Ride Quest

Checkpoint 2

Projet Transverse I

Gogniat Aaron

Table des matières

Introduction au projet

À la suite du cours de Design Science du Professeur Jean-Henry Morin que j'ai suivi le semestre passé (Automne 2023), j'ai rendu un projet de recherche intitulé « Ride-Quest : a proposal for a mobile application to push people to travel using a ride sharing and public transportation system ». Le but de ce projet a été de penser une application mobile qui permettrait à une personne d'atteindre sa destination à l'aide de co-voiturage et des transports publics en une course, dans le but d'inciter les personnes a utilisé plus régulièrement les transports en commun. Les passagers n'auraient pas à payer pour ce service mis a part leur billet de transport en commun, et les conducteurs se verraient récompenser de diverses manières. J'ai donc rendu une proposition d'un système qui permettrait en partie l'implémentation d'une telle application. J'ai dans l'idée de continuer cette proposition pendant ce cours de Projet Transverse 1 en sélectionnant le projet n°10 (« Système de réservation de taxi ») car celui-ci permet un accomplissement partiel de mon objectif final, qui est de proposer une application entièrement implémentée et fonctionnelle pour mon projet de Bachelor.

Description du projet et de sa problématique

L'objectif du projet est donc de créer un système qui matcherait un conducteur au passager qui a fait une demande pour une course. Le conducteur doit être choisi parmi d'autres conducteurs ayant acceptés la course, doit pouvoir recevoir les informations nécessaire pour conduire le passager à une plateforme de transport public qui pourrait ensuite l'amener à la destination définie, de le rejoindre à la station convenue pour ensuite le conduire à la destination renseignée, ou de le conduire de l'origine à la destination si aucune option de transport en commun ne peut être trouvé ou si le passager a choisi de ne pas utiliser ce type de transport.

En ce qui concerne ce projet, il va falloir designer et implémenter un système de réservation de covoiturage. Pour cela, il sera nécessaire d'implémenter un API qui permettra le choix d'un itinéraire ainsi qu'un algorithme de triage selon des critères de sélection qui sera utilisé pour trouver le conducteur correspondant le plus au passager. Il faudra également récolter les données GPS des utilisateurs en tout temps pour permettre de les afficher sur une carte ainsi que de les transmettre à leur pair pour permettre la bonne réalisation de la course.

Analyse des objectifs (MAJ)

Modèle des objectifs

Le modèle des objectifs permet d'identifier, explorer, évaluer et opérationnaliser les objectifs organisationnels et stratégiques [1].

Nous pouvons définir trois objectifs principaux : permettre la réservation d'une course (Objectif 1), réunir un conducteur et un passager (Objectif 2) ainsi qu'organiser un itinéraire (Objectif 3). Même si ces deux derniers objectifs sont visualisés comme des sous-objectifs, ceux-là n'en restent pas du moins très importants. Ces trois objectifs formant la base de ce système, nous pouvons ensuite décrire les sous-objectifs. Les sous-objectifs pertinents à mentionner sont « Permettre au passager de placer une demande de course » (Objectif 18), « Permettre à l'utilisateur de choisir entre être conducteur et être passager » (Objectif 15), « Implémenter un algorithme d'itinéraire » (Objectif 8), « Suivre en temps réels les coordonnées GPS des

utilisateurs » (Objectif 13), « Implémenter un algorithme de matching » (Objectif 9) et « créer une base de données » (Objectif 12).

En ce qui concerne l'implémentation d'un algorithme d'itinéraire (Objectif 8), trois opportunités ont été soulevées. En effet, les APIs mis à notre disposition par les différents services de transport en commun ainsi que Google Maps (Opportunité 1-3) permettront de trouver l'itinéraire le plus cohérent aux différentes informations récoltées.

Trois obligations ont été relevées. Il s'agit d'obligations légales concernant l'âge et les papiers légaux des utilisateurs. Ainsi, le conducteur doit être en âge de conduire (Obligation 1), être muni d'un permis de conduite valable (Obligation 2) et le passager doit avoir âge légal pour utiliser un tel service (Obligation 3).

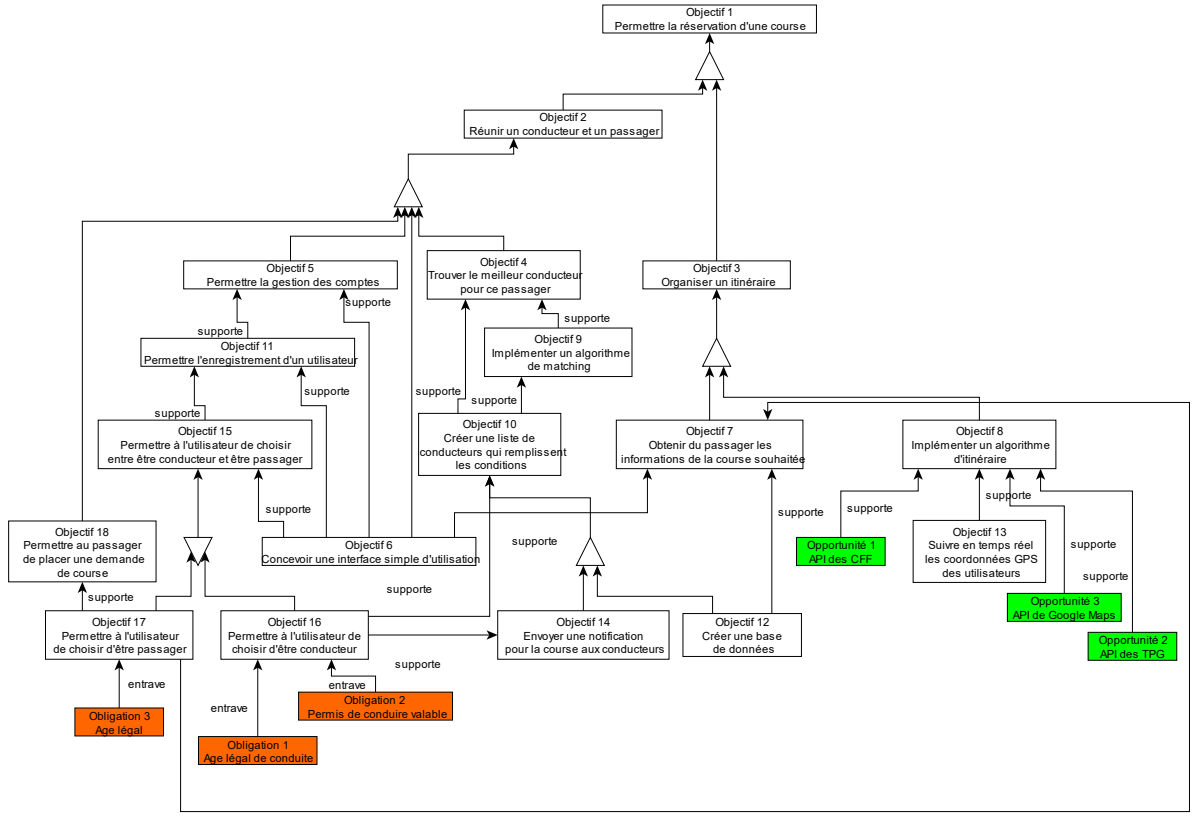

Figure 1 : Modèle des Objectifs

Liste des acteurs

La liste des acteurs nous permet d'identifier les différents membres du système.

Nous n'avons qu'un type général d'acteur, c'est-à-dire l'utilisateur de notre système. Celui-ci peut être spécialisé en conducteur, qui offre une course, et passager, qui demande une course.

Modélisation des activités

Le modèle des activités permet une identification, exploration et développement des activités [1].

Par soucis de lisibilité, nous avons pu distinguer 5 modèles des activités : le processus du conducteur, du passager, de l'algorithme de sélection, de l'envoi de la course ainsi que de la course. Chacun modélise une partie et/ou des acteurs différents allant de la mise à disposition d'une course par le conducteur au déroulement de la course en elle-même.

Processus du conducteur

Le premier modèle est celui du conducteur. Le modèle des activités du conducteur définit les étapes et processus que celui-ci doit traverser pour pouvoir se rendre disponible pour une course.

Avant que le conducteur puisse offrir une course, ce dernier doit s'enregistrer s'iel n'a pas de compte (Processus 1) ou se connecter s'iel en a un (Processus 2). Dès que cela est fait, iel doit choisir son rôle de conducteur (Processus 3), entrer les informations du véhicule qu'il utilisera (Processus 3.5)(Info 7-9) et une fois cela fait, le conducteur verra attendre une course (Processus 5) et sera ainsi définit comme « disponible » (Info 11). Ce dernier peut se déconnecter (Processus 7) et se verra donc retirer son rôle de conducteur (Info 18).

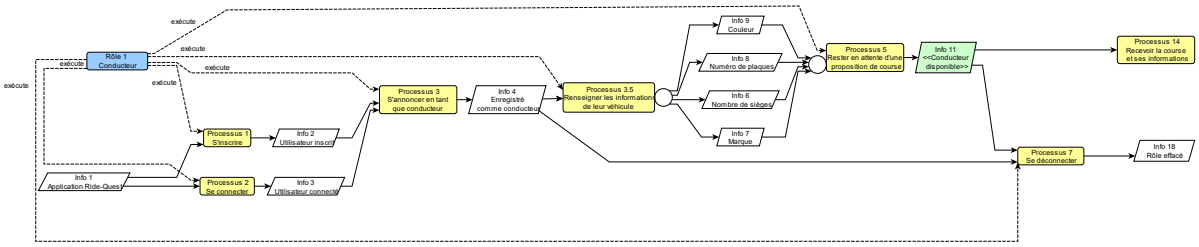

Figure 2 : Modèle des Activités du Conducteur

Processus du passager

Pour ce qui est du passager, celui-là suit les mêmes processus pour s'annoncer en temps que passager que le conducteur (Processus 1,2 et 4). Pour demander une course (Processus 6), iel doit entrer les informations de la course souhaitée (Processus 8)(Info 12-17) et une fois que cela est fait et l'itinéraire va être créé (Processus 9) à l'aide des différentes APIs (Ressource NH 1)(Ressource Ext 1-3). Le passager, comme le conducteur, peut se déconnecter en tout temps (Processus 7), ce qui lui effacera son rôle de passager (Info 16) et supprimera la demande de course (Info 19).

