



Implémentation d'un service de mobilité partagée intelligente.

Meskine Wassif

Supervisé par Bah Slimane

Ecole Mohammadia
des ingénieurs

Résumé

Abstract in French

Abstract

Abstract in English

ملخص

Remerciements

Introduction générale

Depuis le milieu du 19^e siècle, les agglomérations n'ont cessé de s'agrandir et de s'étendre. C'est la révolution industrielle. Donc il va falloir repenser nos modes de transport.

Avec l'avènement de la voiture, chaque individu peut se procurer le plaisir de se déplacer confortablement entre son milieu de travail et son domicile. Résultat, Nos villes s'engorgent, il y a de plus en plus de voitures, de bouchons, de consommation d'énergie et de pollution.

Solution, Les services à mobilité partagée ou covoiturage qui s'imposent comme complémentaires et rivaux aux transports en commun offrent :

- Un mode de transport écologique et économique.
- Une réduction de la moyenne quotidienne du temps de déplacement.
- Un aspect social et convivial a nos déplacements.

Sous cet angle de vision futuriste s'inscrit ce PFE au sein de la société Peoplin et qui consiste à modéliser et réaliser une application de covoiturage.

Conséquence, ce document synthétise le travail effectué.

Table des matières

Résumé	i
Remerciements	iv
Introduction générale	iv
1 Définition du projet	1
1.1 Présentation de l'organisme	1
1.2 Contexte	2
1.3 Problématique	3
1.4 Objectifs	4
1.5 Étude de l'existant	5
1.6 La méthodologie de démarche suivie du PFE	5
1.6.1 Mission	5
1.6.2 Démarche du gestion de projet	6
1.6.3 Planification	8
1.6.4 Outils de collaborations	8
2 Analyse des besoins	10
2.1 Identification des acteurs	10
2.2 Spécification des besoins	11
2.2.1 Besoins fonctionnels	11
2.2.2 Besoins non fonctionnels	12
2.3 Modélisation métier	13

2.3.1	Diagramme de cas d'utilisation	14
2.3.2	Diagrammes de séquences	14
2.3.3	Conception graphique	19
2.3.4	Diagramme de classe d'analyse	19
2.4	Aperçu de l'algorithme de covoiturage	21
2.4.1	Étapes de l'algorithme	21
2.4.2	Processus du matching	22
3	Implémentation	24
3.1	Architecture	24
3.1.1	Architecture globale	24
3.1.2	Architecture client side	25
3.2	Technologies utilisées	28
3.3	Justification des choix techniques	30
3.3.1	partie cliente	30
3.3.2	Partie serveur	31
3.3.3	Choix du système de gestion de données	31
3.4	Quelques interfaces utilisateur	32
4	Tests et évaluation	37
4.1	Code testable	37
4.2	Intérêts des tests	38
4.3	Outils de tests	38
4.4	Les types de test	39
4.5	Évaluation	39
5	Résumé et réflexions	41
	Bibliographie	41

Liste des tableaux

2.1	Description textuelle de l'inscription	15
2.2	Description textuelle de terminer le KYC	16
2.3	Description textuelle du démarrer un trajet instantané	17
2.4	Description textuelle du demander un covoiturage instantané	18

Table des figures

1.1	organigramme de l'équipe de projet	2
1.2	Le cycle de vie d'un projet Scrum	7
2.1	diagramme de contexte statique	11
2.2	diagramme de cas d'utilisation	14
2.3	diagramme de séquence inscription	15
2.4	diagramme de séquence de validation du processus KYC	16
2.5	diagramme de séquence démarrer un trajet instantané	17
2.6	diagramme de séquence demander un covoiturage instantané	18
2.7	prototypes d'interfaces graphiques	19
2.8	diagramme de classe métier simplifié	20
2.9	logigramme du processus de matching	21
2.10	processus de matching dans le cas d'une seule offre	23
3.1	architecture de l'application	26
3.2	le patron de conception target-action	27
3.3	patron de conception mvc	27
3.4	le patron de conception MVVM	28
3.5	HERE Maps API vs Google Maps API	30
3.6	Performances CPU par dollar de toutes les machines virtuelles testées . . .	31
3.7	Les offres planifiés	33
3.8	Les offres instantanés	34
3.9	Utilisateur	35

3.10 L'inscription	36
4.1 Xcode Instruments	40

Liste d'abréviations

GES Gaz à effet de serre.

MVC Modèle vue contrôleur.

MVVM Modèle vue vue modèle.

Chapitre 1

Définition du projet

Comme tout projet a un cycle de vie qui débutera par une phase de lancement qui peut avoir une signification différente pour chaque organisation mais qui généralement clarifie trois points essentiels :

- la problématique.
- les objectifs.
- la démarche à suivre.

Nous allons dans ce chapitre présenter l'organisme d'accueil afin d'avoir une idée sur les parties prenantes, par la suite on va définir notre problématique, après on fixe nos objectifs, ensuite on fait une étude de l'existant, enfin on explique La méthodologie de démarche suivie du PFE.

1.1 Présentation de l'organisme

Dans cette section, nous présentons l'organisme d'accueil de ce projet qui est la société Peoplin.

Peoplin est une start-up technologique, créée en 2018 sous la dénomination CORE TECHS, et ayant pour but de fournir des services financiers digitalisés, innovants et à forte valeur ajoutée. Un prototype de Workflow décisionnel pour l'octroi de financements entièrement digitalisé et destiné aux nouvelles banques participatives a été élaboré, cependant, le marché n'était pas encore suffisamment mature pour absorber une telle solution, ce qui a

amené à la refonte de l'identité de la start-up, pour aborder un nouveau créneau, celui de la ville connectée.

Ainsi, depuis début 2019, la start-up se lancera dans la réalisation d'une solution de covoiturage dynamique, première brique de sa feuille de route à moyen et long terme, en sous-traitance à une entreprise Indienne, cette dernière effectuera le prototypage du socle global de la solution et livrera une première partie des développements demandés, la deuxième partie des développements va être poursuivie au Maroc à partir de février 2020.

Les parties prenantes impliquées dans ce projet et qui suivent son cycle de vie sont :

- Le product owner : L'entreprise Peoplin.
- Scrum master : Mohammed Salim El Azzouzi.

L'équipe du développement comporte quatre personnes :

- Responsable de la partie mobile IOS.
- Responsable de la partie mobile Android.
- Deux responsables du back-end.

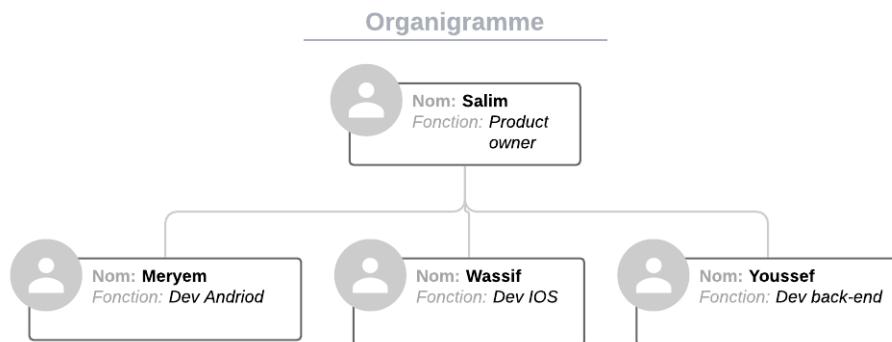


FIGURE 1.1 – organigramme de l'équipe de projet

1.2 Contexte

Ces deux dernières décennies marquent au Maroc une forte augmentation de demandes du déplacement, ce qui pousse les gouvernements à constamment améliorer les moyens de transports pour répondre à ce besoin.

Mais, malgré les efforts ces moyens de transport n'arrivent pas à couvrir la demande toujours en croissance.

Résultat, la voiture individuelle reste le moyen le plus pratique et confortable pour se déplacer soit dans les villes, soit dans les zones non urbaines. Néanmoins, la mobilité humaine accrue combinée avec la forte utilisation de la voiture engendrent des effets néfastes sur la qualité de voyage et soulèvent des questions environnementales.

En effet, l'usage abusif de la voiture est source de stresser, de charges personnelles de plus en plus en hausse (à cause du prix du carburant, de l'entretien ...) et des gazes à effet de serre (GES) .

