibilité de revenir à la page précedent ou de quitter l'application


Mon Profil Locataire
Tâches
Sorties
Annonces
Publier une annonce
Publier une annonce
Choisissez un destinataire
Texte :
Publier l'annonce
Sorties
Que souhaitez-vous faire ?
Liste des bars
Barathon au choix
Barathon surprise
Max de bars, Min de marche
Tâches
Résumé :
Tâches
Responsable
État (F/NF)
Mes Tâches :
Liste des bars de Rouen
Bar 1
Bar 2
Bar 3
Max de bars, min de marche
Entrez votre temps de marche max entre les bars :
Valider
Barathon au choix
Sélectionnez vos bars :
Valider
Barathon surprise
Entrez un nombre de bars :
Valider

III. Diagrammes de classes

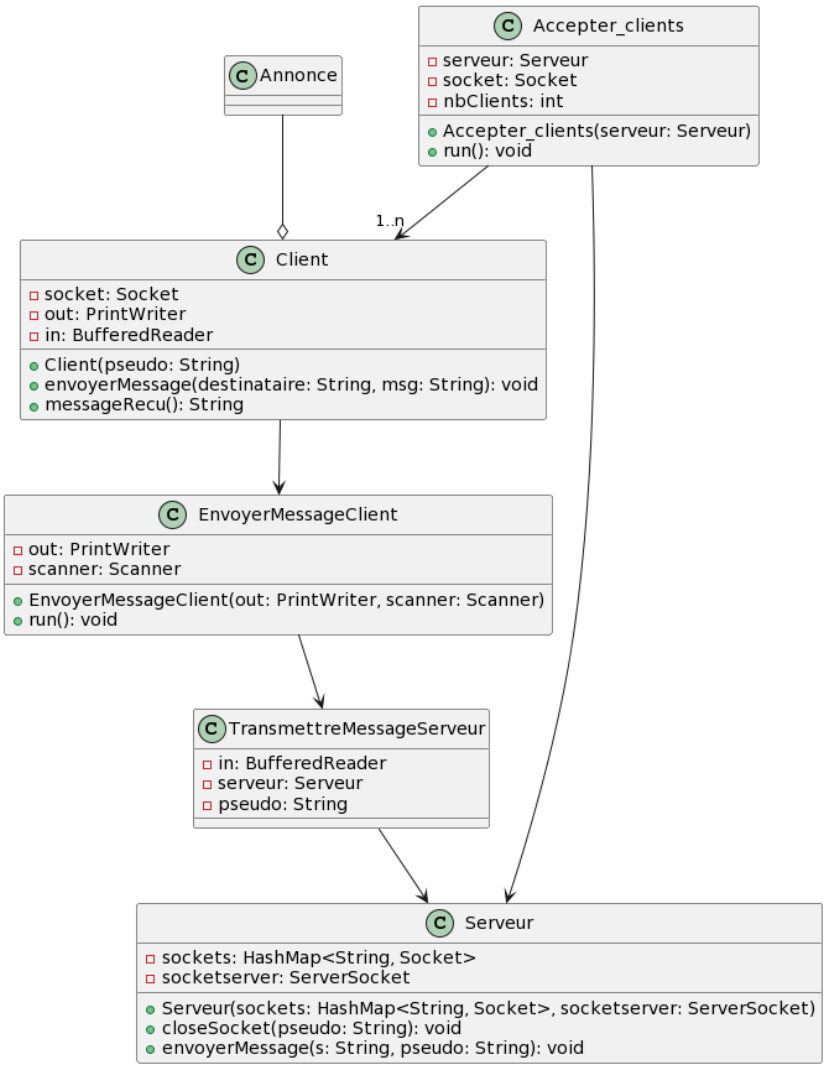
a. Version 1

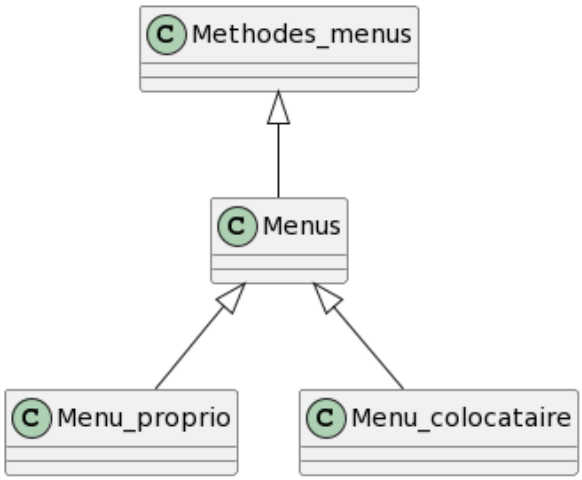
Le diagramme ci-dessous correspond à notre première version, la plus « intuitive »et qui ne prend pas en considération de nombreux éléments. En débutant la programmation, nous avons constaté que notre diagramme de classe initial figurant plus haut ne répondait pas aux besoins. Le diagramme étant obsolète nous ne fournirons pas plus d'explications dessus. En revanche, nous avons réalisé une seconde version (cf. III.b.).



b. Version 2


Loyer
mois: String
montant: double
estPaye: boolean
Loyer(mois: String, montant: double, estPaye: boolean)
TableauPaiements
loyers: List<Loyer>
TableauPaiements()
ajouterLoyer(loyer: Loyer): void
Sortie
banqueDeDonnéeBar(nomBar: String): ArrayList<String>
BarsLesPlusCourtChemin(filtreBar1: String, bar2: String): ArrayList<String>
listeBars(): String[]
supprimerPseudo(pseudo: String): String
envoyerMessagePrivés: String, pseudoDestinataire: String, pseudoExpediteur: String): void
envoyerAnnonce(annonce: String, destinataire: String): void
Graphe
noeuds: ArrayList<Noeud>
aretes: String
ajouterNoeud(ici: Noeud): void
barDuGraphe(noeuds: ArrayList<Noeud>): ArrayList<String>
afficherVoisin: Noeud): void
estVoisin(ici: Noeud): void
obtenirCoutEntreNoeuds(noeudCourant: Noeud, voisin: Noeud): double
obtenirNoeudParNom(nom: String): Noeud
obtenirVoisins(noeudCourant: Noeud, graphe: Graphe): ArrayList<Noeud>
Bars
nom: String
Bars(nom: String)
getNom(): String
Personne
nom: String
prenom: String
identifiant: String
motDePasse: String