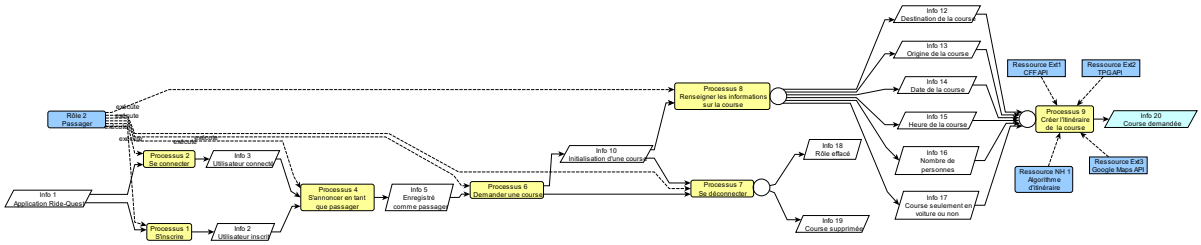

Figure 3 : Modèle des Activités du Passager

Processus de l'algorithme de sélection des conducteurs

Une fois la course demandée (Info 20), l'algorithme de sélection des conducteurs sera déclenché. Celui-ci créé donc une liste des conducteurs (Processus 10), analyse des informations des conducteurs obtenues par la base de données (Processus 28) en se focalisant sur le nombre de sièges (Info 6), la distance entre le conducteur et le passager (Info 42) et si la disponibilité du conducteur (Info 11). Grâce à cette analyse, l'algorithme peut donc choisir les conducteurs qui conviennent et les mettre dans la liste (Processus 29). Une fois cette liste remplie avec tous les conducteurs disponibles (Info 43), les conducteurs inscrits sont ensuite juger (Processus 30) de par leur nombre de sièges, leur distance au passager, et leur rating (Info 6,44 et 45) pour pouvoir les classer selon un ordre de priorité (Processus 31). La liste mise à jour est dès lors retenue une fois ce classement terminé (Info 22).

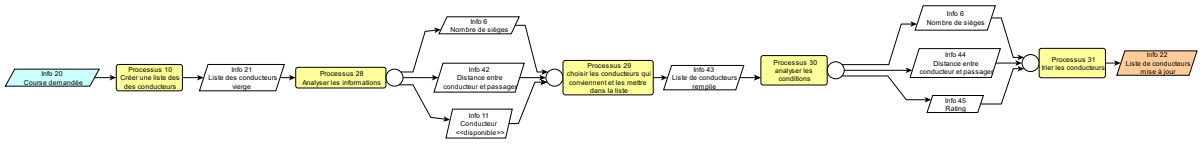

Figure 4 : Modèle des Activités de l'algorithme de sélection

Processus de l'envoi de la course

Grâce à cette liste de conducteurs (Info 22), il est ensuite possible de regarder si le premier conducteur inscrit dans la liste est toujours disponible (Processus 12). Si ce dernier n'est pas disponible (info 24), il est retiré de la liste (Processus 20), donnant ainsi une liste de conducteurs de nouveau mise à jour (Info 22) et cela recommande jusqu'à ce qu'un conducteur soit disponible (Info 23). La course lui est ensuite envoyée (Processus 13)(Info 25) et son statut change pour devenir « en attente » (Info 41). Une fois qu'iel la reçu (Processus 14), iel peut choisir entre accepter la course (Processus 15) ou alors la rejeter (Processus 16). S'iel décide d'accepter la course, le conducteur et choisi et le prochain processus prend le relais. Au contraire, s'iel l'ignore, le conducteur sera effacé de la liste (Processus 20). Si la liste est vide (Info 32), signifiant que le dernier conducteur de la liste a été supprimé, la course l'est également (Processus 21). Les deux partis peuvent en tout temps se déconnecter pour arrêter le processus (Processus 7)

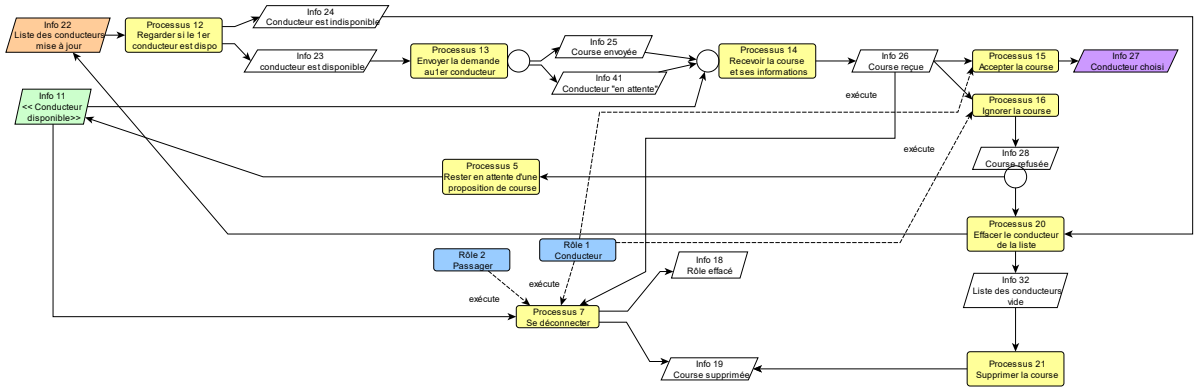

Figure 5 : Modèle des Activités de l'envoi de la course

Processus de la course

Dès que le conducteur a été choisi (Info 27), la liste est supprimée (Processus 19) et le passager et le conducteur sont prévenus du matching (Processus 17-18). La course peut ainsi commencer (Processus 22). Celle-ci peut commencer soit par le covoiturage (Processus 23), soit par transport public (Processus 24) et peut se continuer de l'autre manière. Une fois la course terminée (Processus 25), l'utilisateur peut noter sa paire (Processus 27). L'utilisateur peut également signaler l'autre personne (Processus 26).

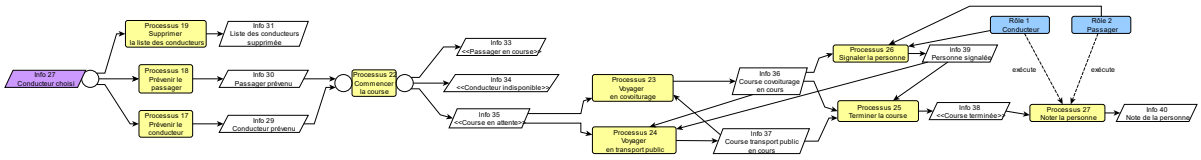

Figure 6 : Modèle des Activités de la course

Modélisation des règles de gestion

Ce modèle permet une définition des règles de gestions, qui elles-mêmes servent à structurer, coordonner, contraindre, contrôler et influencer les activités de différents services et acteurs [1].

Modèle des objectifs

Les règles de gestion se rattachant au modèle des objectifs viennent au nombre de cinq. Tout d'abord, une règle peut être ajouté à l'Objectif 2 qui stipule que le conducteur ne peut être lié qu'à un seul passager (Règle 1). Une autre règle peut être définie quant à l'Objectif 8 qui prescrit de trouver l'itinéraire le plus efficace (Règle 2). Une troisième règle liée l'Objectif 7 peut être également ajouté qui ordonne que les informations soient valables (Règle 3). L'Objectif 12 se voit suivit d'une règle qui indique que la base de données doit contenir toutes les informations liées aux utilisateurs et aux courses (Règle 4). Une dernière règle peut être mentionnée, qui ajoute à l'Objectif 9 le fait que le matching doit se faire en prenant compte de conditions définies (Règle 5).

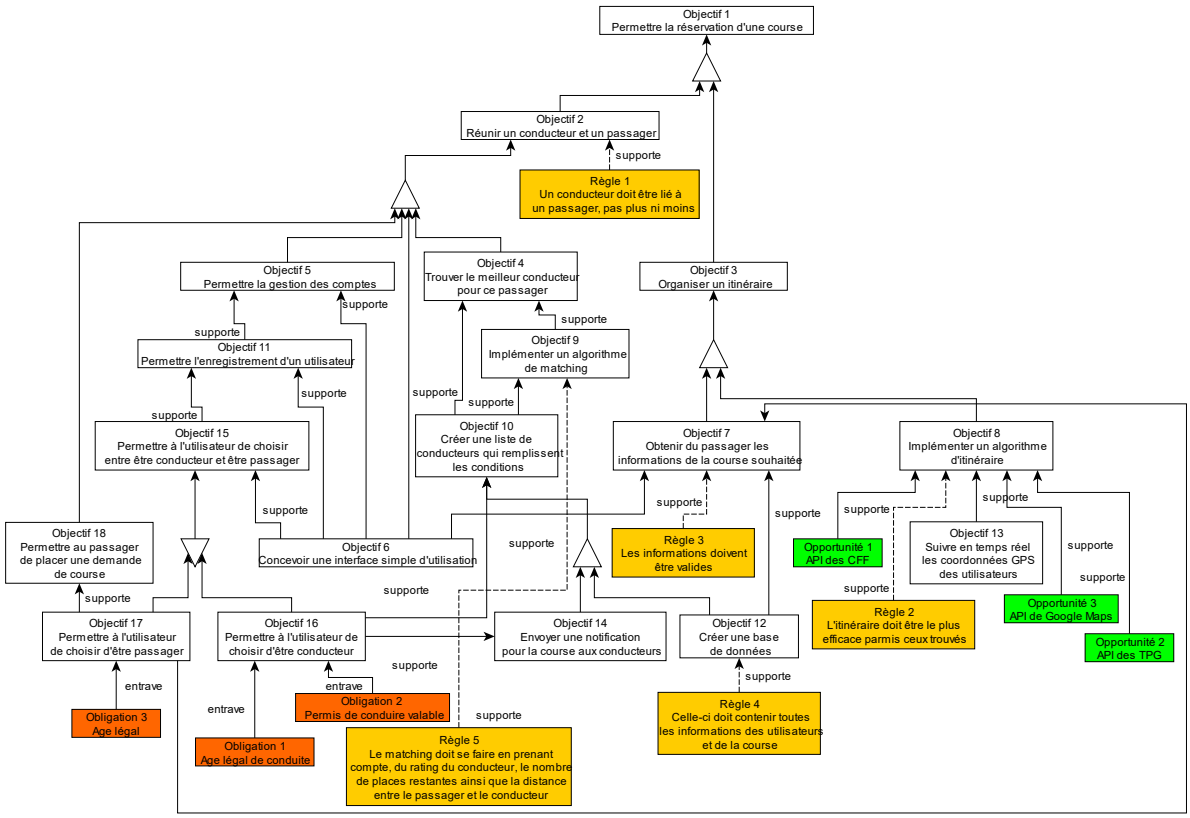

Figure 7 : Modèle des Règles de gestion des objectifs

Modèles des activités

Un total de 27 règles peut être recensé au travers des différents modèles des activités : les règles rattachées à l'utilisateur en général (Règle 1-3,5,15-16), au conducteur (Règle 4,21-22,25-27), au passager (Règle 12,22), à la course (Règle 6-9,12-14,17-22,24) et au système (Règle 9,24).