Une solution qui s'impose comme immédiate est le partage de la voiture personnelle et qui permet à un chauffeur de la partager avec des passagers potentiels, afin d'effectuer le trajet ensemble.

Donc, notre solution consiste à automatiser cette opération, en optimisant les temps de détours du chauffeur tout en laissant un maximum de chance aux passagers de trouver un conducteur qui va leurs amenés vers leurs destinations tout en nouant les liens sociaux qui sont en déclin.

1.3 Problématique

Dans le but de procurer un service de covoiturage répondant aux besoins de déplacement et palier aux problèmes écologiques, sociaux et financiers cités auparavant.

Les solutions qui sont présentes sur le marché ou qui font l'objet de recherche des industriels offrent des services assez limités qui se résument dans des trajets planifiés ou des trajets entre les grandes villes.

En effet, un utilisateur doit chercher une offre déjà planifiée et envoyer une demande au publicateur, et récupérer ainsi les coordonnées des covoitureurs pour une éventuelle prise de contact pour un trajet bien déterminé auparavant, ce qui n'automatise pas l'opération. Dans ce cadre, en se basant sur les attentes des utilisateurs, nous nous intéressons à trouver une solution informatique qui répond aux problèmes mentionnés avant et qui ne sont pas la priorité des créateurs de solutions existantes, et qui peuvent se résumer comme

suit :

- dépenses du déplacement en augmentation.
- trajets quotidiens ennuyeux et liens sociaux en déclin.
- l'embouteillage et la pollution.
- sécurité soit des passagers soit des conducteurs.

1.4 Objectifs

Dans l'optique d'apporter de nouvelles fonctionnalités et remédier aux insuffisances et limites explicitées auparavant. Notre application permettra aux covoitureurs de faire des économies, aux passagers de se déplacer conformément à leurs préférences et en toute sécurité et à réduire l'émission des gaz polluants.

Effectivement, l'application permet à un chauffeur de faire un choix entre des offres instantanées (pratiques dans les zones urbaines dépassant pas 30 km) et des offres entre les villes. En plus toute personne ayant l'âge légal, un permis peut diviser les charges du déplacement et de l'entretien de sa voiture avec d'autres potentiels passagers. Et à un passager de se déplacer avec un prix moins cher que les transports en commun sans perdre son temps et en choisissant ses préférences (fumeur, bagages, sexe du conducteur ...). Ainsi il permet de fournir une offre en quasi-temps réel à l'« usager covoitureur ». La personne souhaitant effectuer un itinéraire en covoiturage contacte le service quelques secondes avant son départ. Le service va alors chercher le conducteur adéquat qui est en mesure d'offrir le covoiturage souhaité sur l'itinéraire demandé.[7]

L'application propose les chemins les plus optimales avec des temps de détour pour prendre un passager, courts et rapides. Dans le but de réduire la consommation d'énergie et par conséquence, réduire l'émission des gaz polluants. En gros, l'application devrait respecter les exigences listées ci-dessous.

- Vérification de la carte d'identité(CIN) pour chaque utilisateur pour des raisons de sécurité.
- Publier une offre de covoiturage planifiées ou lancer une course instantanée.
- Trouver des offres en temps réel ou réserver dans les trajets planifiés.

- Afficher les trajets optimaux.
- Effectuer des transactions monétaires.

1.5 Étude de l'existant

Dans l'intention de faire une analyse des points forts et faibles des solutions déjà présentes actuellement et en dégager des améliorations. Nous avons recueilli toutes informations qui nous paraient utiles, ce qui nous a obligé de faire le passage qui concrétise le premier contact d'un étudiant avec un domaine totalement ignoré.

Ainsi, pour faire cette tache nous avons combiné les différentes informations et qui sont :

- Exploitation des documents relatifs à la législation du covoiturage au Maroc.
- Étude des interfaces graphiques et des fonctionnalités des application comme : Careem, Heetch, Roby, Yassir.
- Recueillir des avis des utilisateurs de covoiturage sur les réseaux sociaux afin de mettre en scène leurs expériences.

Par la suite, ces donnés sont classées et représentées sous forme plus synthétique afin de s'inspirer et d'en tirer le meilleur.

1.6 La méthodologie de démarche suivie du PFE

1.6.1 Mission

Ma mission au sein de l'équipe peut se résumer dans la conception et implémentation du service de mobilité partagée, en temps réel et en mode planifié et enrichissement du socle commun par les fonctionnalités nécessaires au bon fonctionnement du service.

Ainsi que la gestion de projet en mode Agile, depuis l'expression du besoin client, conception, prise en charge des développements, tests de qualification et participation à la publication sur l'Apple Store.

Résultats attendus et plan d'action :

- Analyse du besoin client, exprimé par le Product Owner.

- Analyse collaborative du Product Backlog et priorisation de la liste des fonctionnalités.
- Définition collaborative des Sprints fonctionnels et des Artefacts du projet.
- Développement de l'application en respectant les contraintes et les défis du développement mobile.
- Tests unitaires et les tests fonctionnels pour vérifier que l'application répond bien au besoin exprimé et qu'elle convient à l'utilisateur final.
- Soumission à l'Apple Store
- Correction des bugs et mise à jour de l'application.

Par la suite nous allons citer quelques pré-requis techniques exigés pour aboutir au résultat souhaité :

- Maîtrise du langage Swift et de l'environnement de développement intégré(IDE) Xcode, et du Kit de développement IOS.
- CocoaPods qui est un gestionnaire de dépendances pour automatiser l'intégration d'autres bibliothèques dans notre application.
- Utilisation des API Rest et de la plate-forme AWS Cloud.
- Here qui est un service de cartographie.

1.6.2 Démarche du gestion de projet

Pour le volet de gestion de projet, nous avons adopté une méthode Agile plus spécifiquement Scrum qui est un cadre ou canevas (framework en anglais)[5] simple et efficace qui repose sur 3 piliers :

- Transparence : garantir que toutes les informations relatives à la bonne compréhension du projet sont bien communiquées aux membres de votre équipe et aux différentes parties prenantes.
- Inspection : vérifier à intervalles réguliers que le projet respecte des limites acceptables et qu'il n'y a pas de déviation indésirable par rapport à la demande de votre client.
- Adaptation : encouragez la correction des dérives constatées et proposez des chan-

gements appropriés afin de mieux répondre aux objectifs de votre gestion de projet. Donc, le processus choisi pour le développement du projet est empirique, itératif, incrémental et agile :

- empirique : l'inspection quotidienne de l'état du projet qui oriente les décisions.
- itératif : découper le projet en plusieurs cycles identiques ou itérations. Vous vous rapprocherez graduellement du produit ou du service final afin de limiter les risques d'erreurs.
- incrémental : La partie du projet a réalisée doit être utilisable. Vous pouvez donc livrer votre client régulièrement avec des fonctionnalités complètes.
- agile : vous impliquez votre client et vos utilisateurs dans votre gestion de projet. Vous choisissez toujours des méthodes pragmatiques et adaptatives pour être plus réactif aux demandes.

Donc pour respecter ce cadre. Premièrement, nous avons élaboré le Backlog Scrum qui est destiné à recueillir tous les besoins du client que l'équipe projet doit réaliser. Il contient donc la liste des fonctionnalités intervenant dans la constitution d'un produit. Deuxièmement, nous avons découpé la liste des fonctionnalités sur des intervalles de temps limités qu'on va appeler Sprint. Et on attribue à chaque Sprint un nombre de fonctionnalités puis on applique le cycle de vie classique d'un projet informatique qui va de la modélisation jusqu'aux tests.