Personne(prenom: String, nom: String)
creerIdentifiant(): void
sauvegarderPersonne(nomFichier: String): void
verifierConnexion(identifiant: String, motDePasse: String): boolean
chargerPersonne(nomFichier: String): Personne
Colocataire
toDoList: ArrayList<Tache>
annonces: ArrayList<Annonce>
sortiesOrganisees: ArrayList<Sortie>
participantA: ArrayList<Colocataire>
tempsTrajets: double
tableauPaiements: TableauPaiements
tache: Tache
Colocataire(nom: String, prenom: String)
getToDoList(): List<Tache>
setToDoList(toDoList: List<Tache>): void
getAnnonces(): List<Annonce>
setAnnonces(annonces: ArrayList<Annonce>): void
tacheFini(tache: Tache): void
ajouterAnnonce(annonce: Annonce): void
Tache
description: String
fait: boolean
responsable: Colocataire
Tache(description: String, fait: boolean, responsable: Colocataire)
ajouterTacheTout(tache: Tache): void
supprimerTacheTout(tache: Tache): void
ajouterTacheColoc(tache: Tache, colocataire: Colocataire): void
supprimerTacheColoc(tache: Tache, colocataire: Colocataire): void
tacheFiniPourColoc(tache: Tache, colocataire: Colocataire): void
chargerTaches(nomFichier: String): ArrayList<Tache>
Proprietaire
listeLocataires: ArrayList<Colocataire>
tableauPaiements: TableauPaiements
annonces: ArrayList<Annonce>
adresse: String
Proprietaire(nom: String, prenom: String, listeLocataires: ArrayList<Colocataire>, tableauPaiements: TableauPaiements, annonces: ArrayList<Annonce>)
Proprietaire(nom: String, prenom: String)
setListeLocataires(listeLocataires: ArrayList<Colocataire>): void
setAnnonces(annonces: ArrayList<Annonce>): void
afficherProfil(): void
Annonce
destinataires: List<Client>
contenu: String
dateDePublication: LocalDate
dateExpiration: LocalDate
Annonce(contenu: String, destinataires: List<Client>, dateDePublication: LocalDate)
ajouterDestinataire(client: Client): void
setDateDePublication(dateDePublication: LocalDate): void
setDestinataires(destinataires: List<Client>): void
setContenu(contenu: String): void
setDateExpiration(dateExpiration: LocalDate): void
getDateExpiration(): LocalDate
convertirAnnonceEnChaine(): String
chargerAnnonce(nomFichier: String): Annonce
supprimerAnnoncesPerimees(annonces: List<Annonce>): void
0..n
1
1..n
1..*