Processus du conducteur

Ce processus ne contient que 5 règles. 3 de celles-ci sont des contraintes posées à l'utilisateur (Règle 1-3) et une contrôlant que le conducteur ait un permis de conduire (Règle 4), et une dernière indiquant que lorsqu'un utilisateur se déconnecte, son rôle est supprimé de la base de données (Règle 5).

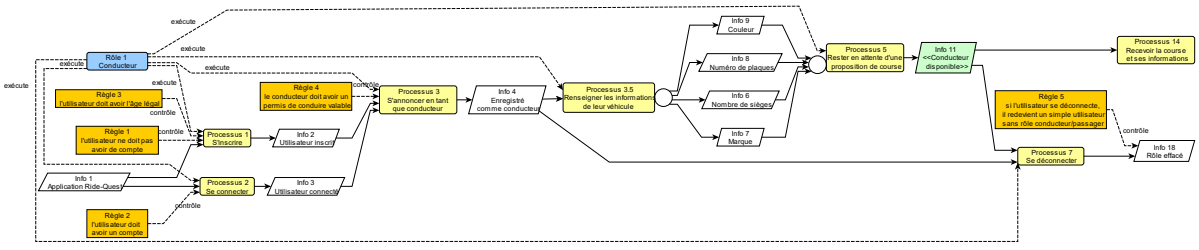

Figure 8 : Modèle des Règles de gestion des Activités du conducteur

Processus du passager

Quant au modèle des activités du passager, les 4 premières (Règle 1-3,5) sont les mêmes que celles décrites dans la partie précédente. 3 règles s'y ajoutent : le fait que la course soit supprimée si le passage se déconnecte (Règle 6), que si la destination n'est pas dans le même pays que l'origine, cela doit être signalé (Règle 7) et que les informations doivent être valables (Règle 8).

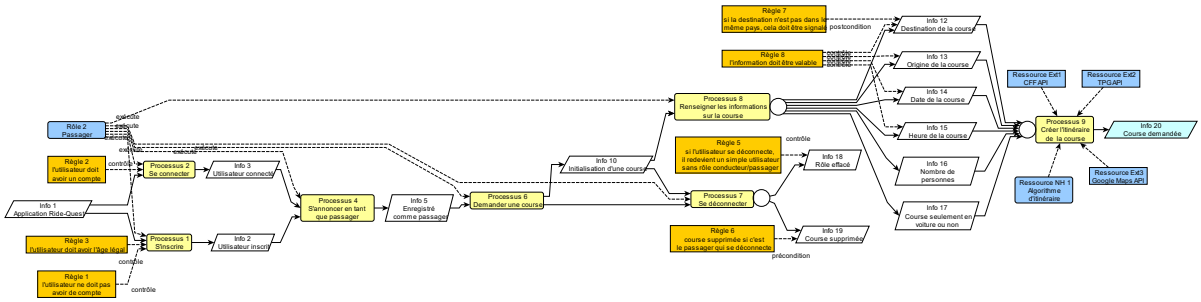

Figure 9 : Modèle des Règles de gestion des Activités du passager

Processus de l'algorithme de sélection des conducteurs

Quant à ce qui est du processus de l'algorithme de sélection des conducteurs, les règles définies se rapportent à la bonne exécution de l'algorithme. Nous pouvons mentionner les règles 21 à 22 qui contrôlent l'analyse des informations des conducteurs pour placer dans la liste ceux qui répondent aux critères, ainsi que 25 à 27 qui vérifient les informations du tri des conducteurs.

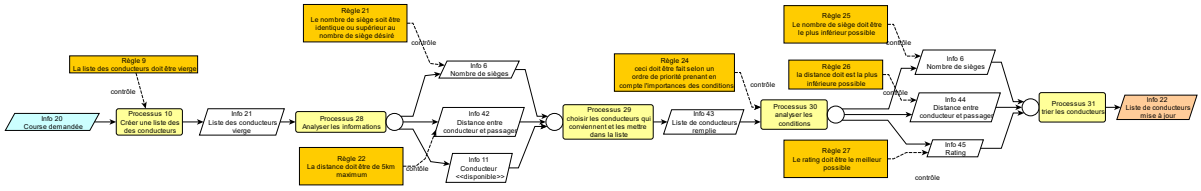

Figure 10 : Modèle des Règles de gestion des activités de l'algorithme de sélection

Processus de l'envoi de la course

Les règles se rapportant au processus d'envoi de la course se comptent au nombre de 3. Les règles 5 et 6, comme décrites précédemment, se rapportent à la déconnexion. Quant à la règle 20, celle-ci se réfère à au fait que si la liste devient vide, cela veut dire qu'aucun conducteur ne correspond aux critères de sélection.

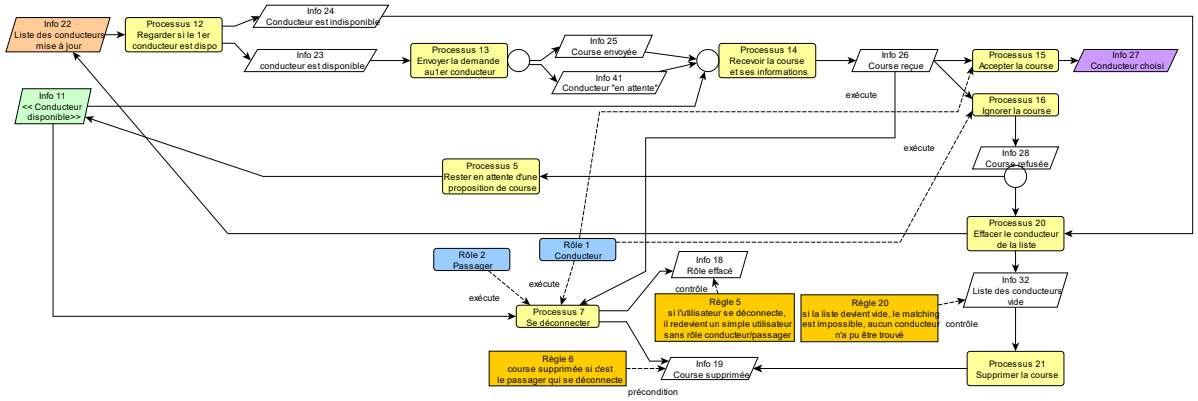

Figure 11 : Modèle des Règles de gestion des Activités de l'envoi de la course

Processus de la course

Les règles du processus de la course indiquent comment la course se déroule, c'est-à-dire qu'en fonction de l'itinéraire reçu et du choix du conducteur, la course peut commencer en transport public et se terminer en covoiturage (Règle 13, 14), vice versa (Règle 16), ou alors du fait que le conducteur ait sélectionné l'option « voyager seulement en covoiturage », tout le trajet se ferait en covoiturage, donc sans prendre les transports en commun (Règle 12, 13). 3 autres règles existent dans ce modèle. Celles-ci se rapportent aux processus de la fin de la course (Règle 19), de signalement d'une personne (Règle 15) et pour finir de notation de la personne (Règle 16).

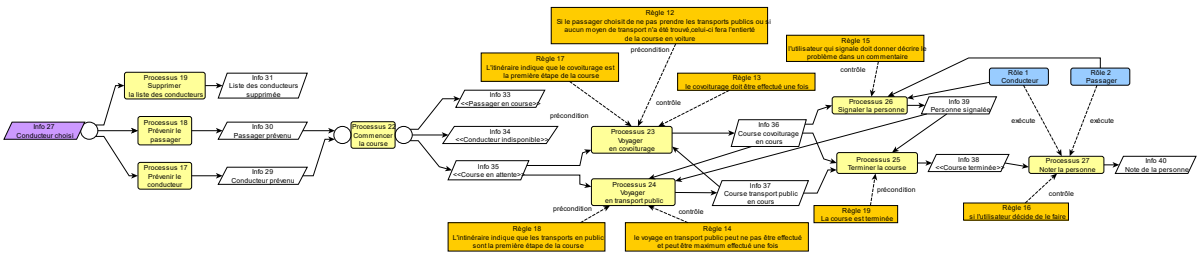

Figure 12 : Modèle des Règles de gestion des Activités de la course

Modélisation des exigences fonctionnelles et non fonctionnelles

Le modèle des exigences permet de décrire les exigences pour le développement d'un service ou d'un système d'information [1].

Beaucoup d'Objectifs SI, d'exigences fonctionnelles et non-fonctionnelles ont été recensés. Nous pouvons en nommer quelques-unes.