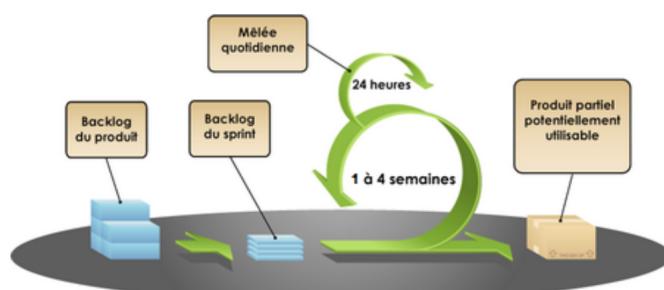


FIGURE 1.2 – Le cycle de vie d'un projet Scrum

1.6.3 Planification

Pour mener à bien notre travail nous avons fixé quelques itérations qui nous semblent nécessaires et prioritaires pour entamer le développement de la solution. Après concertation avec mon encadrant qui est le product owner nous avons découpé les deux itérations ou Sprints en plusieurs user storie qui est une description simple d'un besoin et qui seront listés ci-dessous :

- Sprint 1 (en relation avec le chauffeur) sur **2 semaines** :
 - Démarrer un trajet instantané
 - Recevoir et confirmer la prise en charge d'un passager.
 - Recevoir et rejeter la prise en charge d'un passager.
 - Recevoir, confirmer la prise en charge d'un passager et annulation en cours de route.
- Sprint 2 (en relation avec le passager) sur **2 semaines** :
 - Demande d'un covoiturage instantané.
 - Annulation d'un covoiturage instantané post-confirmation

1.6.4 Outils de collaborations

Afin de favoriser un meilleur travail en équipe nous avons décidé de travailler avec les outils nous allons présenter brièvement ci-après.

Slack

Slack fonctionne à la manière d'un chat IRC organisé en canaux correspondant à autant de sujets de discussion. La plateforme permet également de conserver une trace de tous les échanges, et permet le partage de fichiers au sein des conversations et intègre en leur sein des services externes comme GitHub, Dropbox, Google Drive ou encore Trello pour centraliser le suivi et la gestion d'un projet.

Trello

Trello est un outil de gestion de projet en ligne. Il repose sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement. Afin de visualiser graphiquement nos user stories et l'état d'avancement des Sprints.

Git

Git est un logiciel de gestion de versions, ce qui veut dire qu'il permet de stocker l'ensemble des fichiers sources, en conservant la chronologie de toutes les modifications qui ont été effectuées.

L'outil est décentralisé ce qui veut dire que chacun des membres à sa propre version sur le local, et qu'il peut modifier à sa guise. Mais après avoir terminé le développement d'une fonctionnalités, il synchronise avec les versions des autres membres.

Conclusion. Ce premier chapitre avait pour finalité la présentation générale du projet notamment le contexte du projet, la problématique, le cadrage, pour enfin conclure avec la méthodologie adoptée dans le développement.

Chapitre 2

Analyse des besoins

Ce chapitre rassemble les ressources pour mener l'expression des besoins. Nous allons commencer par l'identification des acteurs, puis nous passons à la spécification des besoins et faire une analyse en s'appuyant sur UML, enfin nous donnons un aperçu sur l'algorithme de jumelage qui est le cœur de notre métier.

2.1 Identification des acteurs

Dans cette section on va décrire **QUI** utilisera notre système. On commence dans un premier lieu par considérer notre système comme une boite noire pour répondre à la question qui va interagir avec notre logiciel ?

les entités ou acteurs qui entre en jeu sont :

- Utilisateur : Après son inscription, il peut consulter les offres de covoiturage planifiées, mais aucun autre service ne sera pour lui opérationnel tant qu'il n'a pas terminé le processus KYC (Know Your Customer), on détaille par la suite ce que le KYC dans notre cas.
- Chauffeur : doit pourvoir ajouter des offres de covoiturage planifiées ou instantanées.
- Passager : doit pouvoir faire des demandes de covoiturage instantanées et de réserver sa place dans les autres planifiées.

Il convient de noter que nos acteurs ont plus qu'un rôle qu'une personne physique, puisque

une personne physique peut à la fois être un chauffeur et un passager.

La figure 2.1 donne une représentation graphique du diagramme du contexte statique sans rentrer dans le détail des cardinalités.

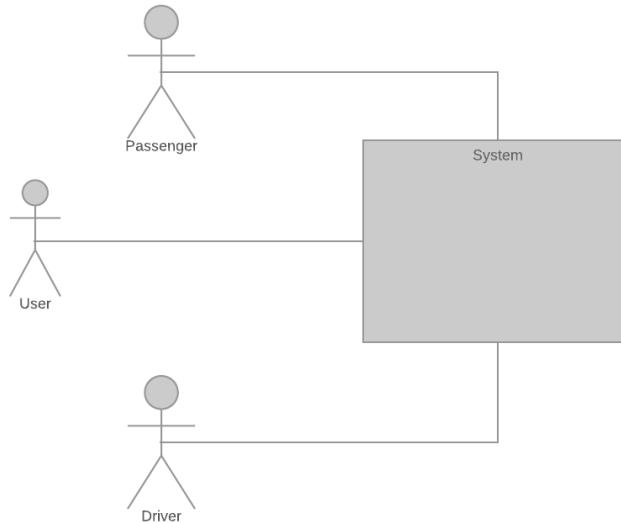


FIGURE 2.1 – diagramme de contexte statique

2.2 Spécification des besoins

2.2.1 Besoins fonctionnels

Après la définition des acteurs de notre application, on détermine les besoins de chaque acteur pour répondre à la question **QUI** devra pouvoir faire **QUOI** ?

L'expression de la liste des fonctionnalités est comme suit :

- **Authentification** : chaque utilisateur doit pouvoir s'inscrire et activer son compte en vérifiant son numéro du téléphone et son émail.
- **Consultation des offres** : chaque utilisateur doit pouvoir après activation du compte consulter la liste des offres planifiées.
- **Terminer le KYC** : l'utilisateur doit pouvoir prouver son identité afin que l'ensemble des services seront accessibles (Identification doit être automatisée).

Tout service de ceux qui suivent n'est exploitable que si l'utilisateur à terminer le processus KYC.

- **Ajout des offres** : un chauffeur peut publier des offres de covoiturages planifiées.
- **lancement du trajet** : un conducteur peut lancer un covoiturage instantané, l'itinéraire optimal doit s'afficher tout au long de la course. il reçoit aussi des notifications de prise en charge des passagers.
- **Demande de covoiturage** : un passager peut demander un covoiturage instantané (la saisie de la demande par le système doit être en temps réel), ou consulter la liste des covoiturages planifiés afin de réserver sa place.
- **Réseau des utilisateurs** : Les utilisateurs peuvent avoir un réseau d'amis, ainsi que, synchroniser ou inviter leurs camarades dans les réseaux sociaux.
- **Le matching** : l'affectation des conducteurs aux passagers offre un modèle riche qui tient compte non seulement de l'adéquation des itinéraires, mais aussi de l'âge, du sexe et le nombre de valises.
- **Historique des trajets** : tout utilisateur doit avoir la liste de ses trajets planifiés ou qui ont été faits.
- **Historique des places favoris** : tout utilisateur peut garder une liste de place favorites pour faciliter la recherche des positions.
- **Annulation** : tout utilisateur doit pouvoir annuler des voyages.
- **Paiement** : Un passager doit pouvoir payer soit par cash ou par la création d'un portefeuille (Wallet) qui lui permet d'envoyer, recevoir, contrôler ses transactions.

Cette liste n'est pas exhaustive, et certains processus métiers ont été éliminés pour des raisons du degré de leurs importances et d'allégement de ce document.

2.2.2 Besoins non fonctionnels

Outre les besoins fonctionnels, un système d'informations doit répondre à un ensemble d'exigences non fonctionnelles.

- **Performance** : le chargement rapide de l'application, ouverture d'écran, des délais de rafraîchissement et le temps de réponse de l'algorithme de matching.
- **Disponibilité** : l'application doit être opérationnelle à n'importe quel moment.
- **Architecture** : le respect du modèle du développement logiciel imposé par Apple.

- **Tests** : les tests unitaires doivent être écrits et couvrir 90% du code.
- **Ergonomie** : la densité d'éléments sur les écrans, la disposition et le flux, les couleurs.
- **Interface graphique** : les interfaces graphiques doivent être responsive c'est à dire, elles s'adaptent aux différentes tailles d'écran des appareils iOS. (Dans notre cas on va développer notre application que pour le mode portrait)
- **Sécurité** : les différents échanges entre les composants de l'application doivent être chiffrés. l'accès à l'API se limite à nos applications clientes. protéger les données de l'utilisateur exemple : infos personnelles.
- **Extensibilité** : le projet se développe dans un petit périmètre au sein d'une start-up qui cherche encore son business plan, ce qui implique l'ajout dans l'avenir de plusieurs autres fonctionnalités.

En plus de ces besoins l'application a des défis à surmonter vu que nous développons dans un environnement mobile :

- les petites tailles d'écran.
- la vie de la batterie.
- performances du matériel.

