Ride Quest

Final

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 dernier (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 à utiliser plus régulièrement les transports en commun. Les passagers n'auraient pas à payer pour ce service mis à 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 donc eu 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 de course. Le passager doit pouvoir proposer une course et le conducteur doit être choisi parmi d'autres conducteurs remplissant les conditions pour mener à bien la course, doit pouvoir accepter la course et pouvoir recevoir les informations nécessaires pour conduire le passager à une destination définie. Ce projet étant une version simplifiée de mon projet final, le fait de pouvoir faire une partie de la course en transport en commun n'a pas encore été implémenté, donc une course ne peut avoir comme trajet qu'un itinéraire réalisable en voiture.

En ce qui concerne ce projet, il va falloir designer et implémenter un système de réservation de covoiturage. Pour cela, il sera nécessaire d'utiliser un API qui permettra la création d'un itinéraire ainsi que l'implémentation d'un algorithme de triage selon des critères de sélection. Cet algorithme de tri aura comme utilisation la sélection du conducteur correspondant le mieux à la course ainsi que de l'envoi de la course à ce conducteur et de l'attente d'une réponse. Il faudra également récolter les données GPS des utilisateurs pour calculer les distances entre les conducteurs et le point d'origine de la course pour ainsi permettre un tri efficace.

Nous allons dans ce rapport analyser les objectifs pour préciser le projet et ses contraintes, proposer une conception, implémenter cette dernière, décrire les outils utilisés, évaluer ce projet avec notre sens critique, pour ainsi terminer par une conclusion.

Analyse des objectifs

Nous allons maintenant analyser les objectifs de ce projet. Pour cela, nous allons présenter le modèle des objectifs, la liste des acteurs, les modèles des activités, les modèles des règles de gestion, pour finir par le modèle des exigences fonctionnelles et non-fonctionnelles.

Modélisation 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 important. 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), « Obtenir 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 tel que Google Maps (Opportunité 1) permettront de trouver l'itinéraire le plus cohérent étant données les 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 l'âge légal pour utiliser un tel service (Obligation 3). Ces obligations n'ont pas été prises en compte lors de l'implémentation, mais elles ne restent pas moins importantes à mentionner.

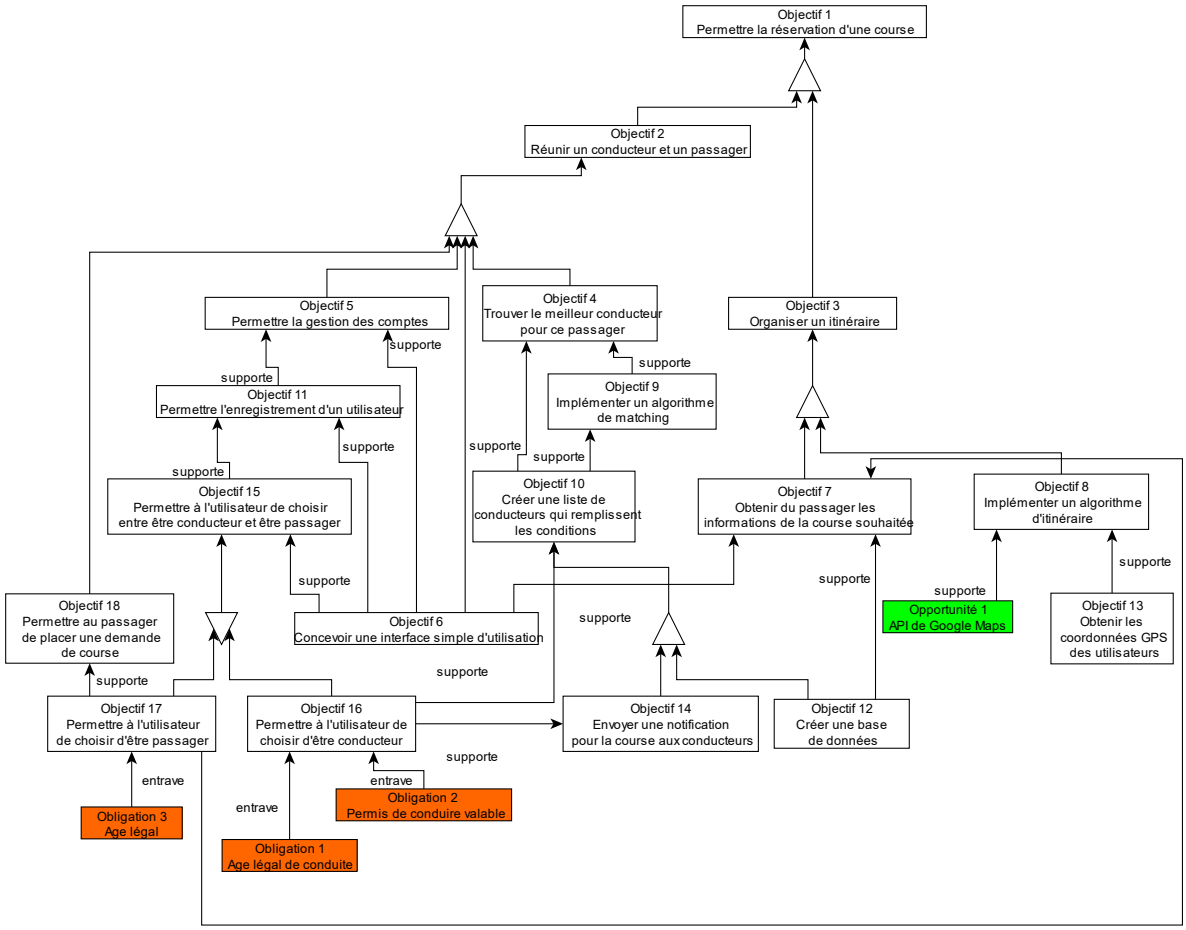

Figure 1 : Modèle des Objectifs

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.

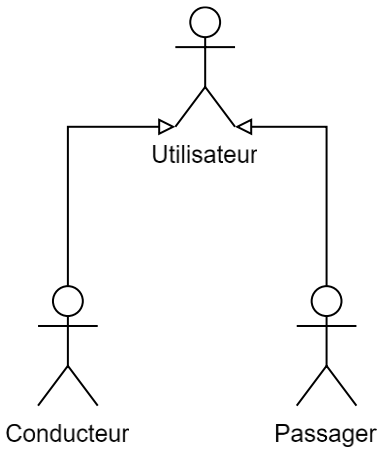

Figure 2 : Liste des acteurs

Modélisation des activités

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

Par soucis de lisibilité, nous avons distingué 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 en elle-même. Chacun modélise une partie et/ou des acteurs différents allant de la mise à disposition d'une course par le passager 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), choisir entre entrer les informations du véhicule qu'il utilisera (Processus 3.5) (Info 7-9) pour ensuite choisir un véhicule (Processus 4) (Info 10) ou directement choisir un véhicule existant (Processus 4) (Info 10). Une fois cela fait, le conducteur attendra une course (Processus 5) et sera ainsi définit comme « libre » (Info 11). Ce dernier peut se déconnecter (Processus 7) et se verra donc retirer son rôle de conducteur (Info 18).

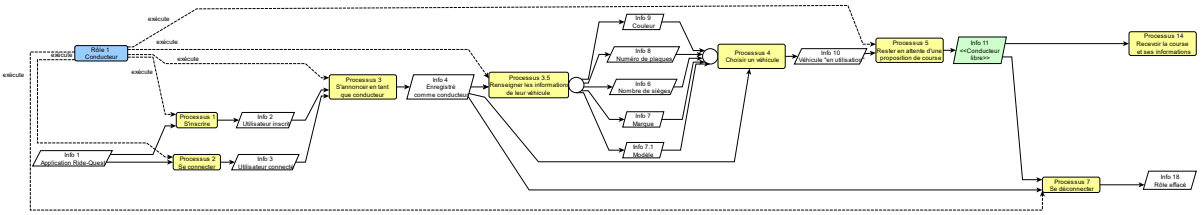

Figure 3 : Modèle des Activités du Conducteur

Processus du passager

Pour ce qui est du passager, celui-là suit les mêmes processus pour s'annoncer en tant 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, l'itinéraire va être créé (Processus 9) à l'aide de l'API de Google Maps (Ressource Ext 1). Le passager, comme le conducteur, peut se déconnecter en tout temps (Processus 7), ce qui lui effacera son rôle de passager (Info 16) et modifiera l'état de la course comme étant « abandonnée » si celle-ci a entre-temps été créée (Info 19).

e

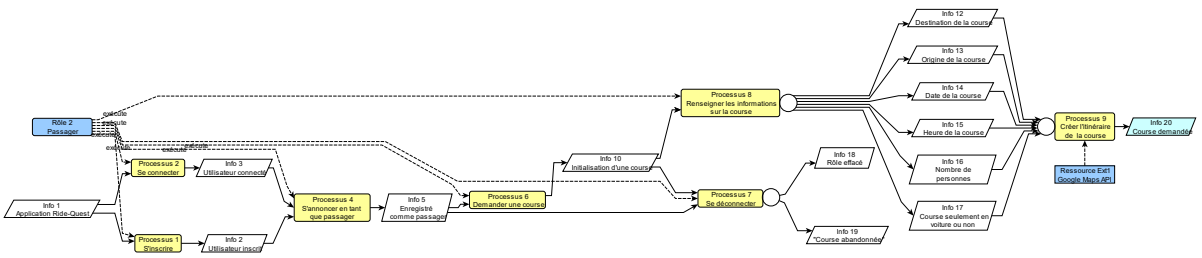

Figure 4 : Modèle des Activités du Passager

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 crée donc une liste des conducteurs (Processus 10), analyse les informations des conducteurs obtenues via la base de données (Processus 28) en se focalisant sur le nombre de sièges (Info 6), la distance entre le conducteur et le point d'origine de la course (Info 42) et de 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 jugés (Processus 30) étant donné 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).

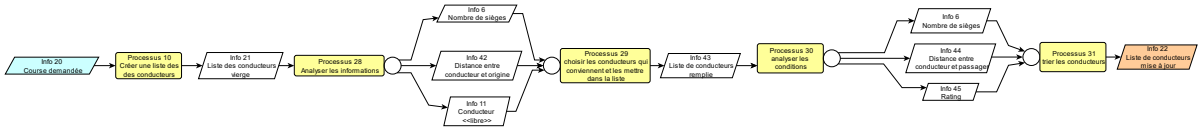

Figure 5 : Modèle des Activités de l'algorithme de sélection

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), nous nous penchons sur le prochain conducteur (Processus 21) et cela recommence 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 l'a 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 est choisi (Info 27) et le prochain processus prend le relais. Au contraire, s'iel l'ignore ou n'accepte pas la course, le statut de la course changera pour « libre », le conducteur est redirigé vers la page d'attente (Processus 5) et la course sera envoyée au prochain conducteur sur la liste en recommençant les mêmes étapes (Processus 21-...). Si l'algorithme arrive à la fin de la liste et que personne n'a encore été trouvé, le conducteur est invité à se déconnecter et à réessayer plus tard (Processus 20). Les deux partis peuvent en tout temps se déconnecter pour arrêter le processus (Processus 7).