L'objectif principal de ce système d'information est le fait de fournir un service de réservation de taxi (Objectif SI1). Celui-ci est accompagné d'une exigence non-fonctionnelles indiquant que le service doit être disponible 24h/24 (Exigence NF1). Cet objectif comprend plusieurs sous-objectifs et exigences : match un passager et un conducteur (Objectif SI2), permettre la gestion des comptes (Objectif SI3) modifier les statuts des utilisateurs et de la course (Exigence F2) ainsi que garder toutes les données dans la base de données (Objectif SI5). Le reste consiste en des objectifs SI, exigences fonctionnelles et non-fonctionnelles servant à appuyer les objectifs SI, les exigences, ainsi que les objectifs et les processus mentionnés dans les modèles précédents.

Parmi les exigences fonctionnelles, certaines peuvent être précisées à l'aide de sous-exigences fonctionnelles. « Entrer les informations de la course » (F7) en contient plusieurs : (i) entrer l'origine (ii) entrer la destination, (iii) entrer la date, (iv) entrer l'heure d'arrivée, (v) entrer le nombre de personnes, et (vi) indiquer si iel veut faire la course seulement en voiture ou également à l'aide de transport public. Toutes les exigences fonctionnelles relatant des statuts se voient toutes être composées de plusieurs sous-exigences fonctionnelles. « Modifier le statut du conducteur » (F4) : (i) « disponible » et (ii) « non disponible », « Modifier le statut du passager » (F11) : (i) « libre », (ii) « en attente », (iii) « en transport public », (iv) « en course » et « Modifier le statut de la course » (F3) : (i) « en attente du conducteur », (ii) « en cours de covoiturage », (iii) « en course de transport commun », (iv) « terminée ». L'exigence « Entrer les informations du véhicule » (F29) peut être spécifiée en 4 sous-exigences fonctionnelles, dont (i) nombre de sièges, (ii) plaques, (iii) couleur et (iv) modèle. La dernière exigence fonctionnelle dans ce cas est « Valider les informations » (F21) : (i) le pays doit être mentionné si la destination n'est pas dans le même pays et (ii) la course ne peut pas être acceptée si une information est manquante.

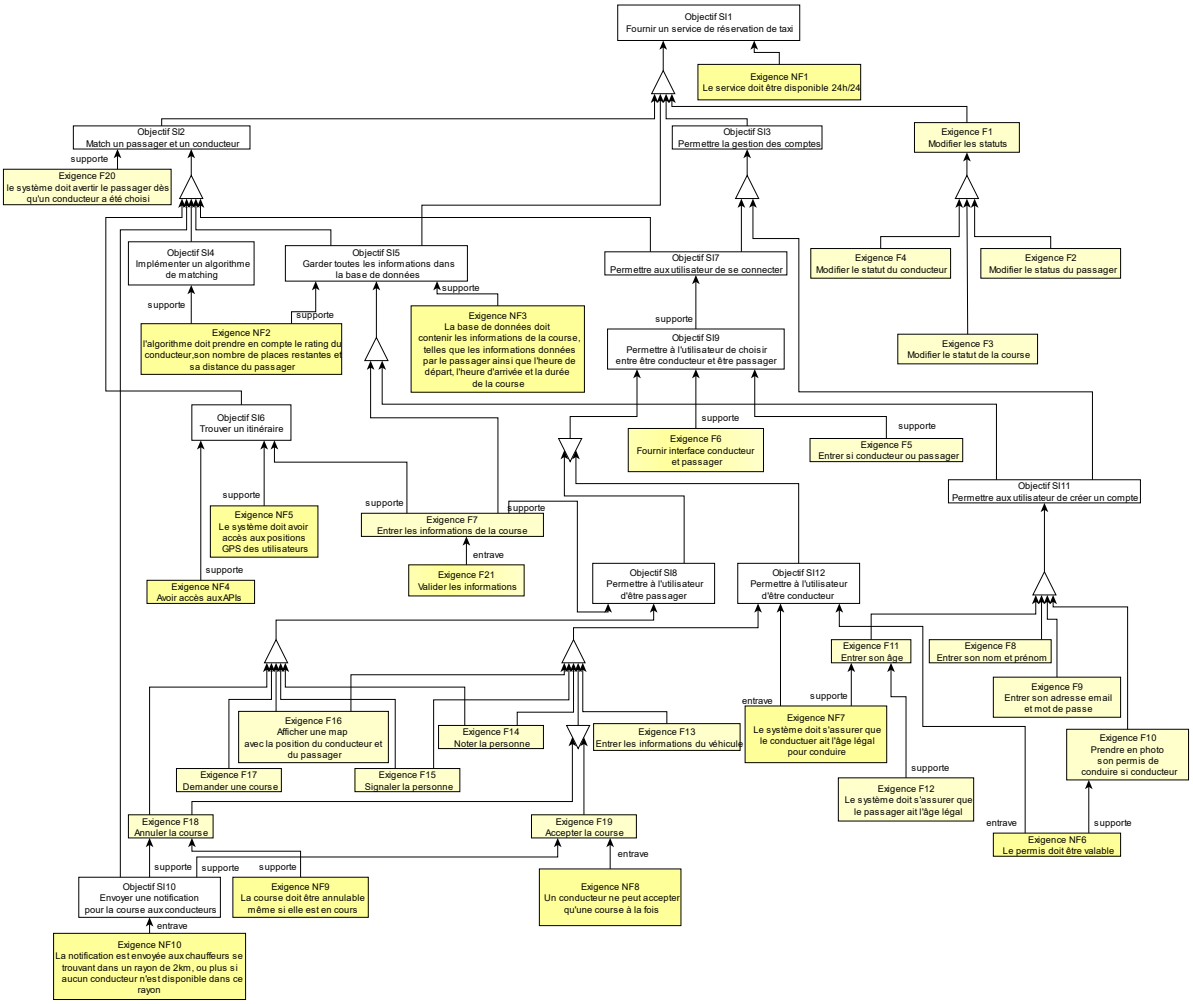

Figure 13 : Modèle des Exigences

Use Case (UML)(MAJ)

Après vous avoir présenté l'Analyse des objectifs, nous allons maintenant passer au Use Case.

Pour rappel, le Use Case permet de représenter l'usage que font les utilisateurs du système d'information [1].

Nous avons pour cela fait deux use cases : le premier concernant la réservation d'un taxi, et le deuxième décrivant le système de la course.

Concernant le premier use case, les acteurs sont encore une fois les conducteurs et passagers généralisés en utilisateurs. Ceux-ci peuvent déclencher plusieurs cas d'utilisation, comme se connecter ou s'enregistrer ainsi que gérer leur compte. Si l'utilisateur décide de se connecter et entre son mot de passe, le système regardera si le mot de passe est le bon comparé à celui renseigné lors de la création du compte. Après s'être connecté avec succès, l'utilisateur se voit attribuer la possibilité d'offrir ou demander une course, ceci en fonction du rôle qu'iel a choisi. S'iel choisit d'être conducteur, iel offre une course, iel doit dans ce cas donner les informations du véhicule. S'iel choisit d'être passager, iel demande donc une course et doit renseigner les informations de la course souhaitée. Ceci amène un conducteur et un passager à être match et d'ainsi pouvoir commencer la course. L'utilisateur à la possibilité de modifier et regarder ses informations.

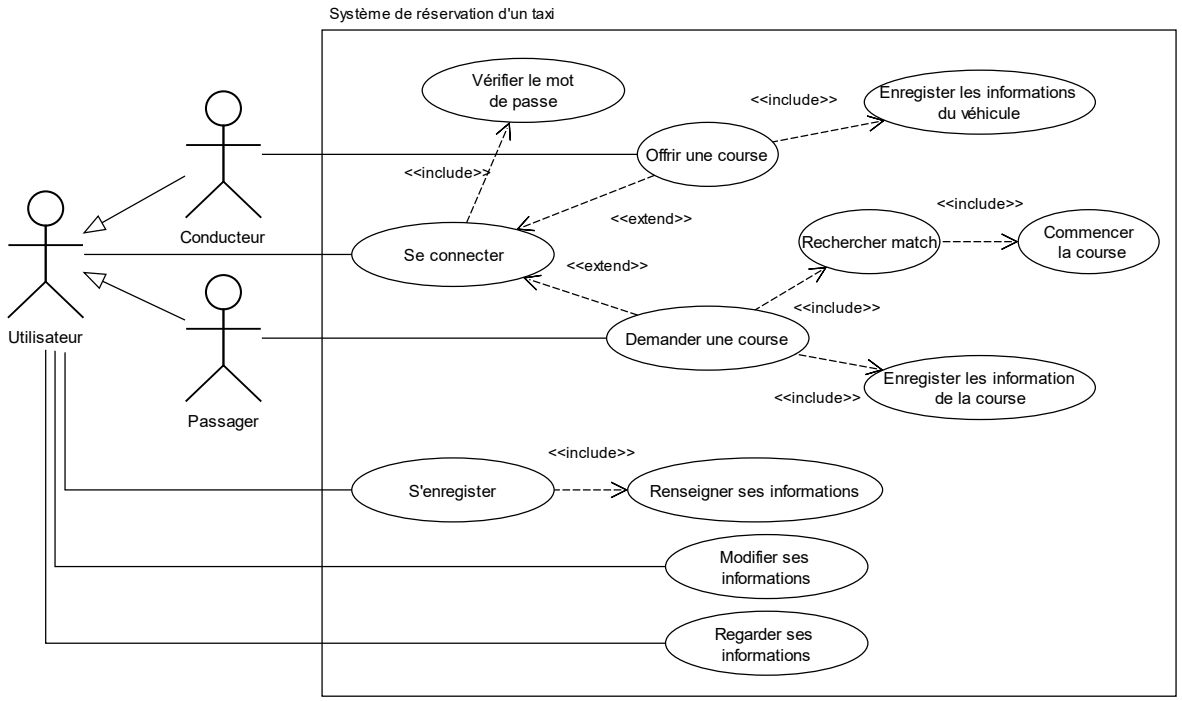

Figure 14 : Use Case du Système de réservation d'un taxi

En ce qui concerne le deuxième use case, celui-ci décrit les cas d'utilisation qui suivent le commencement de la course. Dès que la course commence, le conducteur et le passager peuvent presser « course commencée » pour avertir qu'ils ont bien commencé la course et peuvent signaler l'autre personne. La personne peut en effet signaler son partenaire de course avant même d'avoir presser « course commencée » dans le cas où l'autre partie ne se présenterait

tout simplement pas. Ils peuvent ensuite cliquer sur « Course terminée » pour indiquer que la course est finie et peut ensuite noter la personne si iel le désire.