Accepter_clients
serveur: Serveur
socket: Socket
nbClients: int
Accepter_clients(serveur: Serveur)
run(): void
Annonce
1..n
Client
socket: Socket
out: PrintWriter
in: BufferedReader
Client(pseudo: String)
envoyerMessage(destinataire: String, msg: String): void
messageRecu(): String
EnvoyerMessageClient
out: PrintWriter
scanner: Scanner
EnvoyerMessageClient(out: PrintWriter, scanner: Scanner)
run(): void
TransmettreMessageServeur
in: BufferedReader
serveur: Serveur
pseudo: String
Serveur
sockets: HashMap<String, Socket>
socketserver: ServerSocket
Serveur(sockets: HashMap<String, Socket>, socketserver: ServerSocket)
closeSocket(pseudo: String): void
envoyerMessage(s: String, pseudo: String): void


Methodes_menus
Menus
Menu_proprio
Menu_colocataire

Le diagramme tient sur 2 pages : la première représente les classes plutôt « logiques »se rapportant aux différentes actions possibles sur l'application que l'on retrouve globalement dans notre première version ; la seconde illustre les parties IHM et réseau où la classe *Annonce* est à nouveau présente puisque c'est la seule possédant des relations avec les classes de la partie réseau.

Pour la partie connexion, on suppose que le (seul) propriétaire remet à ses locataires leur identifiant et mot de passe, qui sont stockés dans un fichier avec les informations principales du locataire.

En ce qui concerne la partie « Sorties », le plus court chemin entre deux bars choisis est obtenu grâce à l'algorithme de *Dijkstra*. Quant à la génération du barathon (tournée de tous les bars avec choix du départ), nous nous sommes inspirés du problème du voyageur de commerce et des algorithmes permettant de le résoudre. Nous avons choisi d'implémenter l'algorithme des fourmis car le concept de cet algorithme, assez différent des autres algorithmes utilisés pour ce problème, a capté notre attention et de plus c'est un algorithme pouvant s'appliquer à des graphes de grande taille. Cet algorithme s'inspire du comportement des fourmis qui, lors de leur approvisionnement en nourriture, passeront toutes au bout d'un certain temps par le chemin le plus court entre la fourmilière et la nourriture, et ceci grâce au dépôt de phéromones sur le chemin. L'algorithme simule donc un certain nombre de fourmis, qui, pour un nombre donné d'itérations, vont se déplacer dans le graphe avec une certaine probabilité de prendre l'une ou l'autre des arrêtes (probabilité dépendant de la distance et des phéromones présentes sur cette arrête). Ensuite le plus court chemin sera mis à jour si une fourmi trouve un chemin plus court que celui étant actuellement le plus court. Après chaque passage d'une fourmi sur un chemin, une quantité de phéromones dépendant de la longueur du chemin est rajoutée sur chacunes des arrêtes de ce chemin. Puis après chaque itération une certaine proportion de phéromones s'évaporent pour améliorer la vitesse de convergence de l'algorithme. D'ailleurs rappelons bien que c'est un algorithme convergent (des preuves existent à ce sujet) donc le choix du nombre de fourmis et du nombre d'itérations influe sur le fait que le chemin retourné soit plus ou moins proche du chemin optimal. Nous avons du adapter cet algorithme car les pseudo-codes que nous trouvions étaient faits pour des graphes complets. Pour ce faire nous avons modifié la fonction « coutEntre2Bars »de notre classe graphe pour qu'elle retourne un grand nombre (1900000) si les deux bars entre lesquels on demande le temps ne sont pas voisins. C'est plutôt une bonne solution car la probabilité que les fourmis prennent cette arrête inexistante est alors très très faible, elle ne prendront donc pas ou très peu cette arrête qui de toute façon augmente-