2.3 Modélisation métier

Nous avons identifié les acteurs et un certain nombre de fonctionnalités. Maintenant il est temps de définir en détail chaque fonction et l'attribuer à un acteur. Nous allons tracer le diagramme de cas d'utilisation, quelques diagrammes de séquence système et un diagramme de classe d'analyse.

Vu la méthodologie que nous avons adoptée, nous n'avons pas fait une analyse détaillée des besoins, ce qui veut dire que nous sommes limités aux diagrammes métier sans faire les diagrammes de conception puisque nous étions en communication permanente avec le client (product owner).

2.3.1 Diagramme de cas d'utilisation

Un cas d'utilisation (use case) représente un lot d'actions qui sont réalisées par le système et qui met donc en évidence de quelle façon les acteurs utiliseront le logiciel.

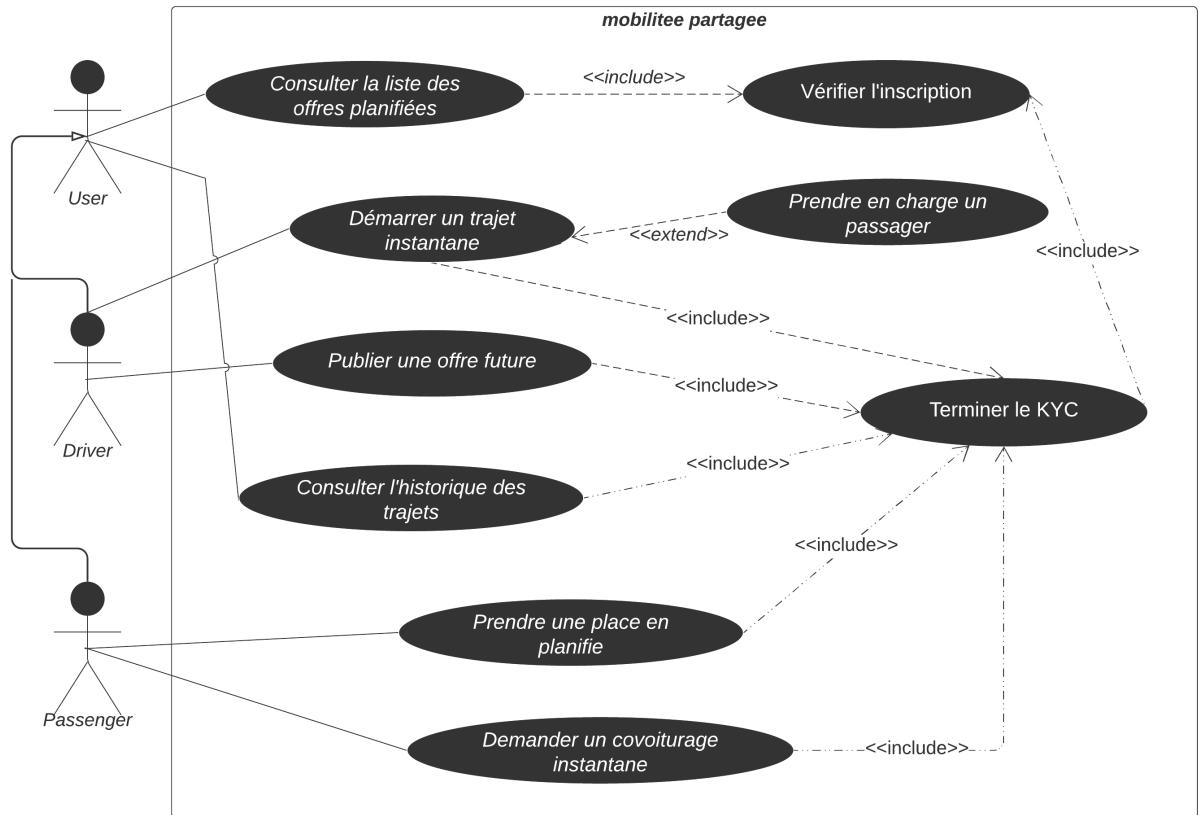


FIGURE 2.2 – diagramme de cas d'utilisation

2.3.2 Diagrammes de séquences

Afin de faire une validation des cas d'utilisation, et compléter le diagramme des use case en mettant en évidence les acteurs et leurs interactions avec le système d'un point de vue temporel, les diagrammes de séquence nous servent de support.

Chaque sous-section ci-dessous contient une fiche descriptive et le diagramme de séquence.

Inscription

Titre	Inscription
Acteur(s)	User
Description succincte	l'inscription doit être possible pour tout utilisateur qui a déjà installé l'application
Pré-conditions	aucunes
Démarrage	l'utilisateur appuie sur le bouton inscription
Post-conditions	création d'un nouveau utilisateur

TABLE 2.1 – Description textuelle de l'inscription

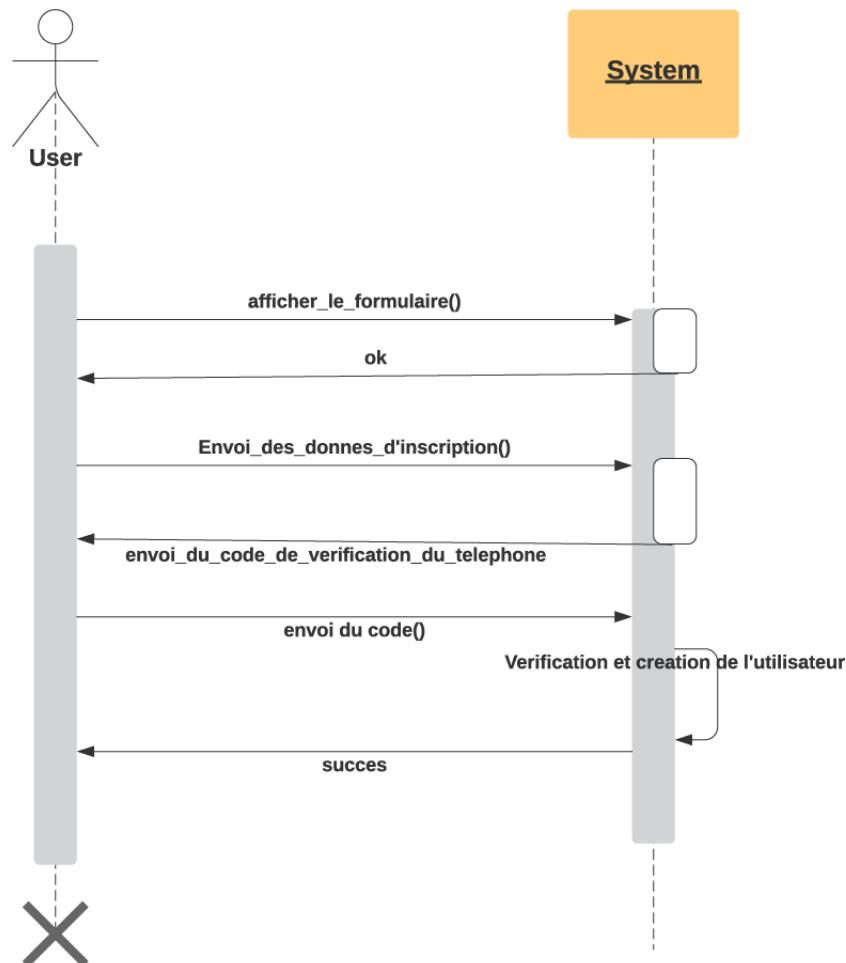


FIGURE 2.3 – diagramme de séquence inscription

Terminer KYC

Titre	Terminer KYC
Acteur(s)	User
Description succincte	après son inscription un utilisateur doit valider son identité pour pouvoir profiter de tous les services
Pré-conditions	inscription
Démarrage	l'utilisateur clique sur le popup qui le notifie de vérifier son compte
Post-conditions	vérification de l'utilisateur