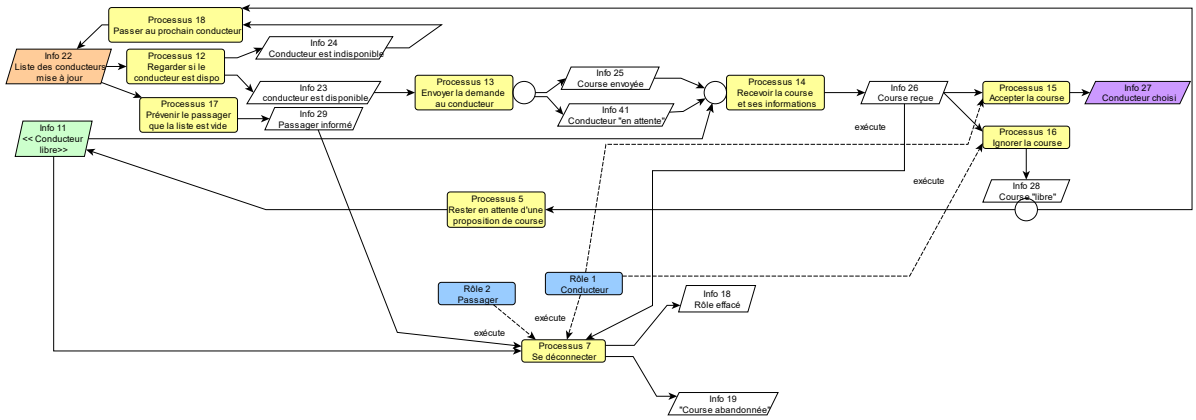

Figure 6 : Modèle des Activités de l'envoi de la course

Processus de la course

Dès que le conducteur a été choisi (Info 27), le passager et le conducteur sont prévenus du matching (Processus 17-18). La course peut ainsi commencer (Processus 22). Les statuts du passager, conducteur et de la course sont changés (Info 33-35). Une fois la course terminée (Processus 25), l'utilisateur peut noter sa paire (Processus 27) et la note sera ensuite calculée par le système en prenant en compte sa moyenne (Processus 28). L'utilisateur peut également signaler l'autre personne (Processus 26) durant la course.

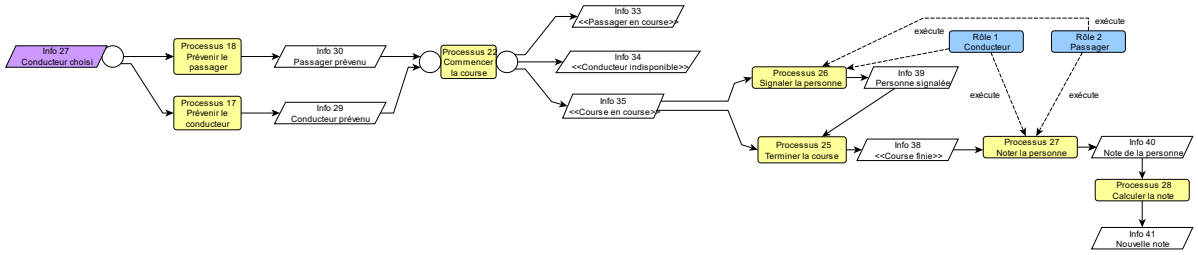

Figure 7 : Modèle des Activités de la course

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 ajoutée, 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 qui peut être mentionnée est celle qui ajoute à l'Objectif 9 le fait que le matching doit se faire en prenant compte de conditions définies (Règle 5).

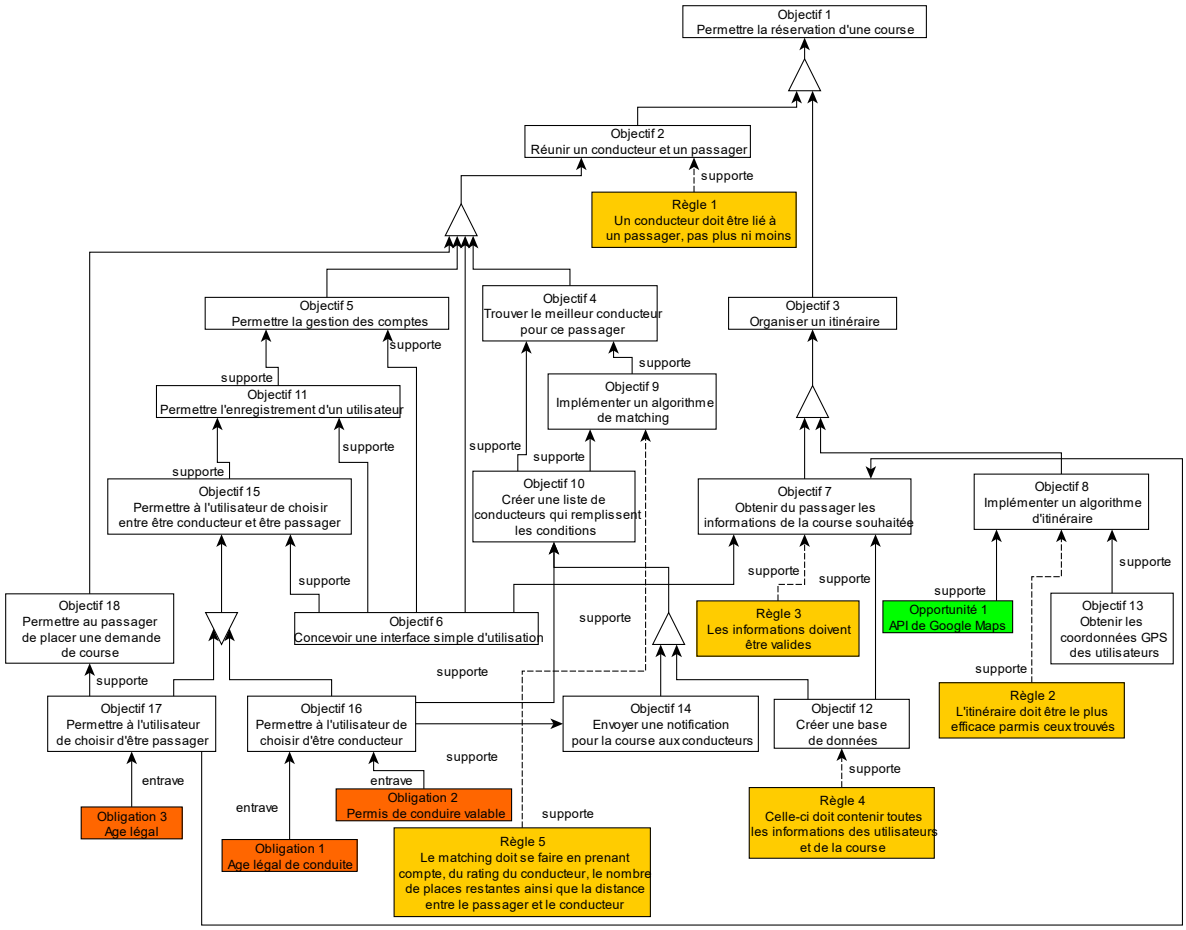

Figure 8 : Modèle des Règles de gestion des objectifs

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), à la course (Règle 6-8,19) et au système (Règle 9,20-22,24-27).

Processus du conducteur

Ce processus ne contient que 5 règles. Trois de celles-ci sont des contraintes posées à l'utilisateur (Règle 1-3), 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).

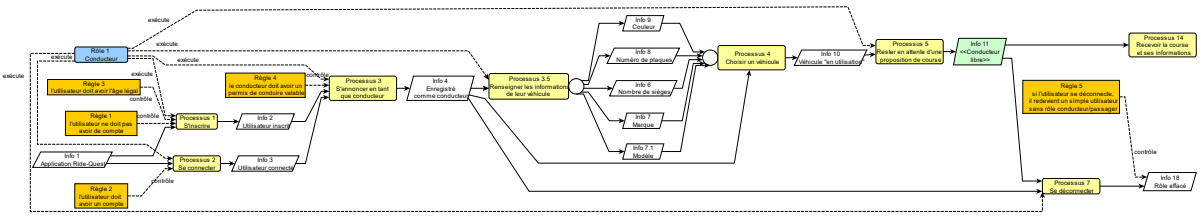

Figure 9 : Modèle des Règles de gestion des Activités du conducteur

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. Trois règles s'y ajoutent : le fait que le statut de la course soit marqué comme « abandonnée » 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).

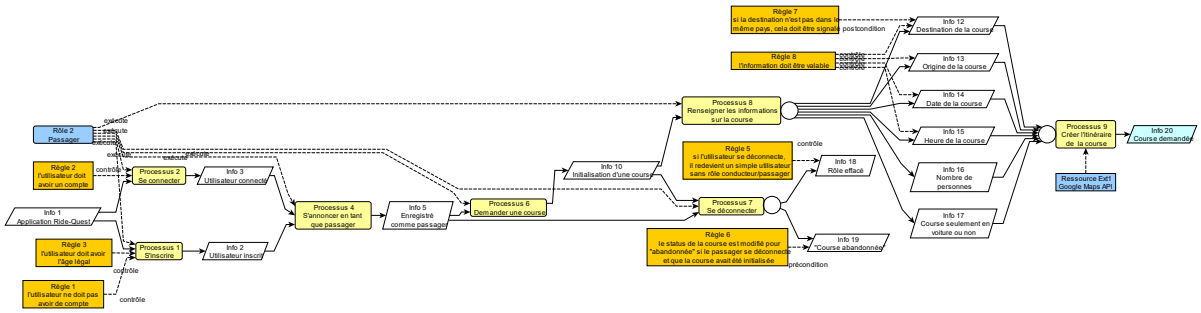

Figure 10 : Modèle des Règles de gestion des Activités du passager

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 et 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.