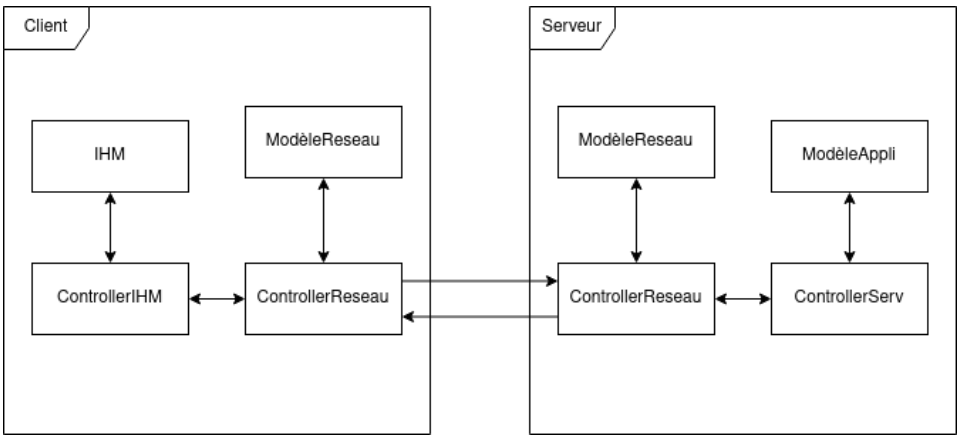
rait beaucoup le coût total du chemin et serait vite oublier. L'algorithme que nous avons implémenté trouve un chemin passant par tous les bars (et ne revenant pas au bar de départ) d'une durée de 30 ou 31 minutes, durée variable dépendant du point de départ. Nous tenions aussi à préciser, que, parfois, pour une raison encore obscure à nos yeux, le chemin retourné passe deux fois par le bar de départ. Dans ce cas, le chemin retourné est quand même optimal mais il passe par tous les bars sauf un.

Notons que les différentes données (liste des colocataires, des paiements, des bars, ...) sont stockées dans des fichiers.

Le système d'annonces, équivalent à un système de messagerie constitue le coeur de notre travail sur le réseau. L'échange ne se fait pas directement entre clients, mais tout passe par le serveur qui récupère l'annonce d'un client pour ensuite la « redistribuer »au(x) destinataire(s). En effet, l'auteur de l'annonce aura le choix entre l'envoyer à l'un des acteurs de la colocation (propriétaire compris) ou à tous.

Enfin, l'IHM est gérée par les classes *Methodes_menus*, *Menus*, *Menu_proprio* et *Menu_proprio*. L'existence de la classe *Methodes_menus* est justifiée par la présence de certaines méthodes et de paramètres « basiques »communs à la gestion de tous les affichages, surchargeant les autres classes. Nous avons donc préféré les regrouper dans une classe qui leur est dédiée.

IV. Diagramme d'architecture réseau



Ce diagramme représente une première modélisation de l'architecture réseau. Au final, afin de retirer une couche de complexité, nous avons décidé de ne pas le respecter.

En effet, la partie réseau implémentée actuellement ne concerne que l'envoi des annonces entre les applications. La partie serveur est donc très légère, elle ne fait que

relayer l'information entre les différents clients.