TABLE 2.2 – Description textuelle de terminer le KYC

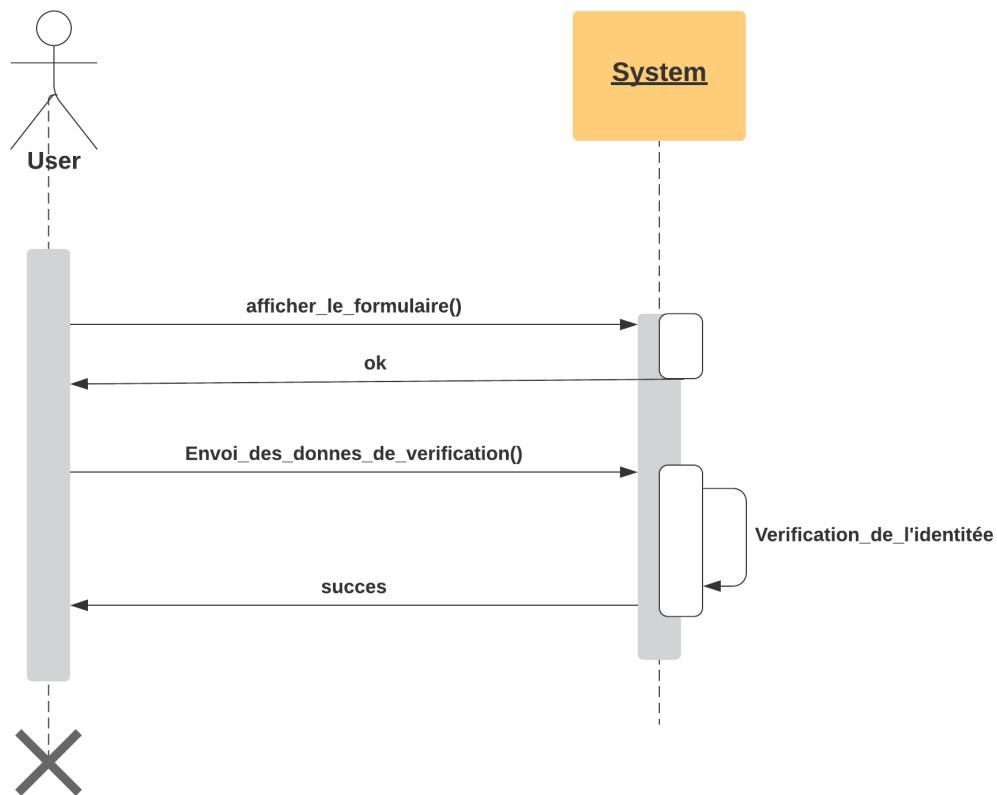


FIGURE 2.4 – diagramme de séquence de validation du processus KYC

Démarrer un trajet instantané

Titre	Démarrer un trajet instantané
Acteur(s)	Driver
Description succincte	un chauffeur doit pouvoir lancer une offre de covoiturage
Pré-conditions	terminer le KYC
Démarrage	le conducteur rempli sa position d'arrivée
Post-conditions	ajout de l'offre

TABLE 2.3 – Description textuelle du démarrer un trajet instantané

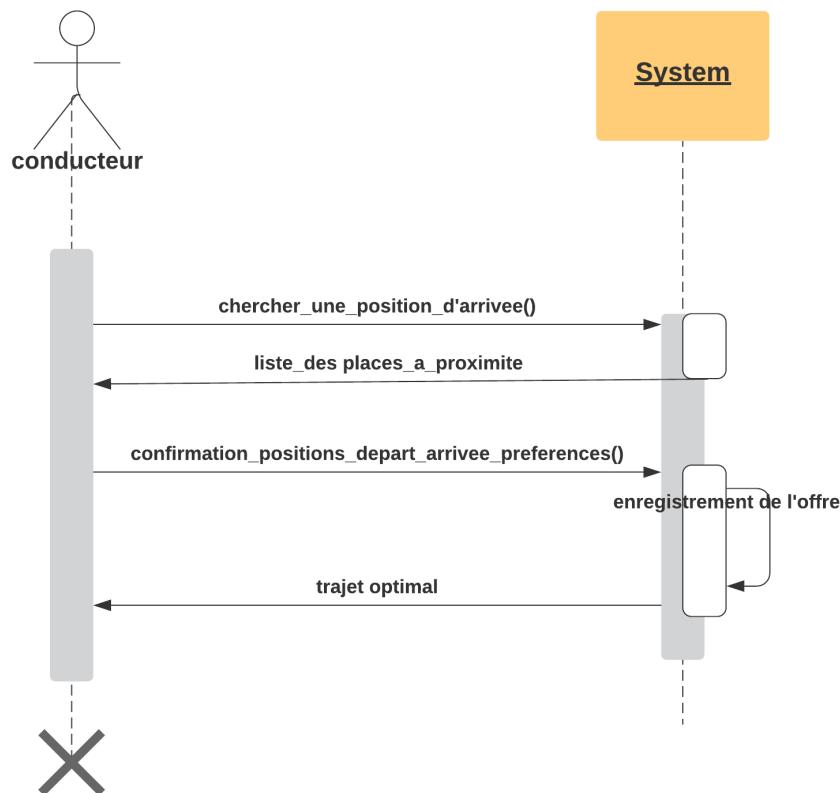


FIGURE 2.5 – diagramme de séquence démarrer un trajet instantané

Demander un covoiturage instantané

Titre	Demander un covoiturage instantané
Acteur(s)	Passenger
Description succincte	un chauffeur doit pouvoir lancer une demande de covoiturage
Pré-conditions	terminer le KYC
Démarrage	le passager rempli sa position d'arrivée
Post-conditions	lancement de l'algorithme de matching

TABLE 2.4 – Description textuelle du demander un covoiturage instantané

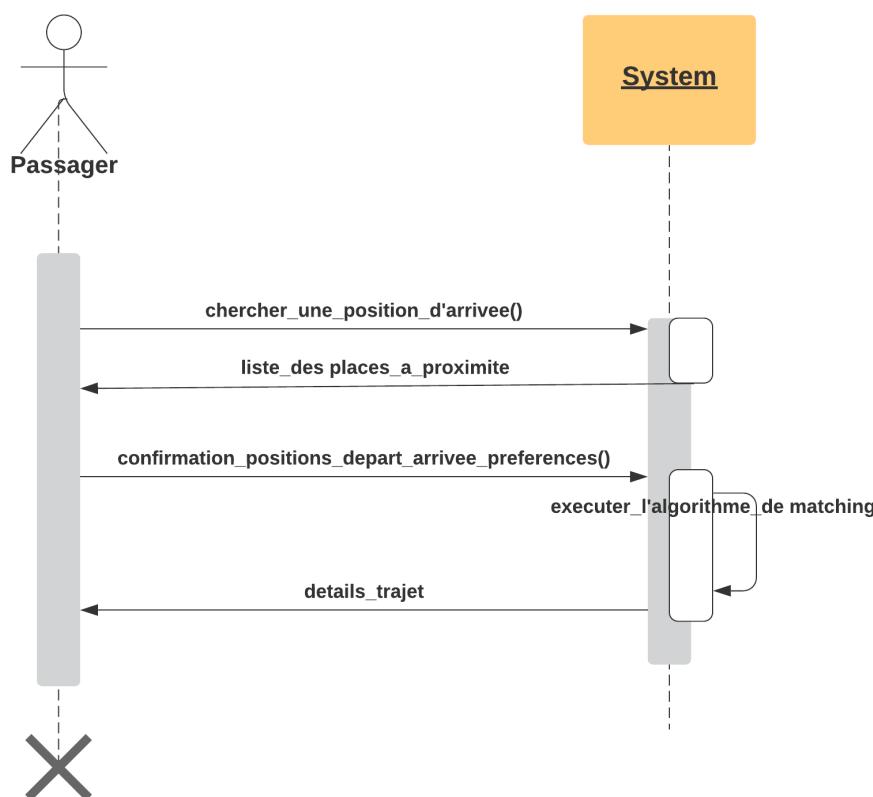


FIGURE 2.6 – diagramme de séquence demander un covoiturage instantané

2.3.3 Conception graphique

À base de l'analyse métier faite, nous avons essayé d'édifier des prototypes d'interfaces graphiques qui nous sert de guide lors de la phase du développement.

les maquettes ont été réalisées en Adobe XD qui permet aux designers de modifier et partager facilement des prototypes interactifs avec collaborateurs et réviseurs et qui est disponible sur l'ensemble des appareils et plates-formes, dont Windows, Mac, iOS et Android.