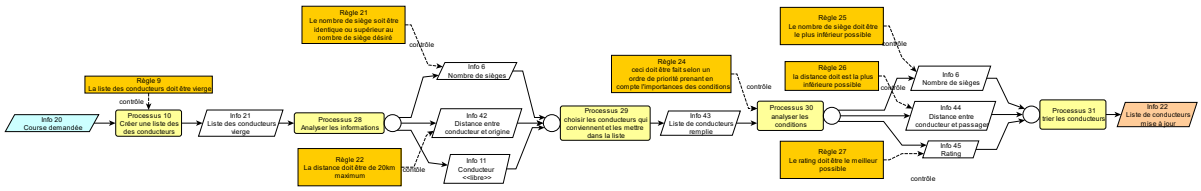

Figure 11 : Modèle des Règles de gestion des activités de l'algorithme de sélection

Processus de l'envoi de la course

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

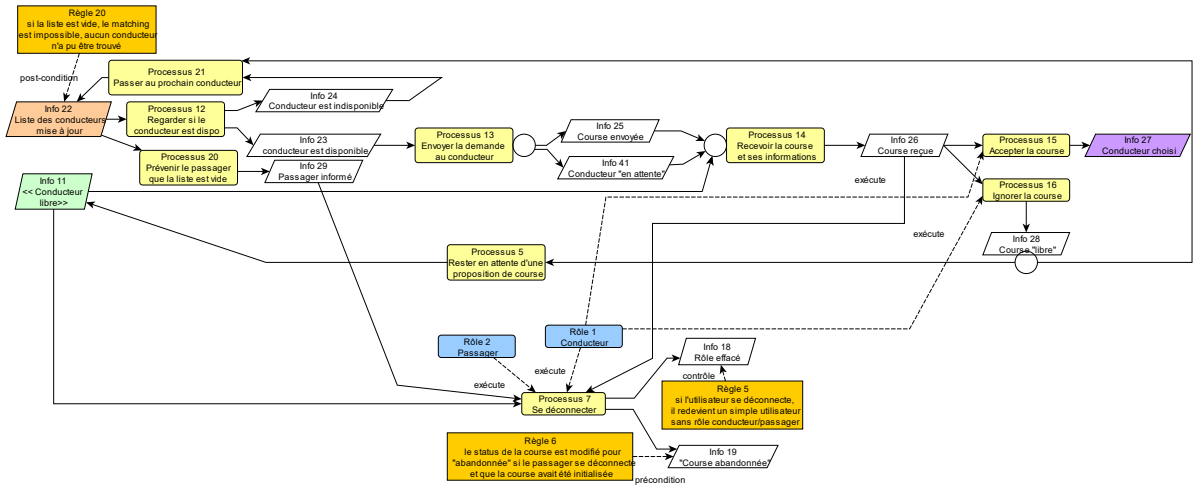

Figure 12 : Modèle des Règles de gestion des Activités de l'envoi de la course

Processus de la course

Les règles du processus de la course se comptent au nombre de trois également. 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).

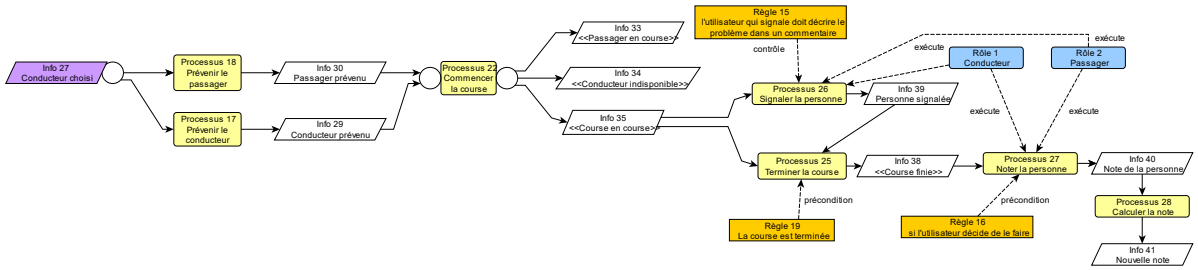

Figure 13 : Modèle des Règles de gestion des Activités de la course

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 sous-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) la destination, (iii) la date, (iv) l'heure de départ ou l'heure d'arrivée et (v) le nombre de personnes. Toutes les exigences fonctionnelles relatant des statuts se voient toutes être composées de plusieurs sous-exigences fonctionnelles. Les exigences fonctionnelles « Modifier le statut du conducteur » (F4) et « Modifier le statut du passager » (F11) sont composées de (i) « libre », (ii) « en attente » et (iii) « en course », et l'exigence fonctionnelle « Modifier le statut de la course » (F3) qui pour sa part est constituée de (i) « libre », (ii) « en attente », (iii) « en course», (iv) « terminée » et (v) « abandonnée ». L'exigence fonctionnelle « Entrer les informations du véhicule » (F29) peut être spécifiée en 5 sous-exigences fonctionnelles, dont (i) entrer le nombre de sièges, (ii) le numéro d'immatriculation, (iii) la couleur, (iv) le modèle et (v) la marque.

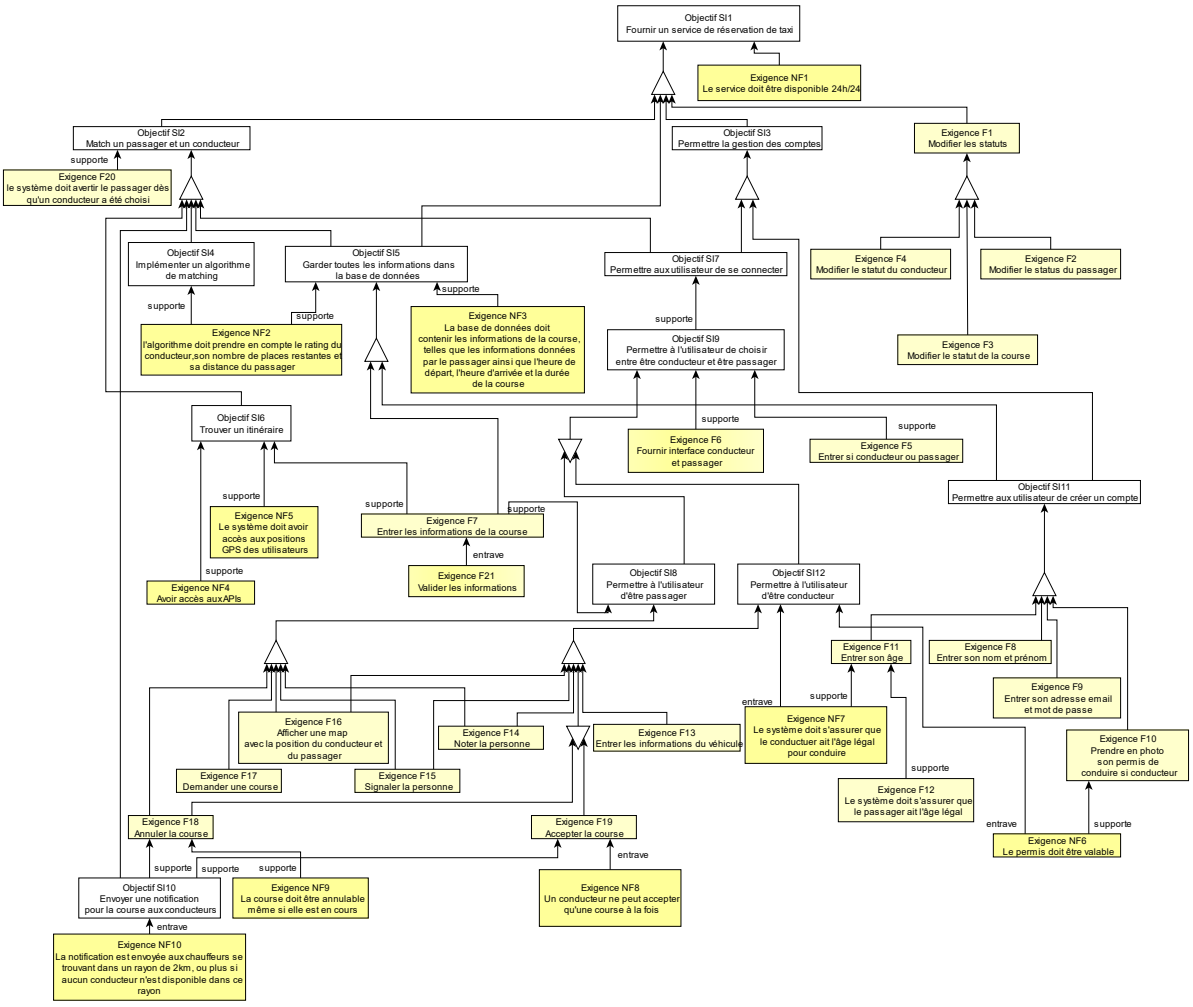

Figure 14 : Modèle des Exigences

Tous ces modèles pensés et complétés au fur et à mesure nous permettent donc de passer à la conception de ce système.

Conception

Après vous avoir présenté l'Analyse des objectifs, nous allons maintenant passer à la conception de ce projet. Nous allons décomposer ce chapitre en plusieurs parties : le Use Case, le schéma conceptuel et le schéma logique.

Use Case (UML)

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. Si l'utilisateur décide de se connecter et entre son adresse e-mail et 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 et doit dans ce cas sélectionner un véhicule, ou dans le sschoisit 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 mis ensemble et d'ainsi pouvoir commencer la course.

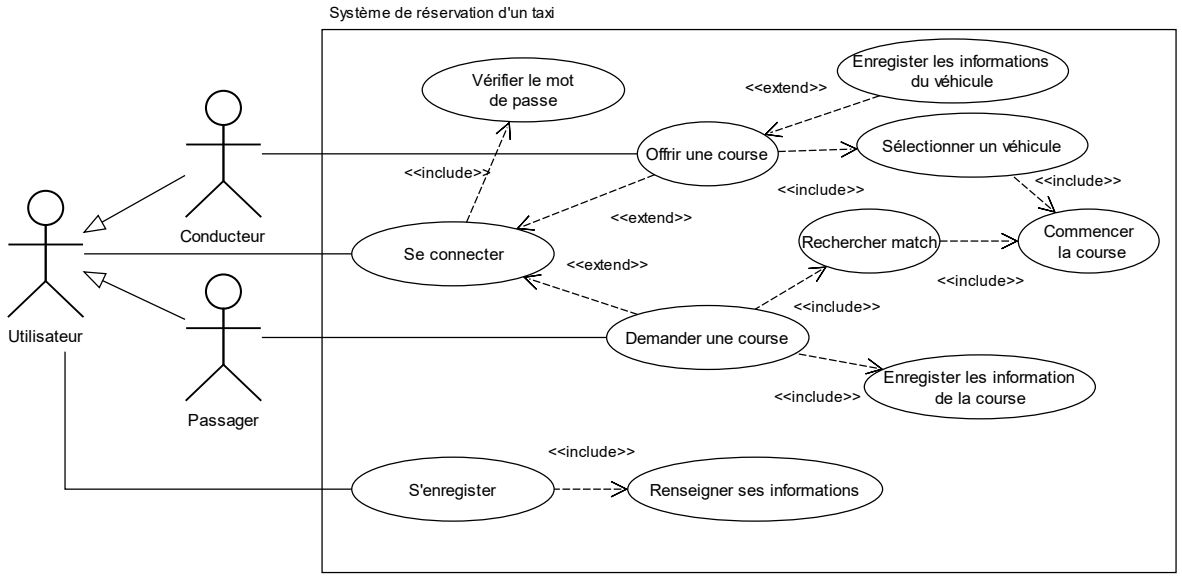

Figure 15 : Use Case du Système de réservation d'un taxi

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 signaler l'autre personne. La personne peut par exemple signaler son partenaire de course dans le cas où celui-ci 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 s'iel le désire.