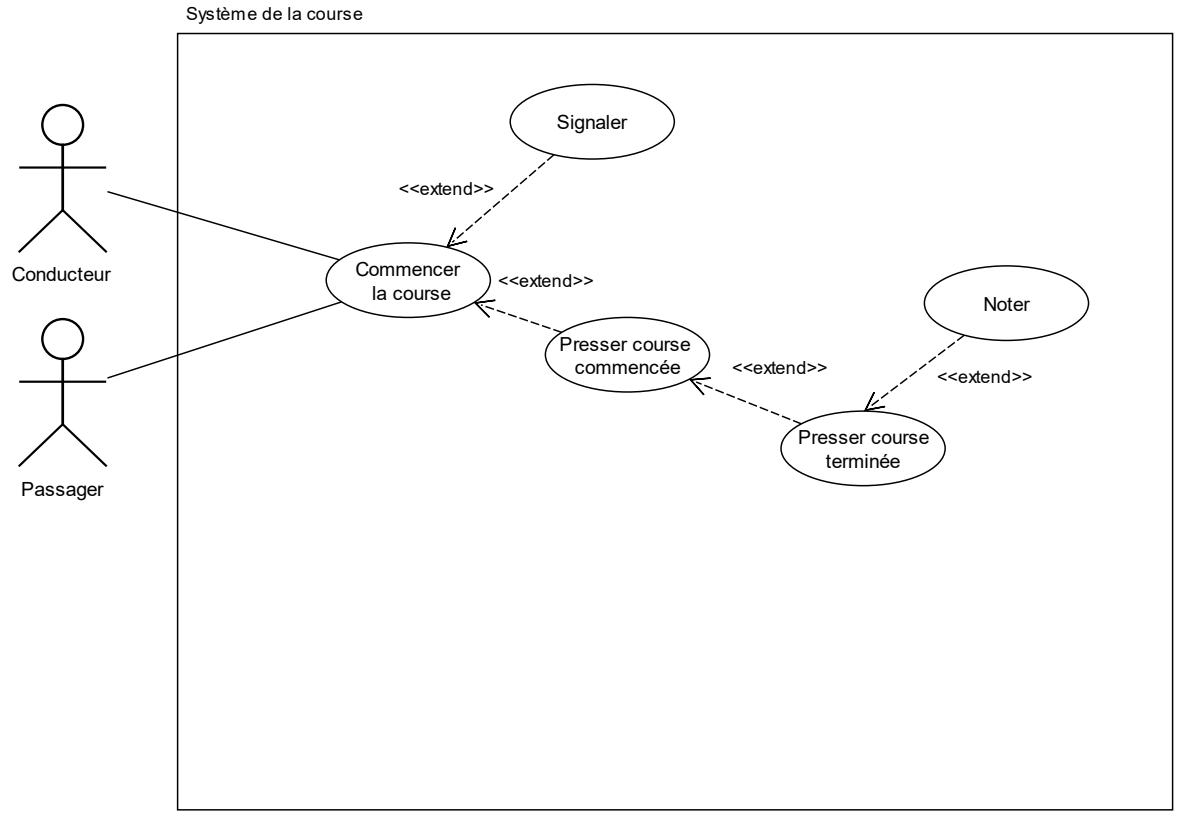

Figure 15 : Use Case du système de la course

Schéma conceptuel (MAJ)

Nous allons maintenant parler du schéma conceptuel. Celui-ci permet de visualiser les entités et les relations qu'elles ont entre elles. Ce schéma ne sert pas à représenter la base de données, mais à représenter les concepts.

Pour ce schéma conceptuel, 7 concepts peuvent être nommés : utilisateur, conducteur, passager, groupe, voiture, possession et course. A noter que « groupe » est l'ensemble des passagers pour une course, car le passager faisant la demande pour la course peut en effet voyager avec d'autres passagers. Mais ces passagers n'ont pas de compte et ne sont donc pas enregistré dans la base de données.

Etant donné que groupe et conducteur sont des spécialisations d'utilisateur, ceux-ci ne portent pas de lien d'association entre eux. Quartes associations font cependant surfaces : conducteur-possession, possession-voiture, conducteur-course, passager-groupe et groupe-course. En effet un conducteur possède une ou plusieurs voitures et une voiture peut être possédée par un ou plusieurs conducteurs (par exemple une voiture familiale), mais comme ceux-ci sont des associations multiples des deux cotées, un concept possession a été rajouté pour savoir qui possède quelle voiture. Ensuite, un conducteur peut offrir entre zéro (si ce dernier est en attente d'une course) et une course, et une course ne peut être offerte par qu'un seul conducteur. L'association entre groupe et course fonctionne de la même manière que la précédente, c'est-à-

dire qu'un groupe peut n'avoir demandé aucune course pour l'instant mais ne peut en demander qu'une maximum et une course ne peut être demandé que par un groupe. Une dernière association peut être relevée, celle de passager et groupe. En effet, un groupe peut être lié qu'à un passager et un passager peut être lié à maximum un groupe, car le groupe n'est créé que dès que ce dernier demande une course.

Nous pouvons remarquer un attribut dérivé, ce qui signifie que la valeur peut être calculée à partir d'autres attributs [1]. Ici, la durée de la course (duree_course') peut être calculée à partir de l'heure de début (heure_debut_course) et l'heure d'arrivée (heure_arrivee_course).

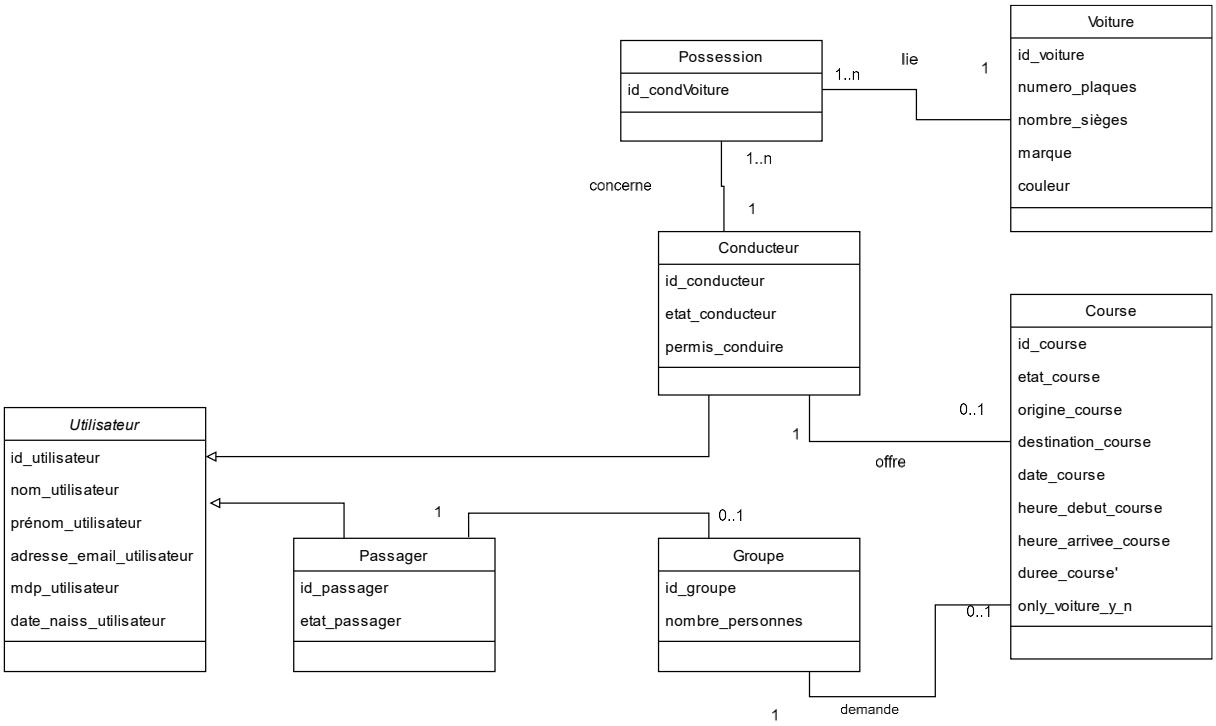

Figure 16 : Schéma Conceptuel

Schéma logique (MAJ)

Un schéma logique permet de représenter les clés primaires (Primary Keys) et étrangères (Foreign Keys). Ce schéma représente la base de données et comment ces différentes tables et attributs seront liés et représentés dans celle-ci.

Les clés primaires dépeignent les clés obligatoires de la table et les clés étrangères sont des clés qui se réfèrent à des clés primaires dans une autre table. Une clé peut être une clé primaire et étrangère en même temps.

Dans ce cas, les clés primaires sont les identifiants de chaque table (id_utilisateur, numero_plaques (celui-ci est un identifiant en lui-même car ces numéros sont uniques, clairs, pertinents et monovalués), id_condVoiture, id_course, id_groupe, id_conducteur et id_passager) ainsi que les clés étrangères envoyées faisant références aux clés primaires contenues dans d'autres tables (id_conducteur, id_passager, id_utilisateur et numero_plaques).