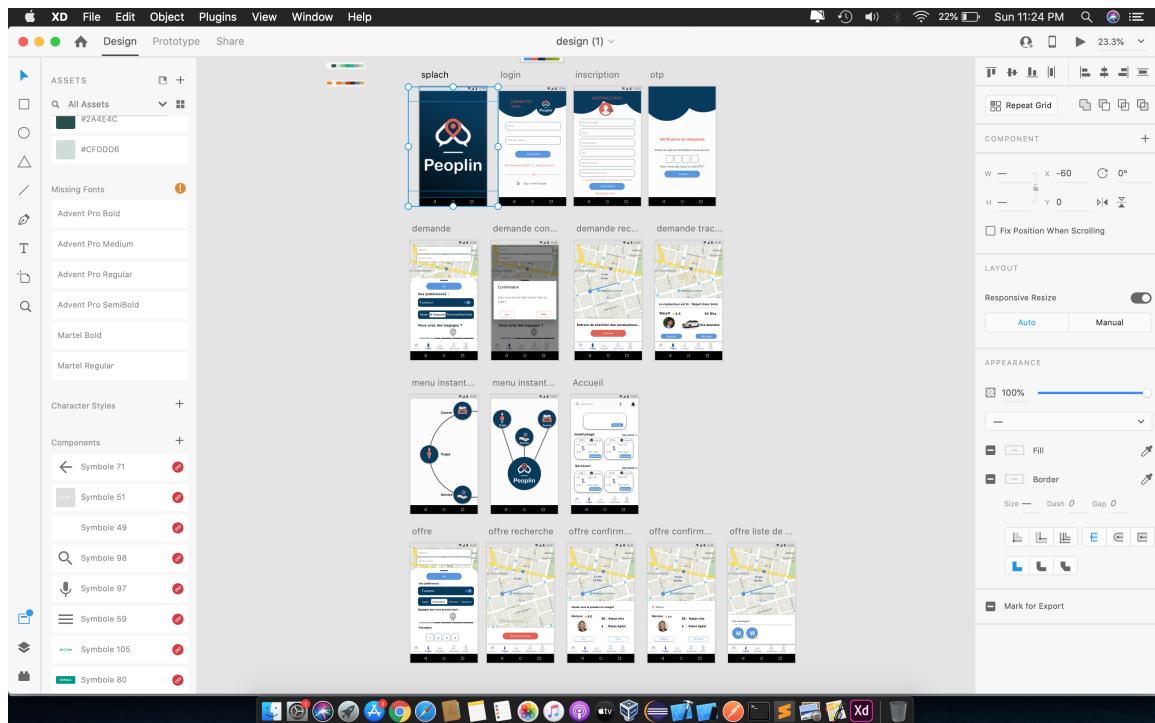


FIGURE 2.7 – prototypes d'interfaces graphiques

2.3.4 Diagramme de classe d'analyse

Pour identifier les éléments du domaine, les relations et interactions entre ces éléments, le diagramme de classe fournit une représentation des entités du système qui vont interagir pour réaliser les cas d'utilisation.

Les classes Offre et Demande présentent sur le diagramme dépendent du covoiturage instantané. Et La classe Preference que je n'ai pas mentionné sur le diagramme représente les préférences soit du passager soit du conducteur dans les deux cas du covoiturage (planifié,

instantané) et qui a comme attributs fumeur, sexe....

Les documents cités dans cette partie d'analyse sont le produit d'une première réflexion et ils ont été améliorés dans la phase du développement, donc ils ne vont pas servir comme moyens de documentation.

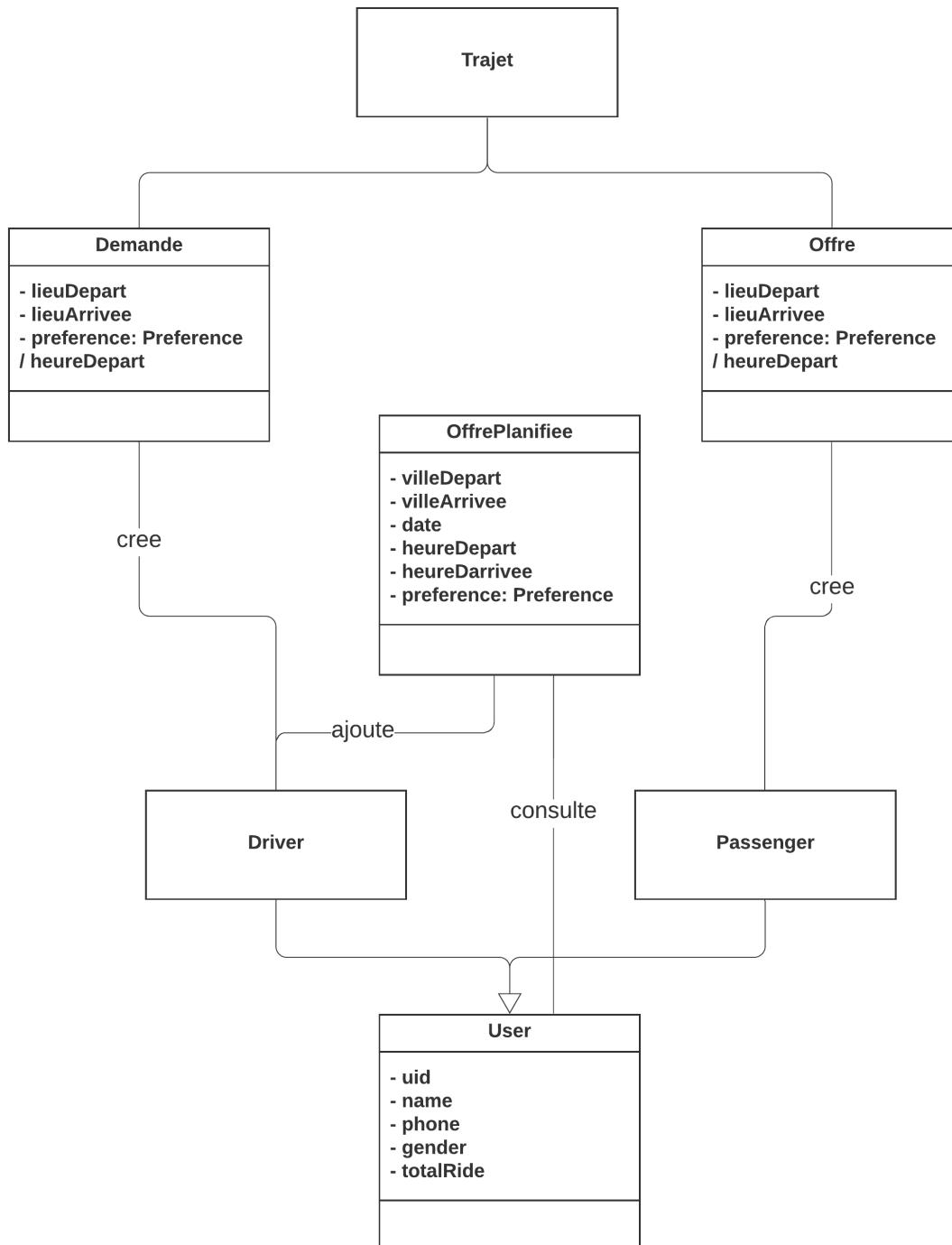


FIGURE 2.8 – diagramme de classe métier simplifié

2.4 Aperçu de l'algorithme de covoiturage

2.4.1 Étapes de l'algorithme

Après la description formelle de nos besoins, on va décrire comment se déroule le processus de matching ou de jumelage qui est le cœur de notre métier.