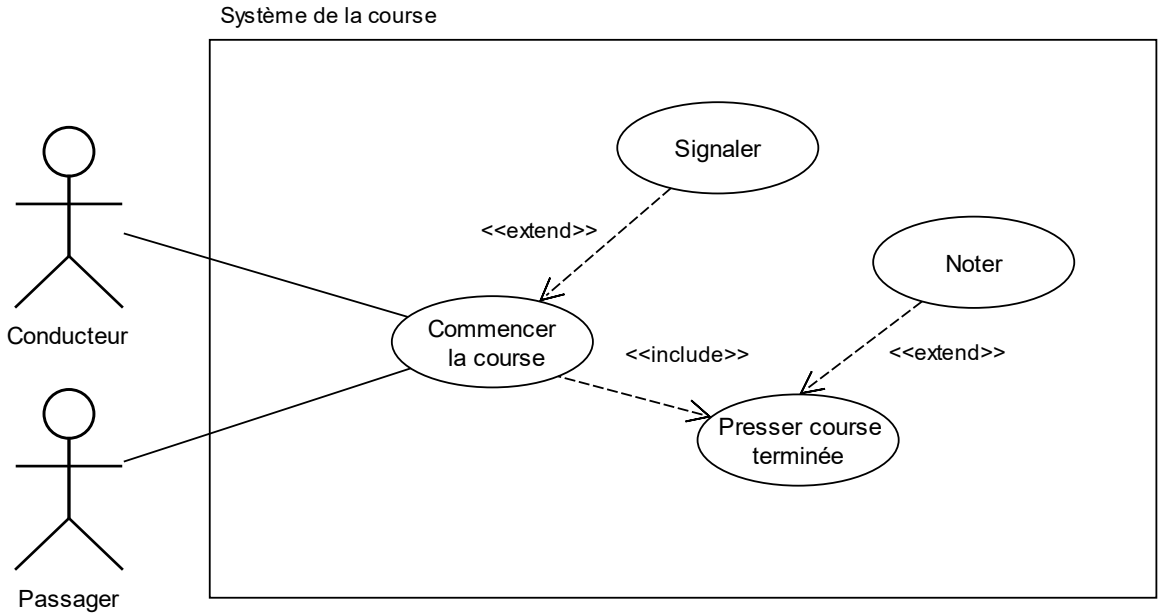

Figure 16 : Use Case du système de la course

Schéma conceptuel

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, 6 concepts peuvent être nommés : utilisateur, conducteur, passager, voiture, possession et course.

Etant donné que passager 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-course. En effet un conducteur peut posséder 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 garder une trace de qui possède quelle voiture. Ensuite, un conducteur peut accepter entre zéro (si ce dernier est en attente d'une course) et une course, et une course ne peut être acceptée que par un seul conducteur. L'association entre passager et course fonctionne de la même manière que la précédente, c'est-à-dire qu'un passager peut n'avoir demandé aucune course pour l'instant mais ne peut en demander qu'une au plus et une course ne peut être demandé que par un passager.

Pour ce qui est des attributs, ceux-ci représentent les données liés aux concepts. Chaque concept en est formé de plusieurs et contient un minimum d'un attribut, qui est l'identification de celui-ci. Ce dernier peut être aussi simple qu'un numéro incrémenté de un pour chaque nouvelle entrée, ou dans le cas du concept « Voiture », le numéro d'immatriculation du véhicule, qui est unique. Ensuite, tous les concepts contiennent des informations telles que les informations de l'utilisateur, de la course, etc. Ces informations nous seront utiles pour la mise à bien de notre système.

Nous pouvons remarquer deux attributs dérivés, 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 (debut_course) et l'heure d'arrivée (arrivee_course), et la distance (distance_course') peut être calculée grâce au point d'origine (origine_course), de destination (destination_course) et à l'API de Google Maps (sur lequel nous reviendrons plus tard).

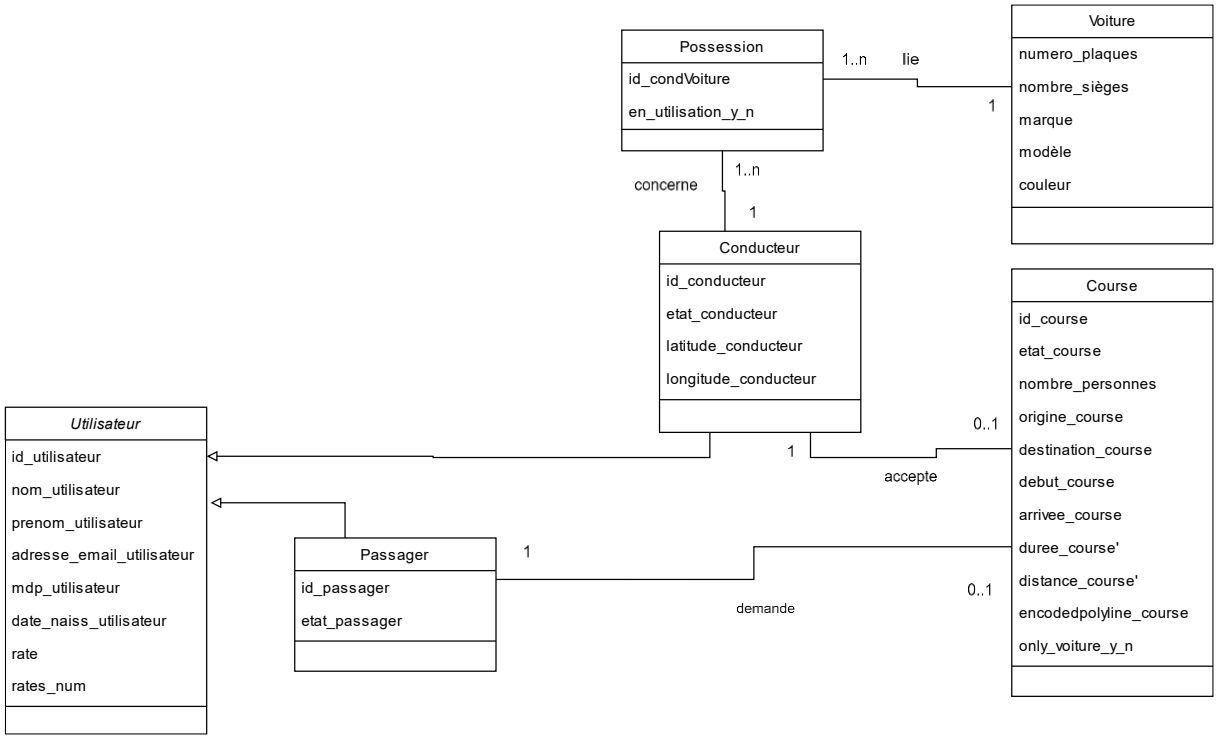

Figure 17 : Schéma Conceptuel

Une fois ce schéma conceptuel créé, il est possible de modéliser le schéma logique en se servant de ce premier comme point de départ.

Schéma logique

Le schéma logique est une représentation plus fidèle de la base de données que le schéma conceptuel. Il permet de représenter les clés primaires (Primary Keys) et étrangères (Foreign Keys). Ce schéma représente 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 à la fois une clé primaire et étrangère.

Dans ce cas, les clés primaires sont les identifiants de chaque table (id_utilisateur, numero_plaques (celui-ci, comme mentionné précédemment, est un identifiant en lui-même car ces numéros sont uniques, clairs, pertinents et monovalués), id_condVoiture, id_course, 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).

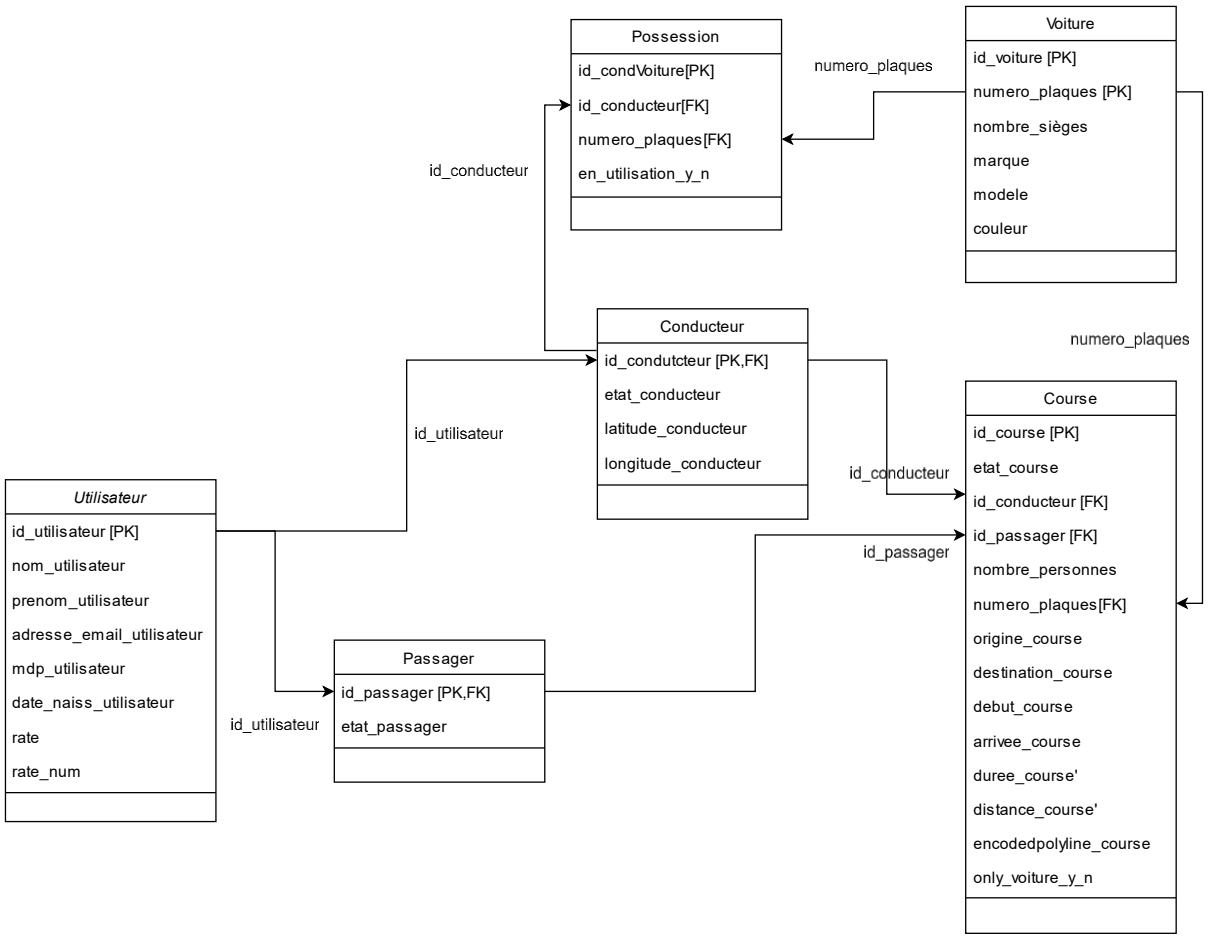

Figure 18 : Schéma Logique

Modèles du système

Après ces différents diagrammes, nous allons vous parler des différents modèles du système que nous avons envisagés. A noter que ceux-ci ne sont pas exhaustifs, ils ne servent qu'à représenter comment le système fonctionne.