3. Exemples d'utilisation de Coloc'APP

Après le lancement de l'application, voici la première page sur laquelle on tombe :

On choisit alors de se connecter sur l'espace propriétaire ou l'espace colocataire.
Chaque profil possède un identifiant et un mot de passe de la forme suivante :

- Identifiant : *PrénomNom* ;
- Mot de passe : *PrénomNom123*.

De plus, 5 profils sont pré-enregistrés dans l'application :

- Propriétaire : Prénom = « Jean », Nom = « Michel » ;
- Colocataire 1 : Prénom = « Paul », Nom = « Méhaud » ;
- Colocataire 2 : Prénom = « Kenza », Nom = « Bennani » ;
- Colocataire 3 : Prénom = « Constance », Nom = « Bau » ;
- Colocataire 4 : Prénom = « Ines », Nom = « Silhadi ».

Connectons-nous d'abord du côté du propriétaire :

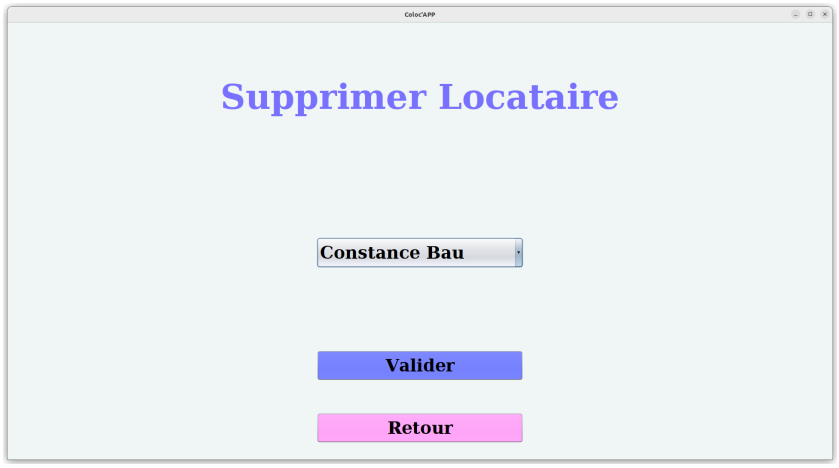
Après la connexion, on se retrouve sur la page du profil du propriétaire :

Allons suivre les paiements.

Plusieurs options s'offrent à nous :

- Ajouter un colocataire :

- Supprimer un colocataire :



- En revenant sur la page du suivi des paiements, on choisi un colocataire dans la liste déroulante et en appuyant sur valider, voici la page qui s'affiche :

Ensuite, on retourne sur la page du profil et on se dirige vers l'envoi d'annonces : On choisi le destinataire dans la première liste déroulante, on écrit le message dans le champ du dessous et les messages reçus s'affichent dans le champ du bas.

Ici, on s'est envoyé un message à soi-même.

Dirigeons-nous maintenant vers le profil du colocataire :

On peut suivre les tâches du colocataire :

On peut également organiser des sorties :

Affichons la liste des bars :

Nous pouvons organiser 2 types de sorties :

- Une tournée des bars : on sélectionne un bar et l'application nous ressort le plus court chemin partant de ce bar qui parcourt tous les bars ;
- Un plus court chemin entre 2 bars : on sélectionne deux bars et l'application nous ressort le plus court chemin entre ces deux bars.

Voici un exemple de tournée des bars, partant de la Taverne de Thor :

Une petite précision est à apporter à cet exemple d'utilisation : les parties sur le suivi des loyers et le suivi des tâches sont implémentées dans la partie logique mais n'ont pas pu être encore reliés dans l'IHM car nous avons préféré nous concentrer sur les parties obligatoires de ce projet, à savoir l'algorithme de parcours de graphes et la partie en réseau.