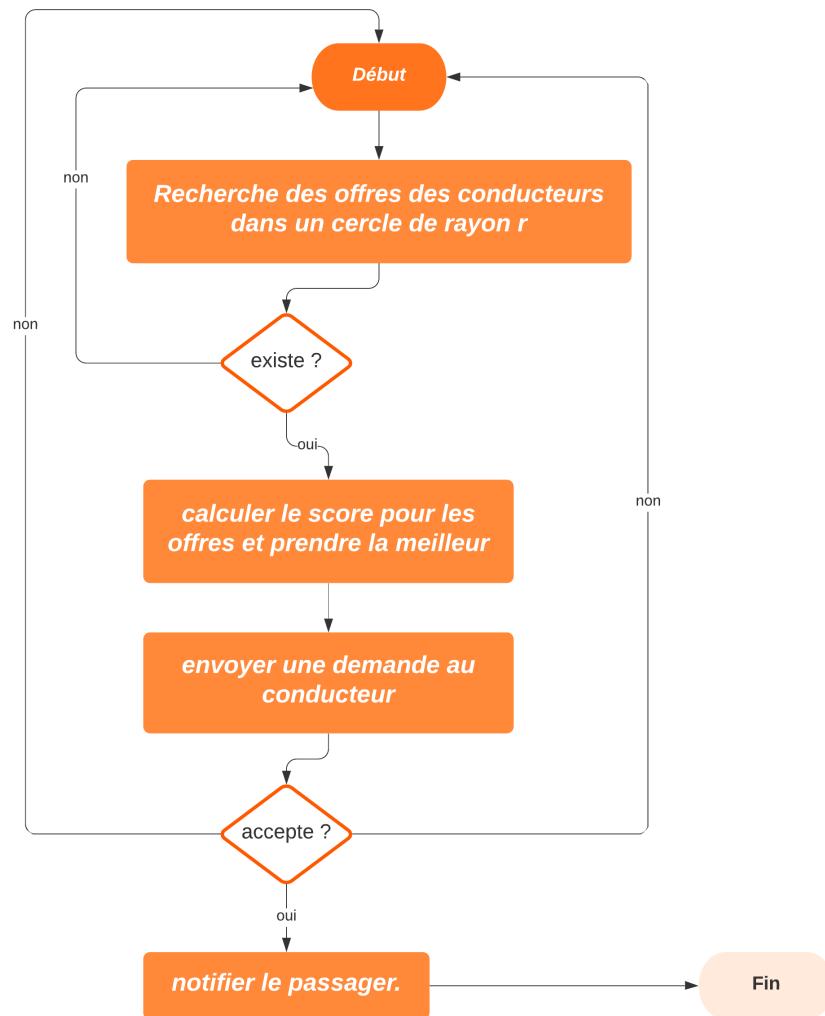


FIGURE 2.9 – logigramme du processus de matching

La figure 2.9 décrit étape par étape les différentes tâches. Notant que ce processus se répète trois fois, si aucune offre n'est trouvée on notifie le passager, dans ce cas il a la possibilité de relancer la recherche manuellement.

2.4.2 Processus du matching

Pour des raisons de confidentialité, on ne va pas rentrer dans les détails d'implémentation et on va se limiter aux cas simples.[10]

Au début du processus, un conducteur qui veut partager sa voiture avec des passagers potentiels enregistre une offre qu'on va noter $O = (O_D, O_A)$, où O_D est le point (position géographique) de départ, et O_A le point d'arrivée. En plus le conducteur nous renseigne sa position en temps réel O_t , qu'elle met à jour chaque fois qu'il s'est déplacé d'une distance d . Le système procède au calcul et au découpage du trajet O en points distancés de d ce qui nous donne que $O = (O_1, O_2, \dots, O_n)$ avec $O_1 = O_D$ et $O_n = O_A$.

Un passager qui demande un covoiturage enregistre sa demande qu'on note $D = (D_D, D_A)$, où D_D est le point (position géographique) de départ, et D_A le point d'arrivée.

Dans un premier temps on va considérer le scenario où on a une seule offre et une seule demande. Notons C_D et C_A l'ensemble des points des deux cercles ayant pour centre respectivement D_D et D_A et pour rayon r . Pour qu'un covoiturage ait lieu il faut :

$$O_t \subset C_D \quad (2.1)$$

ce qui veut dire que la position actuelle du chauffeur est d'une distance moins de r . Deuxièmement, on va vérifier que le point d'arrivée du passager est sur le chemin du conducteur. Ce qui peut se traduire en :

$$(O_1, O_2, \dots, O_n) \cap C_D \neq \emptyset \quad (2.2)$$

Dans le cas de plusieurs offres l'algorithme choisit le chauffeur en calculant un score qui se détermine par la convenance des préférences du chauffeur et le passager (fumeur, bagages, sexe...). Et par le temps de détour des deux offres, par détour on apprend le temps qui va mettre le chauffeur pour prendre en charge le passager.

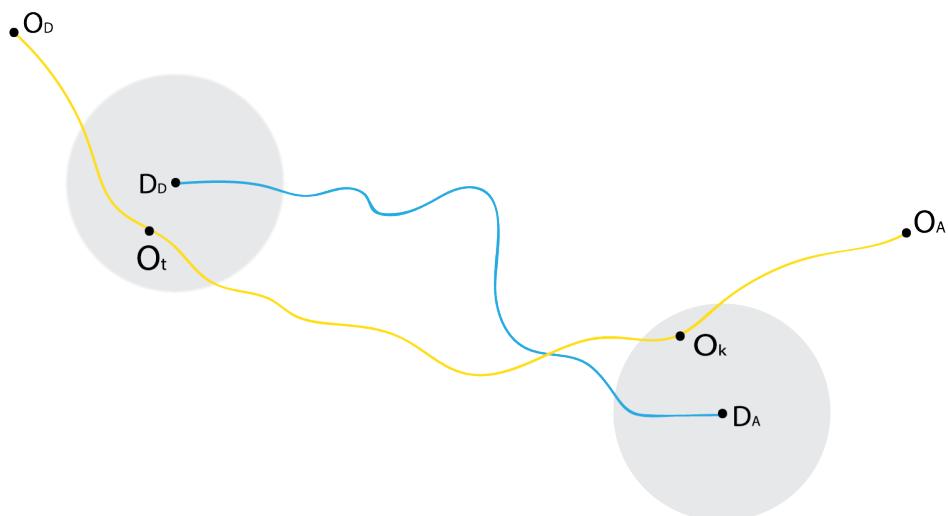


FIGURE 2.10 – processus de matching dans le cas d'une seule offre

Conclusion. Ce deuxième chapitre avait pour finalité l'identification et analyse des besoins, et la présentation de l'algorithme passons maintenant à l'implémentation.

Chapitre 3

Implémentation

Ce chapitre donne quelques éclaircissements quant à l'architecture soit globale soit celle de l'application mobile tout en mettant le doigt sur les problèmes rencontrés, ensuite il liste les différentes technologies utilisées, enfin il justifie nos choix techniques.

3.1 Architecture

3.1.1 Architecture globale

L'architecture globale de mise en œuvre est illustrée par la figure 3.1. Cette architecture est subdivisée en 3 couches et repose sur l'utilisation des services Web, qui ont pour objectif d'assurer une flexibilité d'échange de services entre les différents composants du système.

Les différentes parties vont être décrites comme suit :

1. Couche présentation : c'est notre application cliente dans notre cas mobile, qui tourne sur le système IOS ou Android et assure l'affichage et le dialogue avec l'utilisateur.
2. Coté serveur : assure le traitement métier des données et correspond à la mise en œuvre de l'ensemble des règles de gestion et de la logique applicative, cette partie contient deux principaux services Web : l'un gère les covoiturages instantané et l'autre planifié. Elle est hébergée sur le cloud Digital Ocean :

- Service de covoiturage instantané : il assure les demandes et offres de covoiturage temps réel en utilisant l'algorithme de matching. chaque fois qu'un passager lance une offre il est intercepté par ce service qui lance le processus du matching.
- Service de covoiturage planifié : il est responsable des demandes et offres de covoiturage planifié ou statique. il est sollicité chaque fois une offre ajouté ou à chaque demande de prise en charge d'un passager par un conducteur.

Nos API ou services sont conformes au standard REST (representational state transfer), donc la communication entre ses services est basée sur le protocole Http. Outre ils respectent des contraintes architecturales comme : La communication client–serveur s'effectue sans conservation de l'état de la session de communication sur le serveur entre deux requêtes successives.[8]

3. Base de données : Elle correspond à la partie gérant l'accès aux données de l'application, deux bases de données sont utilisées :

- MySql : les informations sur les utilisateurs, les offres planifiés où autres donnés qui ne changent pas souvent, résident sur cette base.
- Firebase : les données comme les offres instantanée et les positions des conducteurs sont sollicités en lecture en temps réel, donc ils seront hébergés sur cette base.

3.1.2 Architecture client side

MVC

Commençons par définir ce que c'est que MVC. Le Model View Controller permet de diviser son programme en trois parties indépendantes :

- Le modèle : il contient la logique de l'application, il renferme nos entités qui sont des classes ou structures qui interagissent entre eux, et aussi les servies qui peuvent par exemple faire des appels réseau ou accéder à une base de données.
- Le contrôleur : il récupère et formate les informations du modèle pour les passer à la vue.