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

Ils comptent au total six lignes de vie : trois acteurs (utilisateur, passager et conducteur), le front end et le backend, ce dernier composé du serveur et de la base de données.

Nous n'allons pas les expliquer en détails, les diagrammes étant lisibles et compréhensibles en par eux-mêmes.

Diagramme de séquence de connexion et de choix du rôle

Le premier diagramme de séquence représente comment l'utilisateur se connecte au serveur et choisit son rôle, donc « conducteur » ou « passager ».

Celui-ci entre ses informations, qui vont ensuite être récoltées par le serveur. Ce dernier envoie ensuite les informations de connexion à la base de données pour analyser si ce dernier existe dans cette dernière. Si les informations sont présentes, la base de données retourne l'identifiant (id_utilisateur) lié à ces informations et le serveur redirige l'utilisateur vers une page qui lui permet de choisir son rôle, rôle qui sera ensuite enregistré dans la base de données dans la table

attribuée au rôle conducteur ou passager selon son choix. Au contraire, si les informations n'y sont pas présentes, le serveur redirige l'utilisateur sur une nouvelle page de connexion en l'informant que son mot de passe ou son adresse e-mail sont incorrects et lui demande de recommencer.

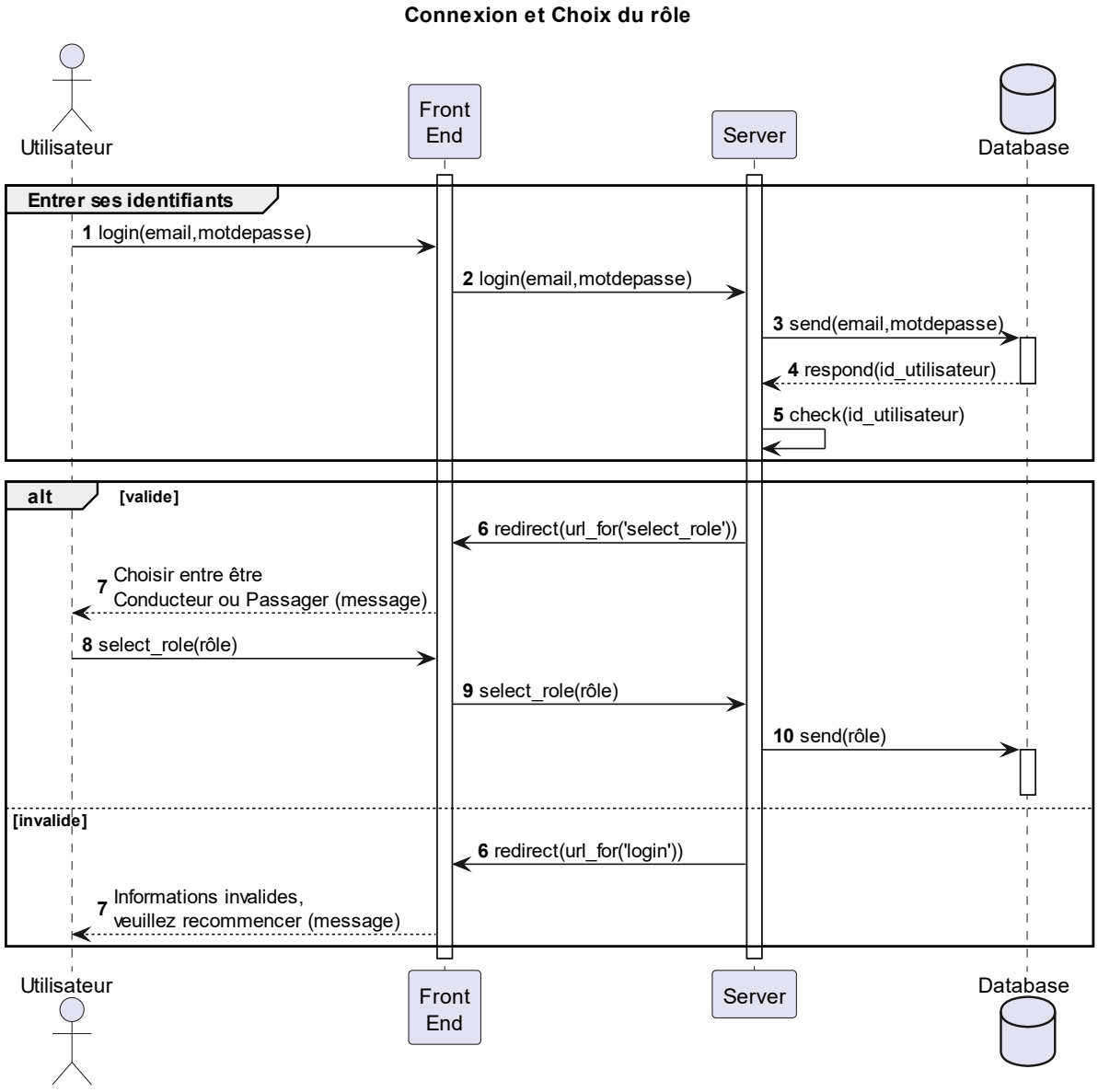

Figure 19 : Diagramme de séquence de connexion

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 choisir le véhicule qui sera utilisé lors de la course. Si ce dernier veut ajouter un nouveau véhicule à sa liste, iel doit renseigner les informations de son véhicule qui seront transmises à la base de données par le serveur. Cela fait, iel est redirigé sur la page initiale et doit choisir son véhicule. Une fois le véhicule sélectionné, son statut se transforme en « libre » et iel est redirigé sur une page web qui lui demande d'accepter de renseigner sa position GPS.

Une fois cela accepté, une page web lui annoncera qu'il doit désormais attendre qu'une course lui soit proposée.

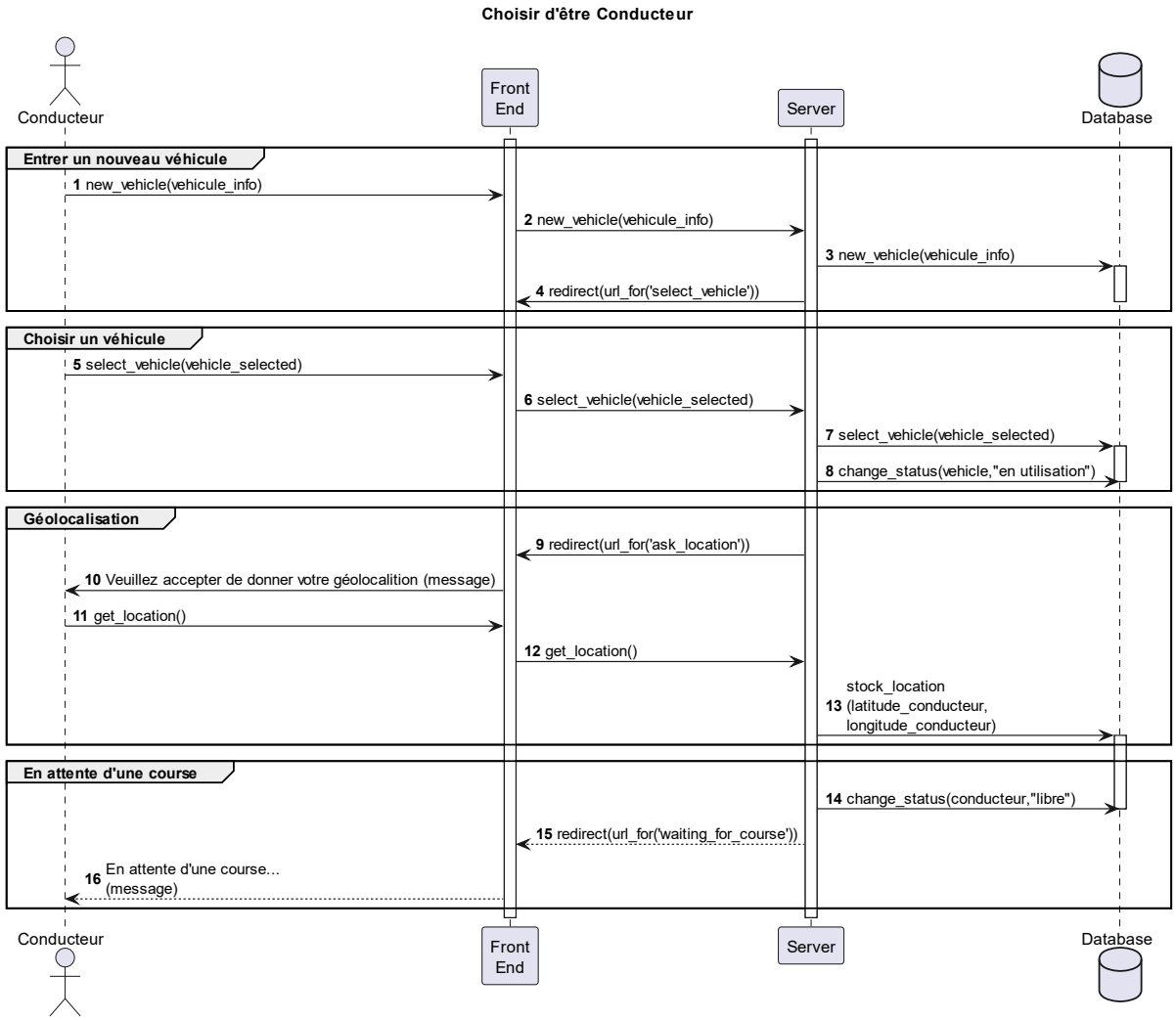

Figure 20 : Diagramme de séquence du conducteur

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 entrer les informations de la course qui sont également envoyées à la base de données par le serveur. Ce dernier 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.