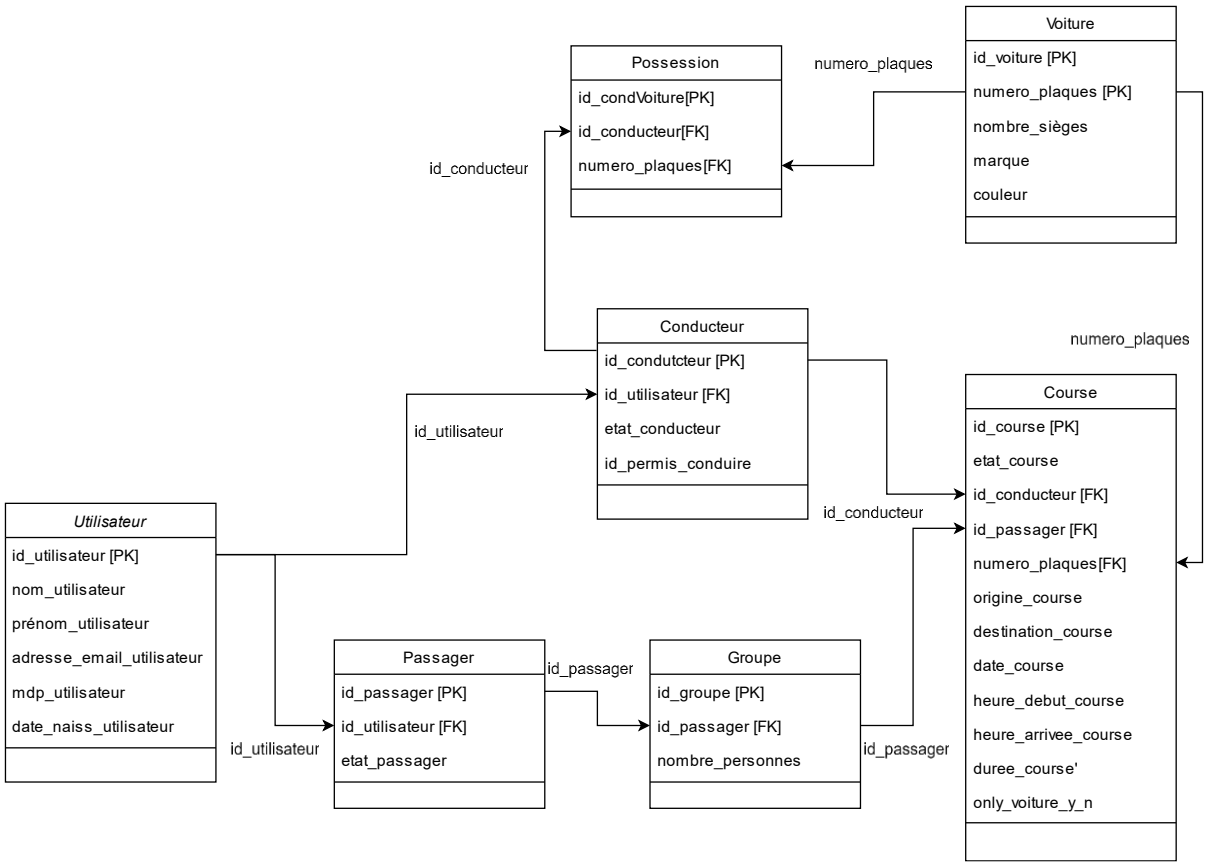

Figure 17 : Schéma Logique

Modèles du système

Après ces différents diagrammes, nous allons vous parler des différents modèles du système que nous avons envisagé.

Les diagrammes de séquence expliquent les liaisons entre les différents objets et comment les informations circulent.

Nous n'allons pas les expliquer en détails, les diagrammes étant lisibles.

Diagramme de séquence de connexion

Le premier, diagramme de séquence de connexion représente comment l'utilisateur se connecte au serveur. Celui-ci entre ses informations, qui vont ensuite être redirigées par le serveur dans la base de données pour regarder si ce dernier existe dans la base de données. S'iel y est présent, le serveur redirigera l'utilisateur vers une page qui lui permettra de choisir son rôle qui sera ensuite enregistré dans la base de données dans la table conducteur ou passager selon son choix. S'iel n'existe pas dans la base de données ou si le mot de passe est faux, le serveur informera l'utilisateur que son mot de passe ou son e-mail sont incorrectes.

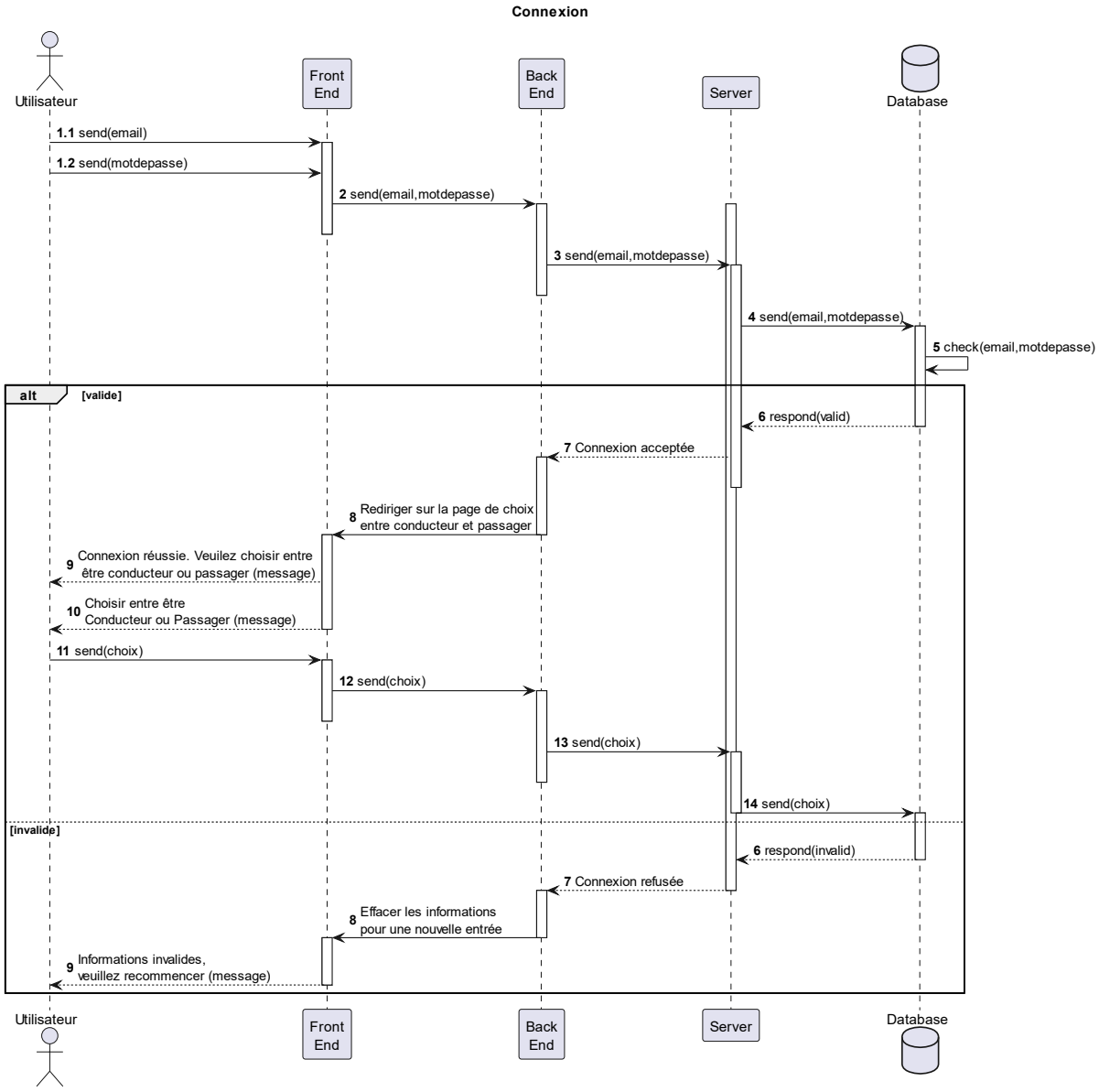

Figure 18 : Diagramme de séquence de connexion

Diagramme de séquence du conducteur

Le diagramme de séquence explique ce qu'il se passe si un utilisateur décide de choisir conducteur comme rôle.

Le conducteur doit dès lors renseigner les informations de son véhicule qui seront transmis à la base de données par le serveur et il sera redirigé sur une page web qui lui annoncera qu'il doit désormais attendre qu'une course s'affiche.

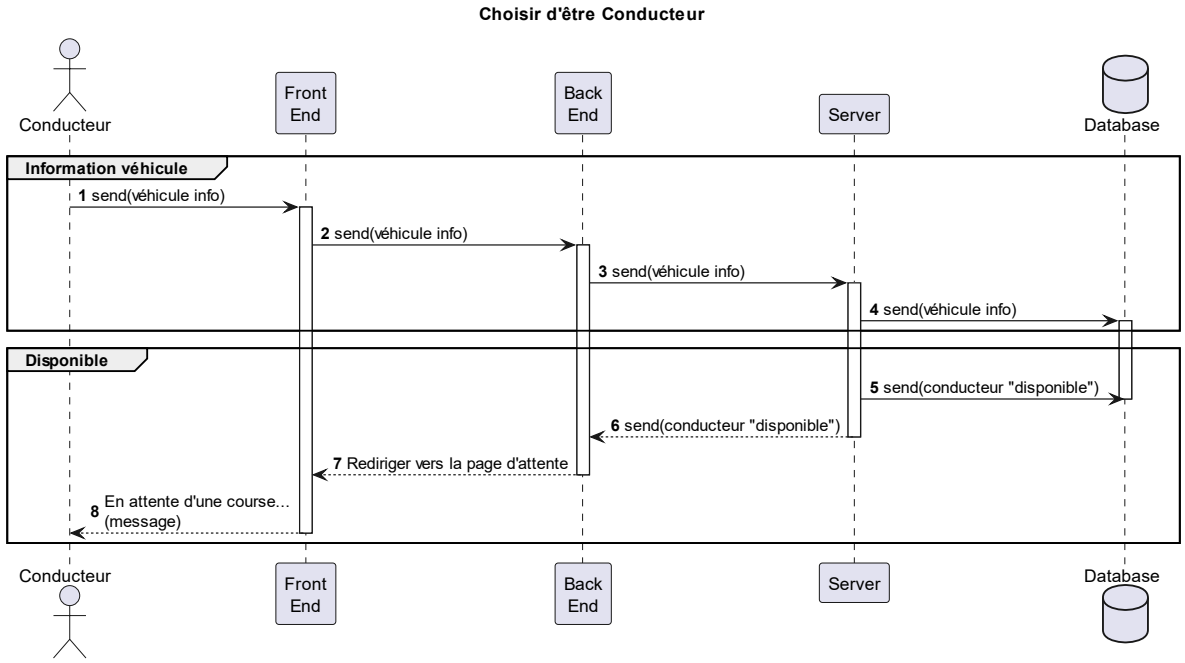

Figure 19 : Diagramme de séquence du conducteur

Diagramme de séquence du passager

Pour ce qui est du diagramme de séquence, celui-ci explique ce qu'il se passe si un utilisateur décide de choisir passager comme rôle.