4. Organisation du projet

I. Répartition du travail

Nous avons tout d'abord réfléchi ensemble sur notre façon de voir l'application puis nous avons réalisé les différents diagrammes ensemble. Ensuite nous nous sommes répartis le travail de la façon suivante :

- Ines et Kenza se sont occupées principalement de la partie réseau et des classes « logiques »gérant les acteurs de la colocation, les tâches, loyers, etc. Elles ont beaucoup travaillé en lien avec Paul pour progressivement intégrer leur code à l'interface graphique ;
- Constance s'est chargée de tout ce qui concerne les graphes (modélisation du graphe, chargement du graphe des bars, algorithme de Dijkstra et algorithme des fourmis) ;
- Paul s'est chargé de l'interface graphique IHM avec l'aide de Kenza, puis, en tant que véritable ciment de notre groupe il nous a donné des conseils sur l'encapsulation de notre code pour qu'au final il puisse facilement lier nos codes ensemble et les intégrer progressivement à l'interface graphique.

Globalement, bien que les tâches aient été reparties, le travail a été fait en groupe sans déséquilibre : nous nous sommes vus très souvent, nous sommes entraidés et avons tous « touché »à tous les aspects du projet. De plus, le travail en simultané a été très fluide grâce à l'utilisation de **gitlab**.

II. Rétrospective

Le logiciel a été développé en prenant en compte les contraintes matérielles et logicielles. Du point de vue matériel, le logiciel est compatible avec des ordinateurs personnels standards, garantissant ainsi une accessibilité facile pour les utilisateurs. Du côté logiciel, le logiciel a été conçu pour fonctionner sur différentes plates-formes, offrant une flexibilité d'utilisation. Les exigences minimales incluent une connexion Internet stable

pour permettre l'accès aux fonctionnalités réseau. Les objectifs que nous nous étions fixés ont été globalement atteints.

Les contraintes que nous avons rencontrées se trouvent principalement dans la partie réseau qui a été sans aucun doute la partie la plus délicate à aborder puisque la mise en lien avec la partie IHM a posé problème. Nous avons du modifier la classe « Client »mais aussi ajouter la classe « Méthodes-menus ».

Résumons à présent les points sur lesquels nous devons nous pencher à l'avenir. Concernant les tâches ménagères, la gestion individuelle de to-do-list fonctionne correctement mais il nous reste à nous charger de la centralisation de celles-ci, afin de permettre à tous les colocataires d'avoir une vue globale des tâches.

Nous aurions également souhaité faire en sorte que l'ajout d'un colocataire par le propriétaire entraîne automatiquement son ajout dans la liste de destinataires proposée lors de l'envoi d'une annonce.

Un autre point à améliorer est le lien avec la partie de gestion fichier et l'IHM. En effet, nous disposons déjà des méthodes de gestion de fichier que ce soit pour la sauvegarde d'un acteur, la liste des tâches et aussi le tableau de paiements. Pour cette partie, nous avons pensé à créer un fichier spécifique à chaque personne. Cette méthode pourrait nous faciliter l'ajout et la suppression d'un colocataire par exemple, puisqu'il suffira simplement de supprimer le fichier spécifique à la personne au lieu d'effectuer une recherche dans plusieurs fichiers.

Enfin, à l'heure actuelle, l'application est adaptée à l'usage d'un seul propriétaire, pour un seul logement. Il serait intéressant d'envisager une application ouverte à plusieurs propriétaires qui peuvent posséder plusieurs biens.

5. Conclusion

En conclusion, ce projet a été une expérience enrichissante, mettant en œuvre divers aspects de la programmation, de la conception réseau et de l'algorithmique des graphes. L'objectif principal était de créer un outil complet et intuitif, capable de répondre aux besoins des propriétaires et des colocataires.

Nous avons traversé des défis significatifs, notamment dans l'implémentation de la communication réseau et la modélisation des graphes. La collaboration étroite entre les membres de l'équipe a donc été cruciale pour surmonter ces obstacles, chacun apportant sa contribution .

En perspective, des améliorations futures peuvent être envisagées, telles que l'extension de l'application pour gérer plusieurs propriétaires possédant plusieurs biens, ainsi que l'optimisation de certaines fonctionnalités, notamment la centralisation des tâches ménagères.