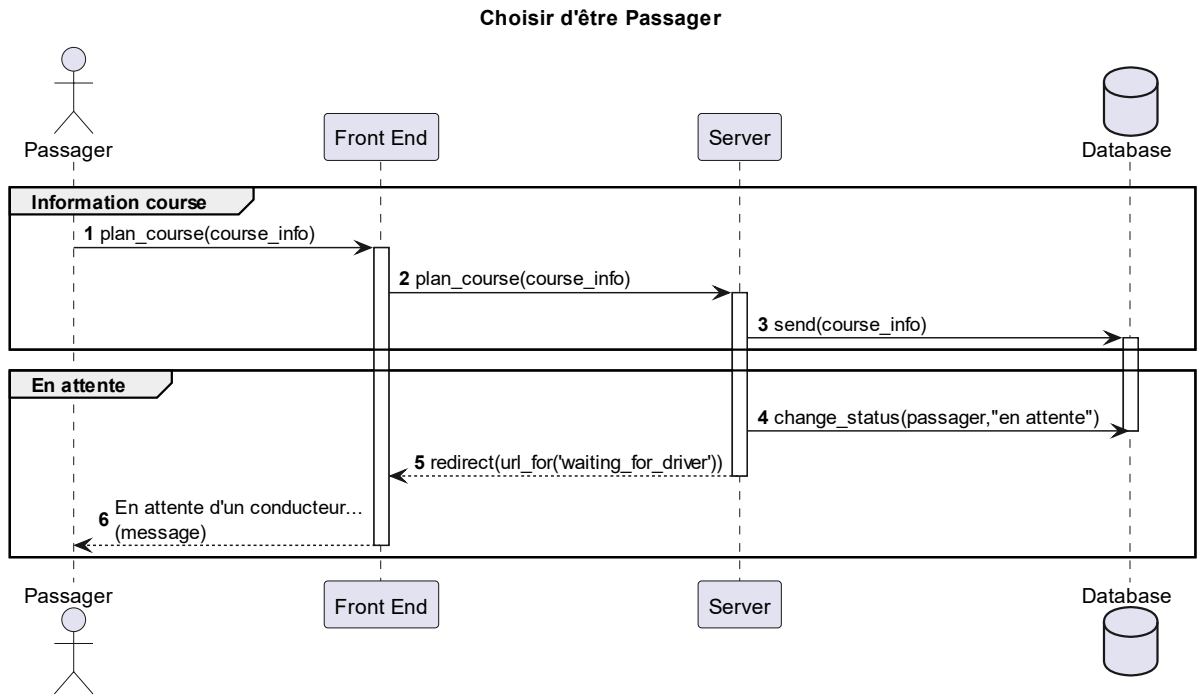

Figure 21 : Diagramme de séquence du passager

Diagramme de séquence de l'envoi et de la course

Dès que le passager envoie la course au serveur -qui l'envoie ensuite à la base de données-, la connexion avec l'API de Google Maps va être demandée et établie. Après lui avoir envoyé les informations de la course nécessaire, celui-ci retourne au serveur les différentes informations nécessaires à l'affichage de l'itinéraire sur une carte. Une fois cela fait, le serveur demande à la base de données les informations de tous les conducteurs disponibles qui répondent aux critères de sélection et les ajoute dans une liste pour ensuite les trier selon les critères de tri, critères vus précédemment.

Une fois cela fait, le serveur envoie la demande de course au premier conducteur présent dans la liste. Si celui-ci l'accepte, le serveur envoie à la base de données le nom du conducteur et le numéro d'immatriculation de la voiture qu'il a décidé d'utiliser, les statuts impliqués sont modifiés et les deux utilisateurs sont redirigés vers la page web de la course. Tandis que si le conducteur n'accepte pas la course, celle-ci est envoyé au deuxième conducteur présent dans la liste, et ainsi de suite.

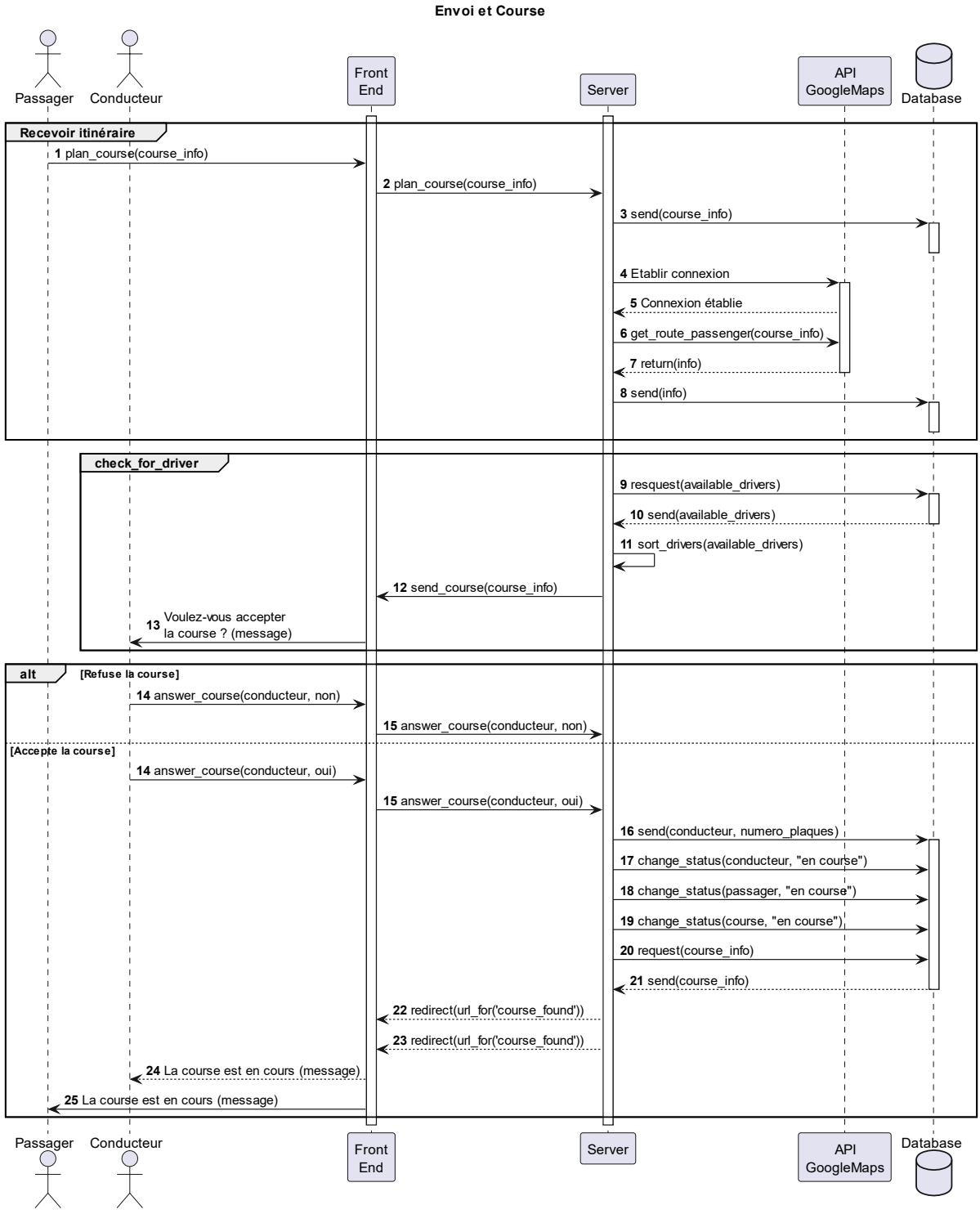

Figure 22 : Diagramme de séquence de l'envoi de la course

Maintenant que les diagrammes ont été présentés et que nous savons comment le système est supposé fonctionner, nous pouvons passer aux modèles algorithmes et ainsi présenter les algorithmes.

Modèles algorithmiques

Ce projet est composé principalement 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 20 kilomètres du passager, doit être disponible et avoir le nombre de place minimum requis pour pouvoir transporter le nombre d'individu indiqué (voir Figure 5). Tous ces conducteurs sont enregistrés dans une liste pour ensuite être triés par rapport à leur rating (le rating doit être le meilleur possible), au nombre de sièges dont ils disposent (le plus proche ce nombre est du nombre de voyageurs, le mieux c'est), ainsi qu'à la distance qui les séparent du point d'origine de la course (plus la distance est faible, mieux c'est). Nous procédons pour cela en un tri hiérarchique [3]. Le fait de trier du moins important au plus important (rating, nombre de sièges et distance) permet de s'assurer que le tri le moins important soit maintenu par le plus important, et donc que le tri résultant de cela soit un tri prioritisant la distance entre le conducteur et l'origine (distance, nombre de sièges et rating). En effet, l'algorithme de tri de python est stable, ce qui veut dire qu'il préserve l'ordre relatif des entrées avec des valeurs égales [2][3], par exemple, une fois le tri du nombre de sièges fait, les valeurs ayant les mêmes nombres de sièges ne vont pas être mélangées même après être passées par le tri suivant.

Si aucun conducteur ne répond aux exigences, le rayon de recherche s'agrandit de 5 kilomètres jusqu'à trouver quelqu'un dans un rayon de maximum 20 kilomètres. Si aucun conducteur n'a pu être trouvé, le passager est prévenu et la tâche se termine sans succès.

```
function sort_drivers
        list available_drivers (From database)

        if available_drivers isn't empty
                while table empty and rayon less or equal to 20km
                        for each driver in rayon + correct number or more seats asked from passenger, add to drivers_to_sort
                        if drivers_to_sort isn't empty, do :
                                sort by rating
                                sort by seat
                                sort by distance
                                sorted_drivers = drivers_to_sort
                        else, add 5 km to rayon

                if sorted_drivers is empty, return "no drivers available in a range of 20km"
                else return sorted_drivers
```

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

Un autre algorithme, utilisant celui précédemment décrit, permet de contacter les conducteurs et de leur proposer la course. Le serveur contacte la base de données pour récolter les informations des différents conducteurs qui sont libres. Ensuite pour chaque conducteur, leur identifiant, nombre de sièges et rating sont récoltés et ajoutés dans une liste qui elle sera utilisé dans l'algorithme de tri. Une fois la liste triée, le premier conducteur de la liste est sélectionné. Le système vérifie que ce conducteur soit toujours disponible. S'il n'est pas disponible, la même démarche est utilisée sur le prochain conducteur. 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, 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.

```
send to the driver the course request
        get info of drivers that are free from database and add to a list

        for driver_id in driver_ids
                driver_info = []
                get and add the driver_id to driver_info
                get and add the number of seats to driver_info
                get and add the distance to driver_info
                add driver_info to available_drivers

        sorted_driver = sort_drivers(available_drivers)

        for driver in sorted_driver
                check if the driver is still free
                if free
                        update everything needed in the database
                        retries = 0
                        while retries is below 30
                                send course
                                if answer == course accepted
                                        return course accepted
                                elif answer == course free
                                        restart the loop for driver in sorted_driver
                                elif answer == course wait
                                        retries +1
                if not free do for driver in sorted_driver loop again
```

Figure 24 : Pseudo-code de l'Algorithme d'envoi

Ces algorithmes nous serviront de plan pour l'implémentation de ce projet, implémentation que nous allons décrire dans la prochaine section.