Le passager doit envoyer les informations de la course qui seront également envoyé à la base de données par le serveur qui va ensuite le rediriger sur un page web qui l'informera que la course est en attente d'une réponse positive de la part d'un conducteur.

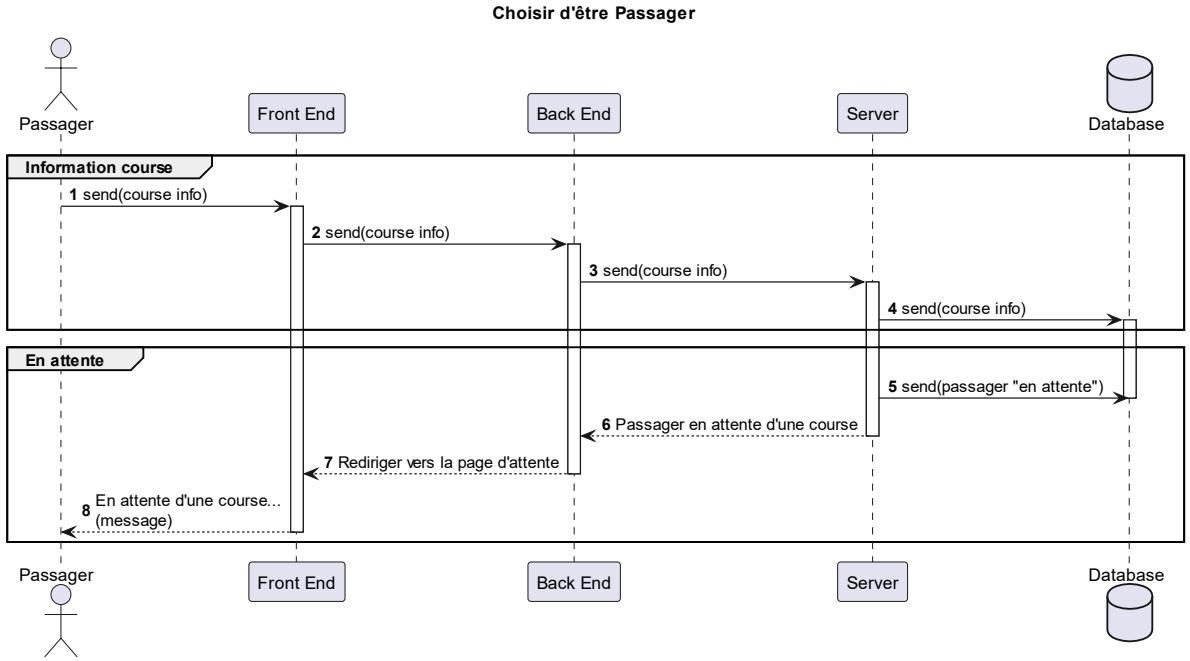

Figure 20 : Diagramme de séquence du passager

Diagramme de séquence de l’envoi de la course et de la course

Dès que le passager envoie la course au serveur qui l’envoie ensuite à la database, la connexion avec l’API d’itinéraire va être contacté et une connexion va être établit. Après lui avoir envoyé les informations de la course nécessaire, celui-ci retournera au serveur la map. Une fois cela fait, le serveur demandera à la base de données les informations de tous les conducteurs disponibles qui répondent aux critères de sélection et les mettra dans une liste pour ensuite les trier selon les critères de tri.

Une fois cela fait, le serveur enverra la demande de course au premier conducteur présent dans la liste. Si celui-ci accepte, le serveur enverra à la base de données le nom du conducteur ainsi que le fait qu’il ait accepté pour que la course soit créée dans celle-ci. Toutes les informations liées à la course seront ensuite envoyées au conducteur et au passager et seront tous deux redirigés vers une la page web contenant la course et la map.

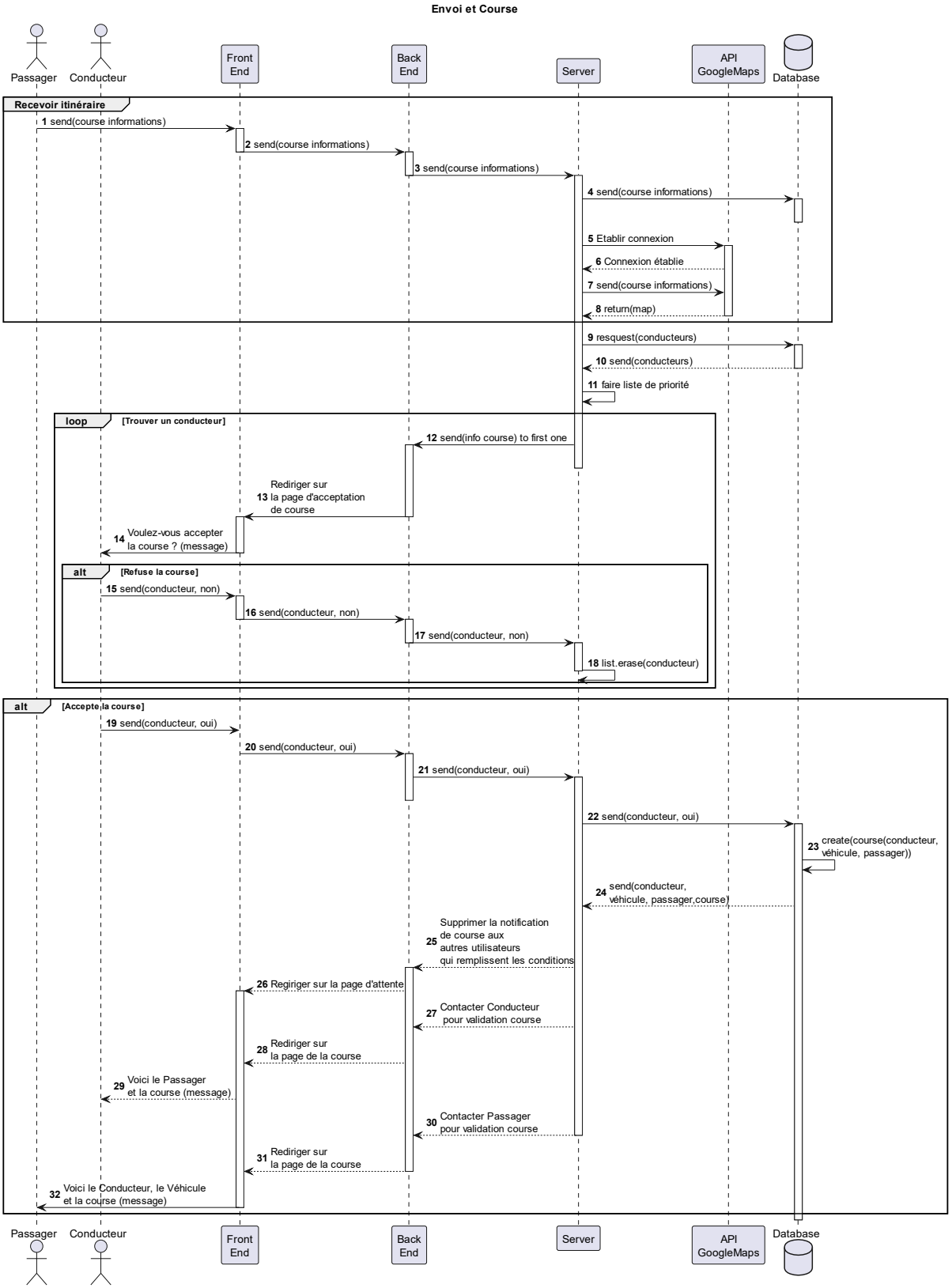

Figure 21 : Diagramme de séquence de l'envoi de la course

Modèles algorithmiques

Ce projet est composé d'un algorithme de sélection des conducteurs et d'envoi de la course. En effet, celui-ci doit créer une liste des conducteurs disponibles répondant aux conditions requise pour la course, c'est-à-dire qu'il doit être à moins de 5 kilomètres du passager, doit être disponible et avoir le nombre de place minimum requis pour pouvoir transporter le groupe (voir Figure 8). Tous ces conducteurs sont enregistrés dans une liste pour ensuite être trier par rapport à la distance qui les séparent du passager (plus ils sont prêts, mieux c'est), au nombre de sièges dont ils disposent (le plus proche ce nombre est du nombre de membres du groupe, le mieux c'est), ainsi que leur rating (le rating doit être le meilleur possible). Dans un troisième temps, le serveur contact la base de données pour voir si le 1er conducteur sur la liste ordonnée est toujours disponible. S'il n'est pas disponible, le conducteur est effacé de la liste et la même démarche est utilisée sur le nouveau conducteur n°1 (qui était précédemment le conducteur n°2). S'il est disponible, son statut est changé pour celui de « en attente » et le serveur lui envoie la course pour lui donner le choix d'accepter ou refuser la course. S'il refuse la course, le conducteur est effacé de la liste et ceci recommence avec le prochain conducteur. Au contraire s'il accepte la course, l'algorithme a rempli son rôle et ce conducteur et le passager sont mis ensemble. Si aucun conducteur n'a répondu aux exigences, le rayon de recherche s'agrandit de 2 kilomètres jusqu'à trouver quelqu'un dans un rayon de maximum 20 kilomètre. Si aucun conducteur n'a pu être trouvé, le passager est prévenu et la tâche se termine sans succès.