Implémentation Client/Serveur

L'implémentation Client/Serveur a été principalement faite à l'aide de Python, Flask, HTML, Javascript et de MySQL.

Python est le langage primairement utilisé pour ce projet. En effet, ce langage est utilisé pour le développement web.

Flask est un « micro web framework » écrit en Python qui permet, grâce à ses librairies, d'implémenter des applications web et de traiter des requêtes et réponses HTTP [3]. Il est donc, dans ce projet, la principale structure du code.

HTML, quant à lui, sert à implémenter des pages web et des applications web [3]. Plusieurs pages web ont été créées pour ce projet afin de permettre à l'utilisateur d'être dirigé tout au long du processus de la course dans différentes étapes.

Javascript est un langage utilisé principalement pour le développement web. Il permet de créer des pages web interactives et dynamiques en manipulant les éléments HTML de ces dernières [3] et a été utilisé à cet effet dans ce projet.

MySQL est utilisé pour l'organisation et la gestion de la base de données [3]. Nous pouvons, à l'aide de requêtes, modifier la base de données ou encore demander des données pour ensuite les manipuler dans le code.

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

Figure 25 : Configuration de MySQL

PhpMyAdmin est un interface web permettant la gestion des bases de données en utilisant MySQL [3].

XAMPP est un outil permettant la mise en place d'un serveur Web local comprenant différents composants tel que PHP [3]. Cela permet ainsi l'utilisation de phpMyAdmin. Cela nous a permis de stocker les bases de données localement pour ensuite pouvoir y accéder.

Un serveur Gmail SMTP a été configuré pour envoyer des e-mails automatiquement. Cela nous sert dans ce projet à envoyer les plaintes à l'adresse e-mail créée pour cet usage, dans le but d'être analysées plus tard par un potentiel employé.

```
# Configure email settings to then be ablt to send the reports via email to myself
SMTP_SERVER = 'smtp.gmail.com'
SMTP_PORT = 587
EMAIL_USERNAME = 'projecttestfunny@gmail.com'
EMAIL_PASSWORD = 'oxge rqwe tdpn utaf' # Generated password by Application Password (Google)
EMAIL_RECEIVER = 'projecttestfunny@gmail.com' # To send it to the same address
```

Figure 26 : Configuration du serveur Gmail SMTP

Routes API [4] et Geocoding API [5] (deux des APIs de Google) ont permis de créer un itinéraire en utilisant l'origine et la destination fournies par le passager ainsi que la localisation du conducteur récoltée. En envoyant les adresses de la course à Geocoding API, celui-ci retourne les coordonnées géographiques qui sont ensuite envoyées à Routes API avec d'autres données pour nous retourner la distance, la durée de la course ainsi qu'une ligne encodée, qui une fois associée à une carte, indique l'itinéraire à suivre.

L'organisation des différentes fonctions du code peut être schématisée comme ceci. Les noms des parties font références aux fonctions contenues dans la partie Flask du code.

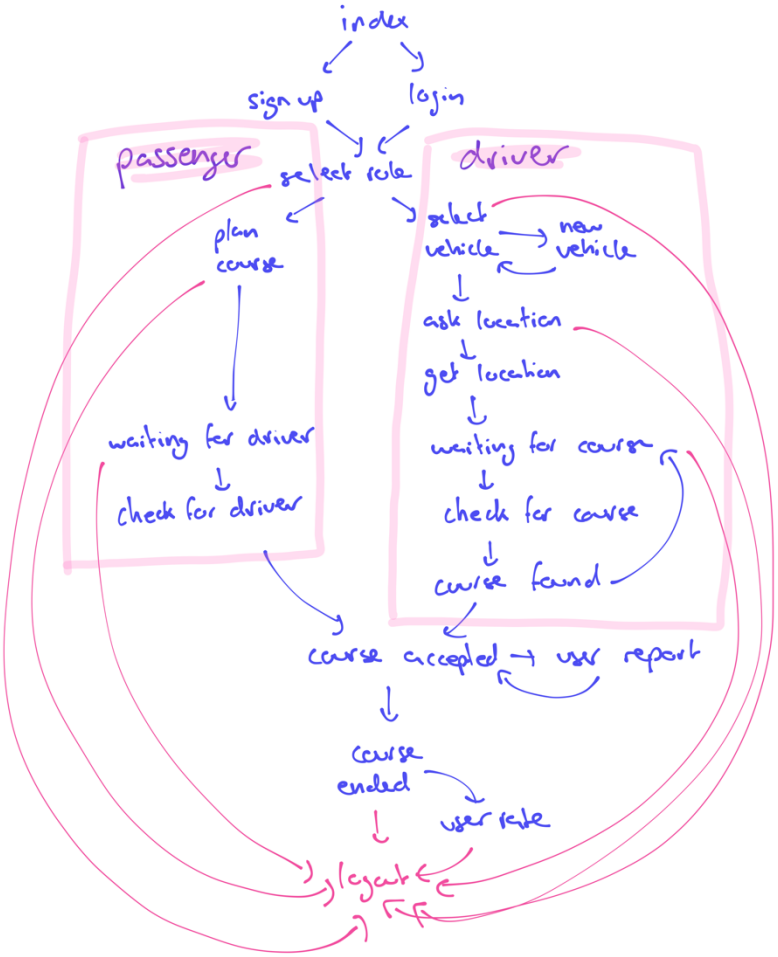

Figure 27 : Organisation des fonctions de la partie Flask

La première partie est l'index. Cette partie permet à l'utilisateur de choisir entre créer un compte ou se connecter. Nous pouvons voir cette page comme la page d'accueil. En fonction de son choix, l'utilisateur devra entrer différentes données. Ensuite vient la page de sélection de rôle. L'utilisateur a le choix entre « passager » et « conducteur ». Selon son choix, iel ne passera pas par les mêmes étapes.

S'iel choisit d'être un passager, iel devra entrer les informations de la course souhaitée et ensuite attendre qu'un conducteur accepte la course. Dans le cas d'un conducteur, iel sera invité à sélectionner un véhicule, ou en ajouter un à sa collection. Une fois cela fait, iel devra accepter de partager sa localisation. Celle-ci sera utilisé pour la création de l'itinéraire. Une fois cette condition acceptée, iel sera mis en attente d'un match. S'iel convient au critère d'une course, la course lui sera proposée et iel pourra l'accepter ou la refuser. Si le conducteur la refuse, iel sera redirigé sur la page d'attente. Au contraire, si le conducteur accepte la course, le passager en sera informé et iels seront tous deux transférés sur la page de course. Une fois sur celle-ci, iels pourront choisir d'envoyer une plainte contre son partenaire de course ou indiquer que la course est terminée en cliquant sur le bouton. Une fois la course terminée, iels sont invitées à noter leur binôme s'iels le désirent. Dans tous les cas, ils sont ensuite déconnectés. Les utilisateurs peuvent également se déconnecter lors de la plupart des étapes, cela effacera leur rôle ainsi que

marquera la course comme étant « abandonnée » si celle-ci a été demandée et si le passager se déconnecte.

« Check for driver » et « Check for course » contiennent les algorithmes de tri ainsi que de vérification de course. Ceux-ci sont conçus de boucles pour permettre une vérification des états des conducteurs et de la course en permanence jusqu'à ce qu'un match soit trouvé et accepté.

La partie « Plan course » contient quant à elle la requête faite aux APIs de Google Maps. De ce fait, les informations de la course obtenue par le passager sont directement envoyées aux différents APIs pour ensuite recevoir en réponse les informations manquantes. Une fois cela récoltées, toutes ces informations sont envoyées à la base de données pour qu'elles soient enregistrées dans la table « course ».

Vous pouvez ici voir un exemple tiré du code qui utilise Python, Flask et MySQL pour permettre à un utilisateur de se créer un compte.

```python
@app.route('/signup/', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        # Get the information from the form
        new_email = request.form['email']
        new_password = request.form['password']
        new_firstname = request.form['firstname']
        new_lastname = request.form['lastname']
        dob_str = request.form['dob']
        dob_date = datetime.strptime(dob_str, '%Y-%m-%d')
        dob_db = dob_date.strftime('%Y-%m-%d')

        try:
            with mysql.connection.cursor() as cur:
                # Check if the email already exists => it should not exist
                cur.execute("SELECT id_utilisateur FROM utilisateur WHERE adresse_email_utilisateur = %s", [new_email])
                existing_user = cur.fetchone()

                if existing_user:
                    flash("This email address is already used, please try another one.", "error")
                    return redirect(url_for('signup'))

                # Insert the information into the utilisateur table
                cur.execute(
                    """INSERT INTO utilisateur (adresse_email_utilisateur, mdp_utilisateur, prenom_utilisateur,
                    nom_utilisateur, date_naiss_utilisateur) VALUES (%s, %s, %s, %s, %s)""",
                    [new_email, new_password, new_firstname, new_lastname, dob_db]
                )
                mysql.connection.commit()

                # Get user_id that will be used in the session and redirect to select_role
                cur.execute("SELECT id_utilisateur FROM utilisateur WHERE adresse_email_utilisateur = %s", [new_email])
                user_id = cur.fetchone()[0]
                session['user_id'] = user_id
                return redirect(url_for('select_role'))

        except MySQLdb.Error as e:
            print(f"MySQL Error: {e}")
            flash("An error occurred while signing up. Please try again.", "error")
        except Exception as e:
            print(f"Error: {e}")
            flash("An unexpected error occurred. Please try again.", "error")

    return render_template('signup.html')
```

Figure 28 : Fonction "SIgn up"

Les informations sont collectées directement de l'utilisateur à l'aide d'un formulaire sur une page web (Figure 29) pour ensuite être traitées. Dans un second temps, une requête est envoyée à la base de données qui lui demande en réponse l'identifiant de l'utilisateur lié à l'adresse e-mail que l'utilisateur a entré. Si la réponse n'est pas vide, cela veut dire que cette adresse e-mail est déjà

connue de la base de données et donc qu'un nouveau compte utilisant cette même adresse e-mail ne peut pas être créé. Dans le cas contraire, les informations sont envoyées à la base de données pour qu'elles y soient enregistrées. Une fois cela fait, l'identifiant donné à l'utilisateur est gardé en mémoire grâce à une session (une variable qui peut être accédée depuis plusieurs pages) et ensuite l'utilisateur est redirigé vers la page de sélection de rôle.