Voici ci-dessous une version simplifiée suivi d'une version plus complexe de l'algorithme de sélection ainsi que l'algorithme d'envoi de la course en version simple et complexe.

```
make table drivers
        list available_drivers (From database)

        while table empty and rayon less or equal to 20km
                for each driver in rayon + correct number or more seats asked from passenger
                        sort by distance
                        sort by seat
                        sort by rating
                if table empty, rayon +5km
        if empty, tell passenger no journey
        else go to send to the driver the journey request
```

Figure 22 : Simple pseudo-code de l'Algorithme de sélection et de tri

```
function sort_within_group(table, start_index, end_index):
    // Sorting drivers within the group by number of seats
    for i from start_index to end_index:
        for j from start_index to end_index - 1 - i:
            if table[j][2] > table[j+1][2]:  // Compare number of seats
                swap(table[j], table[j+1])

function calculate_distance(location1, location2)
    // Use appropriate method to calculate distance between two locations
    // For example, Haversine formula for geographic coordinates
    distance = distance_formula(location1, location2)
    return distance

function add_driver_to_table(table, driver, distance, num_seats, rating)
    // Add driver information to the table
    table.append([driver.name, distance, driver.num_seats, num_seats, rating])

function sort_within_group_by_seats(table, start_index, end_index):
    // Sorting drivers within the group by number of seats
    for i from start_index to end_index:
        for j from start_index to end_index - 1 - i:
            if table[j][2] > table[j+1][2]:  // Compare number of seats
                swap(table[j], table[j+1])

function sort_within_group_by_rating(table, start_index, end_index):
    // Sorting drivers within the group by rating
    for i from start_index to end_index:
        max_rating_index = i
        for j from i + 1 to end_index:
            if table[j][3] > table[max_rating_index][3]: // Compare rating
                max_rating_index = j
        swap(table[i], table[max_rating_index])

function make_table_drivers()
        Tri = empty_table()
        Rayon = 5
        available_drivers = liste obtenue depuis la base de données de tous les conducteurs libres et qui
        à le nombre de places minimal nécessaire ainsi que dans un rayon de 20km

        while Tri == empty and Rayon <= 20 km
                for each driver dans available_drivers
                        if driver in Rayon AND num_seats >= num_seat_journey
                                distance = calculate_distance(driver.location, passenger.location)
                                add_driver_to_table(table, driver, distance, num_seats, rating)

                                // Sorting by distance only the last one added to the table
                                i = 0
                                while i < length(table) - 1:
                                        min_distance_index = i
                                                for j from i + 1 to length(table) - 1:
                                                        if table[j][1] < table[min_distance_index][1]: // Compare distances
                                                                min_distance_index = j
                                                swap(table[i], table[min_distance_index])
                                                i++

                                // Sorting by number of seats within each group of drivers with the same distance(int!!!)
                                i = 0
                                while i < length(table):
                                        j = i
                                        while j < length(table) - 1 and table[j][1] == table[j+1][1]:
                                                j++
                                        // Sort drivers within the group by number of seats
                                        sort_within_group(table, i, j)
                                        i = j + 1

                                // Sorting by rating within each group of drivers with the same distance and same number of seats
                                i = 0
                                while i < length(table):
                                        j = i
                                        while j < length(table) - 1 and table[j][1] == table[j+1][1] and table[j][2] == table[j+1][2]:
                                                j++
                                        // Sort drivers within the group by rating
                                        sort_within_group_by_rating(table, i, j)
                                        i = j + 1

                if Tri == empty, rayon=+5

        if Tri == empty, no drivers for passenger (tell them this and to try again later)

        send_request_driver(Tri, passenger, journey)
```

Figure 23 : Complexe pseudo-code de l'Algorithme de sélection et de tri

```
send to the driver the journey request
        driver = 1er driver in table Tri
        get info journey and passager from database
        // check if driver is available
        if unavailable,
                delete driver from table Tri
                repeat send to the triver the journey request
        else
                send journey and passenger info
                wait for response
                when gets response from driver
                go to received from the driver their answer to the resquest


received from the driver their answer to the resquest
        if says yes
                send driver (journey ok)
                send passgenger (journey ok)


        else (no or ignore for 10 seconds)
                delete driver from table Tri
                repeat send to the driver the journey request
```

Figure 24 : Simple pseudo-code de l'Algorithme d'envoi

```
function send_request_driver(Tri, passenger, journey)
        driver = Tri[1]
        get info journey and passager from database
        if driver = unavailable
                delete.tri(driver)
                send_request_driver(Tri, passenger, journey)
        else
                send journey and passenger info
                // wait for response
                // when gets response from driver
                        receive_answer_request_driver (driver, passenger, journey)

function receive_answer_request_driver (driver, passenger, journey)
        if answer==yes,
                send driver (journey ok)
                send passgenger (journey ok)
        else
                delete.tri(driver)
                send_request_driver(Tri, passenger, journey)
```

Figure 25 : Complexe pseudo-code de l'Algorithme de sélection

Implémentation Client/Serveur

L'implémentation Client/Serveur a été faite à l'aide de Flask. Ce dernier permet d'accéder aux informations entrées par l'utilisateur pour ensuite les traiter et les utiliser dans le reste du code et/ou les envoyer à la base de données. En plus d'envoyer des données à la base de données, ce dernier permet également d'accéder à des données retenues dans la base de données pour les utiliser à son tour.

```python
from flask import Flask, render_template, request, redirect, url_for
from flask_mysqldb import MySQL
from datetime import datetime

app = Flask(__name__)

# Configure MySQL
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = ''
app.config['MYSQL_DB'] = 'test'
mysql = MySQL(app)

# Check MySQL connection
if mysql:
    print("Connection to MySQL database successful!")
else:
    print("Failed to connect to MySQL database.")

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    # Get the data from the user to log in
    if request.method == 'POST':
        username = request.form['email']
        password = request.form['password']
        # Check if username and password match a record in the database
        cur = mysql.connection.cursor()
        cur.execute("SELECT * FROM utilisateur WHERE adresse_email_utilisateur = %s AND mdp_utilisateur = %s",
                    (username, password))
        user = cur.fetchone()
        cur.close()
        if user:
            # Login successful, redirect to logged in
            return redirect(url_for('loggedin'))
        else:
            # Login failed, show error message or redirect to login page
            return "Login failed. Please check your username and password."
    return render_template('login.html')
```

Figure 26 : Flask partie 1

```python
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    # Get the data from the user to sign in
    if request.method == 'POST':
        new_email = request.form['email']
        new_password = request.form['password']
        new_firstname = request.form['firstname']
        new_lastname = request.form['lastname']
        # Get dob as string
        dob_str = request.form['dob']
        # Parse the input date string with the original format
        dob_date = datetime.strptime(dob_str, '%Y-%m-%d')
        # Parse the input date string with the original format
        # Format the date in the desired format
        dob_db = dob_date.strftime('%Y-%m-%d')
        # Insert data into MySQL database
        cur = mysql.connection.cursor()

        ### check if it doesnt already exist !!!
        cur.execute("INSERT INTO utilisateur (adresse_email_utilisateur, mdp_utilisateur, prenom_utilisateur, nom_utilisateur, date_naissance) VALUES (%s, %s, %s, %s, %s)",
                    (new_email, new_password, new_firstname, new_lastname, dob_db))
        mysql.connection.commit()
        cur.close()
        # Login successful, redirect to logged in
        return redirect(url_for('signedin'))
    return render_template('signup.html')

@app.route('/loggedin')
def loggedin():
    return render_template('loggedin.html')

@app.route('/signedin')
def signedin():
    return render_template('signedin.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 27 : Flask partie 2

Identifier des données de test

Mockaroo a été utilisé pour obtenir des données de test, car cela nous permettra de compiler les données qui nous intéressent. Les données sont équivalentes à celles décrites dans le schéma logique.

Usage des outils

Nous avons utilisé GitLab pour mettre les différents dossiers ainsi que le code à disposition à disposition. Celui-ci contient donc le code, les diagrammes de séquence et la différente documentation, telle que le cahier des charges, checkpoints et présentations PowerPoint.

Draw.io et yEd ont été utilisés pour faire les graphes d'analyse des objectifs ainsi que les schéma conceptuel et logique. yEd a permis de faire les modèles, tandis que Draw.io a été utilisé pour faire les schémas conceptuel et logique, les outils proposés par yEd étant moins adéquat pour remplir cette tâche que ceux mis à disposition par Draw.io.

Virtual Studio Code a été utilisé pour écrire le code. Git y étant intégré, cela permet de « commit » et « push » depuis cet IDE et de voir les modifications apportées.

Nous avons également utilisé l'extension PlantUML sur Virtual Studio Code pour dessiner le diagramme de séquence à l'aide de code pour rendre sa construction plus simple.

Mockaroo a, comme mentionné précédemment, été utilisé pour compiler des données de test. Les mots de passes ont été créé avec le type ISBN, car SQL n'a pas pu lire les mots de passe à cause de certains caractères spéciaux.

PhpMyAdmin nous a permi de stocker les bases de données dessus pour ensuite pouvoir les modifier à l'aide de code et de requêtes SQL.

Flask a été utilisé pour permettre la création de pages web localhost pour récupérer les données entrées par l'utilisateur, les traiter, et les envoyer à la base de données, et vice versa.

Conclusion

Dans ce rapport, nous avons introduit le projet en mentionnant ses problématiques, présenté les différents modèles d'analyse des objectifs, les schémas conceptuel et logique, expliquer les modèles du système et algorithmiques identifier des données de test, ainsi qu'énuméré les outils utilisés tout au long de ce dernier.

Les modèles nous donnent une base robuste pour mener ce projet à bien, l'implémentation nécessitant une connaissance des objectifs et de ce qui doit être réalisé pour ne pas implémenter quelque chose d'erroné. Les algorithmes nous permettent quand à eux de voir à quoi pourrait ressembler le code et ainsi nous permettre de créer la logique derrière ceux-là.

Nous suivrons ce rapport par un rapport final, qui prendra en compte les retours de nos clients quant aux améliorations à effectuer, la suite logique de celui-ci, ainsi que le système fonctionnel.

Références

[1] Prof. Jolita Ralyté, cours « Analyse des objectifs »

GitLab depository : https://gitlab.unige.ch/courses1/pt1/2324/g17