L'image suivante montre la page HTML avec une partie Javascript qui est utilisée par la partie de code précédente pour obtenir les informations nécessaires à la création d'un compte utilisateur.

templates > signup.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Ride Quest - Sign Up</title>
</head>
<body>
    <h1>Sign Up</h1>
    {% with messages = get_flashed_messages(with_categories=true) %}
        {% if messages %}
            <ul>
            {% for category, message in messages %}
                <li class="{{ category }}">{{ message }}</li>
            {% endfor %}
            </ul>
        {% endif %}
    {% endwith %}
    <form action="/signup" method="post">
        <label for="email">Email address:</label>
        <input type="email" id="email" name="email" required><br>
        <label for="firstname">First Name:</label>
        <input type="text" id="firstname" name="firstname" required><br>
        <label for="lastname">Last Name:</label>
        <input type="text" id="lastname" name="lastname" required><br>
        <label for="dob">Date of Birth:</label>
        <input type="date" id="dob" name="dob" required><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required><br>
        <button type="submit">Sign Up</button>
    </form>
</body>
</html>
```

Figure 29: code HTML de la fonction "Sign up"

Comme indiqué, une partie de ce code est écrite en Javascript. Cela permet d'afficher des messages d'erreur pour informer l'utilisateur de ce qu'il ne marche pas correctement (par exemple qu'un compte utilisant cette même adresse e-mail existe déjà).

Usage des outils

En plus des outils décrits précédemment, nous en avons utilisé d'autres pour la réalisation de ce projet.

Nous avons utilisé GitLab pour mettre les différents dossiers ainsi que le code 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 schémas conceptuel et logique. yEd a permis de faire les modèles, tandis que Draw.io a été utilisé pour modéliser 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 les diagrammes de séquence à l'aide de code pour rendre leurs constructions plus simples.

Mockaroo a 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.

Evaluation

En général, le projet est en bonne voie. Il n'est certes pas conclu, mais le programme marche, et toutes les sections ont été implémentées comme souhaitées. La principale fonctionnalité à laquelle il faudra penser est la mise en place du système de transport public, car dans ce projet, il n'est possible de faire une course qu'en faisant du co-voiturage, alors que le but était dans un premier temps de permettre à l'utilisateur de choisir entre faire la totalité de la course à l'aide du co-voiturage ou d'accepter de faire une partie de la course en utilisait les transports en commun.

L'évaluation du code a été menée en « trial and error », c'est-à-dire en testant le programme avec différents types de données. Ceux-ci n'étant pas infaillibles, des erreurs non soulevées pourraient encore se trouver dans le code.

Les problèmes rencontrés tout au long de ce projet ont été nombreux. La plus grande complication rencontrée a été le fait que dans la fonction « check_for_driver », la réponse que nous recevions de la requête pour vérifier le statut de la course ne changeait pas alors même que le statut avait bel et bien été mis à jour dans la base de données. Peu importe les changements que nous apportions au code, nous ne pouvions pas modifier cette erreur. Il nous a fallu utiliser ChatGPT pour des indices pour que qu'il était de faire et essayer de corriger l'erreur pendant 3 ou 4 jours. Le problème venait sûrement du fait que la connexion n'était pas rafraichie et que la requête lisait autre chose que la donnée qui a été « commit ». Une autre difficulté rencontrée a été de comprendre comment faire en sorte que « waiting_for_course », « course_found » et « waiting_for_driver » s'écoutent les uns et les autres pour voir si une course a été trouvée vis-à-vis du conducteur et si un conducteur a accepté ou refusé la course qui lui avait été assignée. Nous avons décidé d'utiliser les statuts de la course et du conducteur pour ensuite les analyser et ainsi comprendre si un conducteur a été sélectionné pour une course et si ce dernier l'a acceptée. Vous pouvez ici voir un des premiers schémas pensé lors de la mise en place de ces 3 fonctions pour comprendre comment cela allait être implémenté. Un dernier problème important

à mentionner est le fait que MySQL a arrêté de fonctionner plusieurs fois sur mon ordinateur, pour une raison inconnue. La première fois que cela est arrivée, nous avons essayé de résoudre le problème pendant plusieurs jours, comme désinstaller MySQL, avant de tout simplement supprimer XAMPP pour tout réinstaller. Cela a fait que nous avions perdu notre base de données initiale et une seconde a dû être créée et les données-tests ont dû être importées une deuxième fois. Cela s'est reproduit plusieurs fois, et au bout de quelques erreurs de ce type, une solution a été trouvée, qui a été de reprendre la dossier

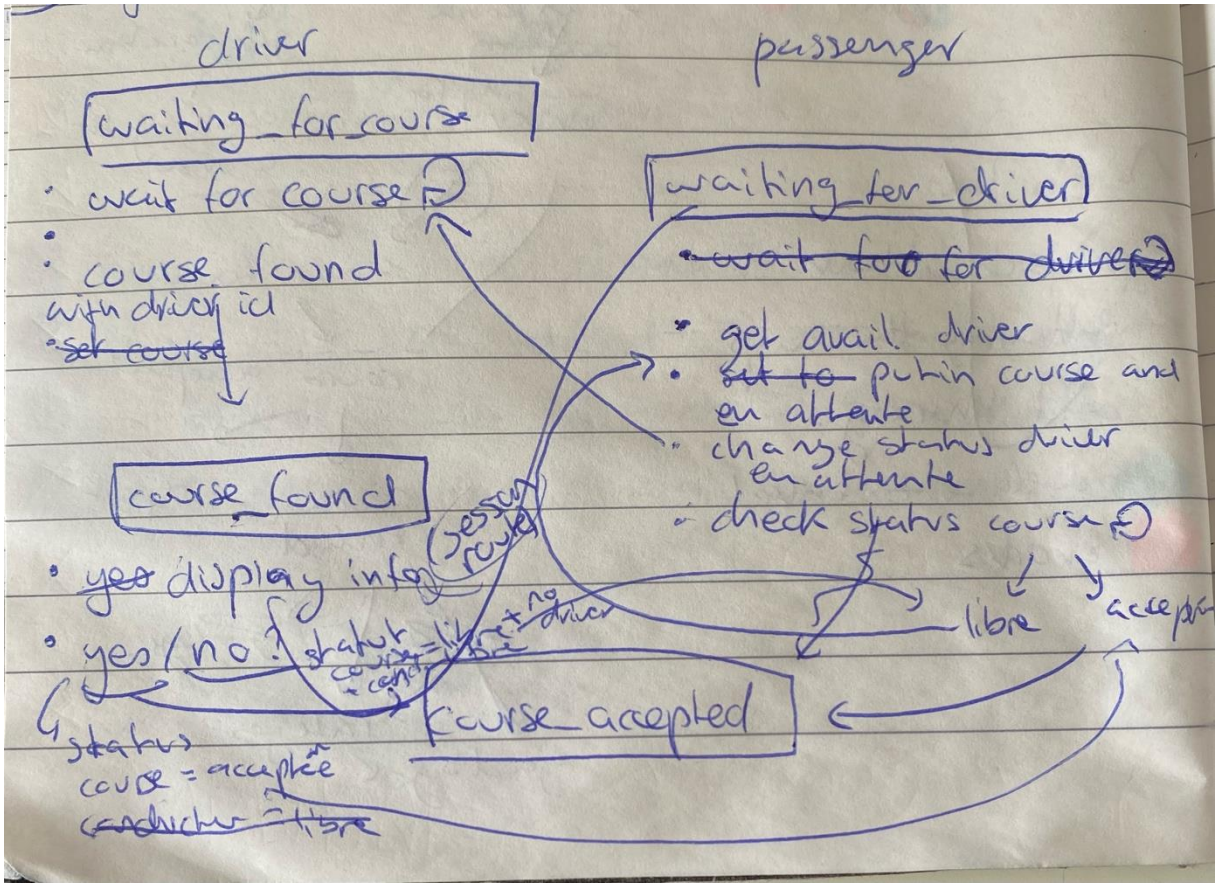

Figure 30 : Schéma des fonctions "waiting for driver", "waiting for course" et "course found"

Un grand nombre d'améliorations peuvent être recensées. Comme mentionné précédemment, l'ajout de la possibilité de choisir les transports en commun pour voyager sera une fonctionnalité importante à rajouter. Nous pouvons aussi nommer l'utilisation de la localisation GPS des utilisateurs en temps réel pour ensuite les afficher sur une carte dès que la course commence pour que les deux partis sachent où leur binôme se trouve à tout moment. Une meilleure gestion de la base de données et des données peut également être envisagée. L'analyse de l'âge des utilisateurs et de leur permis de conduire pour savoir si ceux-ci sont en droit d'être conducteur se verra nécessaire si ce programme veut être publié, principalement pour garantir la sécurité des utilisateurs ainsi que la conformité à la loi. La sécurité des mots de passe pourrait également être amélioré en utilisant par exemple une fonction de hashage. La création d'une application se verrait également plus pratique que l'utilisation de pages web, surtout quand l'utilisation se veut « user-friendly ». En effet, le fait de ne pouvoir utiliser un site web à la page d'une application est bien moins pratique qu'une application pour ce qui est des utilisateurs, car ceux-ci sont déjà

habitués à utiliser des applications sur leur appareils mobiles pour se déplacer et naviguer (Uber, Google Maps, SBB, etc.) et ceci leur permettrait de vérifier le statut de la course en tout temps.

Les succès de ce projet n'en restent pas moins importants et nombreux. Premièrement, le programme fonctionne et la base de données permet un stockage des données nécessaires à l'initiation et au développement de la course. L'utilisation de l'interface est simple et ne demande pas de connaissances générales et informatiques particulières. Tous types de personnes peut utiliser ce programme en toute simplicité. Le code en lui-même est simple et compréhensible, ce qui permet à une personne qui s'intéresse au backend de comprendre comment cela a été créée, le code étant clair et commenté.

Cela marque donc la fin de notre projet, nous en sommes satisfaits.

Conclusion

Dans ce rapport, nous avons introduit le projet en mentionnant ses problématiques, présenté les différents modèles d'analyse des objectifs, de sa conception, y compris des schémas conceptuel et logique, expliquer les modèles du système et algorithmiques, parler de l'implémentation, énuméré les outils utilisés tout au long de ce dernier et l'évaluer.

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 ont quant à eux permis de voir à quoi pourrait ressembler le code et ainsi nous permettre de créer la logique derrière celui-ci. La conception et l'implémentation ont rendu la mise en place du programme ainsi que la mise à bien de celui-ci sans que de majeurs défauts ne soient recensés. L'évaluation quant à elle permet de visualiser l'ensemble du projet pour le critiquer et indiquer les améliorations possibles. Nous avons par exemple déterminé que l'implémentation d'une application mobile sera désirable ou encore que celle-ci se verrait grandement bénéficier de l'ajout d'un système de transport en commun.

Ce projet a été dans l'ensemble un succès, même s'il laisse place à certaines améliorations et modifications.

Références

[1] Prof. Jolita Ralyté, cours « Analyse des objectifs 2022 »

[2] A. Dalke and R. Hettinger. (2024, Jul. 20). Sorting Techniques [Online]. Available: https://docs.python.org/3/howto/sorting.html

[3] Chat GPT

[4] Google Maps Plateform. (2024, Jan. 09). Routes API [Online]. Available: https://developers.google.com/maps/documentation/routes

[5] Google Maps Plateform. (2024, Jan. 09). Geocoding API [Online]. Available: https://developers.google.com/maps/documentation/geocoding

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