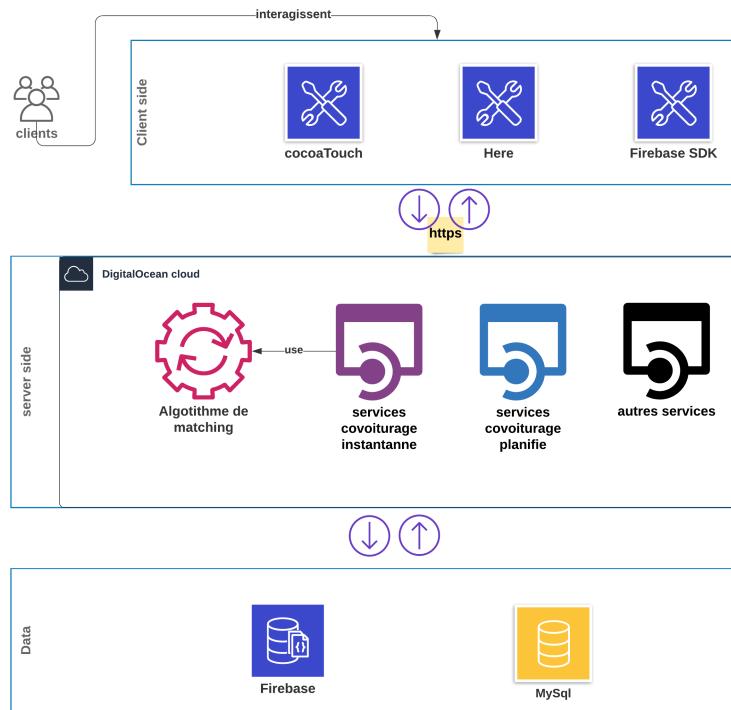


FIGURE 3.1 – architecture de l’application

— La vue : c'est ce que l'utilisateur voit, c'est l'interface de l'application.

Apple a choisi par défaut le très populaire patron de conception architectural MVC pour les applications iPhone[2]. Cela veut dire que la façon dont ils ont conçu le développement d'application iPhone nous encourage à respecter ce design.

MVC avant tout, est un ensemble de règles de communication :

- La vue et le modèle ne peuvent **JAMAIS** communiquer. chacun d'eux ignore l'existence de l'autre.
- Le modèle peut parler indirectement au contrôleur par des notifications ou des fermetures (closures). C'est une communication aveugle, c'est à dire le modèle ne contient pas de référence du contrôleur.
- La vue parle au contrôleur en lui déléguant la gestion des interactions de l'utilisateur avec l'application, le contrôleur dans ce cas s'engage à répondre à sa demande comme montre la figure

La figure 3.3 résume ce qui a été dit dans cette partie.

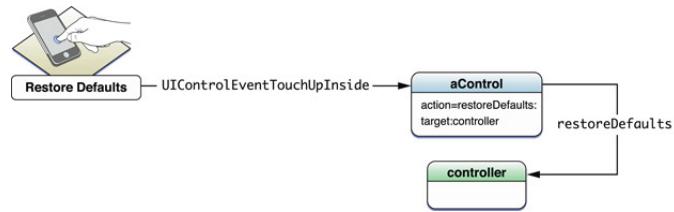


FIGURE 3.2 – le patron de conception target-action

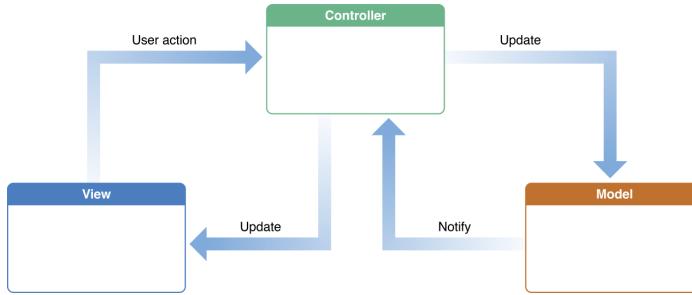


FIGURE 3.3 – patron de conception mvc

Difficultés rencontrées

Au fur et à mesure que notre application croît en taille et en portée, les problèmes complexes de maintenance commencent à surgir. Nous avons rencontré beaucoup de difficultés à tester nos contrôleurs et ce, même en essayant de les alléger et de déléguer beaucoup de fonctionnalités à des classes de services ou modèle. C'est ce qui nous a poussé à adopter une nouvelle structuration du code : le modèle MVVM.[6]

MVVM vise à exporter les traitements de la logique de présentation dans des entités à part entière qui s'appellent les ViewModels, et qui seront placées entre la Vue/Contrôleur (qui désormais constitue une seule entité) et le modèle comme montre la figure 3.4.

- Le modèle : reste le même que celui de MVC.
- La vue/contrôleur : cette couche représente le contexte de l'interface utilisateur et ses interactions. Désormais, nous considérons les vues et leurs contrôleurs MVC comme une seule entité.
- Le View Model : le rôle qui incombe au ViewModel est de récupérer les données à partir du modèle et de les rendre disponibles à la vue sous une meilleure forme. De cette manière, on aura des contrôleurs beaucoup moins chargées qu'avant. On aurait allégé les tests UI puisqu'on a transformé une grande partie des contrôleurs en un modèle de données testable unitairement.



FIGURE 3.4 – le patron de conception MVVM

Puisque je me suis penché plus sur le développement de l'application partie cliente sur ios, je vais me limiter à décrire les outils utilisés dans cette partie par la suite.

3.2 Technologies utilisées

Langages

- **Objective-C** : C'est une extension du C ANSI, comme le C++, mais qui se distingue de ce dernier par sa distribution dynamique des messages, son typage faible ou fort, son typage dynamique et son chargement dynamique. il est principalement utilisé dans les systèmes d'exploitation d'Apple.
- **Swift** : un langage de programmation objet compilé, multi-paradigmes, ayant pour objectif d'être simple, hautes performances et sûr. Il est développé en open source.

Frameworks

- **Cocoa Touch** : il fournit l'infrastructure requise pour les applications iOS ou tvOS. Il fournit l'architecture de fenêtre et de vue pour implémenter l'interface, l'infrastructure de gestion des événements pour fournir Multi-Touch et d'autres types d'entrée à l'application. il regroupe deux autres frameworks qui sont Foundation c'est la base comme son nom l'indique, et UIKit qui permet de créer des interfaces graphiques.
- **Core Data** : c'est un framework qui permet de faire persister les données en local (sur l'iphone), il fournit l'accès à une base de donnée standard (souvent SQLite) et une API orientée objet c'est à dire un ORM (Object Relational Mapping).

- **Here** : il permet aux applications clientes d'utiliser les données de cartographie comme colonne vertébrale pour leurs applications mobiles, et aussi d'accéder à des informations telles que les vitesses moyennes sur route, l'accroissement du trafic et les charges maximales pouvant être transportées sur une route particulière
- **Firebase** : un BaaS (Back-as-a-Service) qui permettra de stocker et de synchroniser les données sur tous vos clients en temps réel. Chaque fois que les données changent, Firebase met à jour les applications sur chaque appareil (mobile ou Web). Si votre application Firebase fonctionne hors ligne, ses données seront synchronisées une fois la connectivité rétablie.
le SDK pour (Software Development Kit) de Firebase regroupe tous les API et il est facile à intégrer dans l'application.

Outils

- **Xcode** : un environnement de développement (IDE) pour macOS, Fournit toute une suite logicielle (graphiques, audio, etc.) pour développeurs et programmeurs. Il permet de créer des logiciels pour iOS, watchOS et tvOS.
- **CocoaPods** : un gestionnaire de dépendances au niveau de l'application pour Objective-C, Swift et toutes les autres langues qui s'exécutent sur l'environnement d'exécution Objective-C, comme RubyMotion, et qui fournit un format standard pour la gestion des bibliothèques externes. Il permet d'automatiser l'intégration des autres frameworks dans un projet.
- **Postman** : un outil qui permet d'interroger et tester les webservices et API, il propose de nombreuses fonctionnalités, une prise en main rapide et une interface graphique agréable.

3.3 Justification des choix techniques

3.3.1 partie cliente

Applications natives

- Notre application a besoin d'accéder rapidement à toutes les fonctionnalités du téléphone, du gps en passant par la caméra et le micro.
- Les notifications push, uniquement disponibles sur les apps natives occupent une place centrale dans notre app. Ils permettent d'alerter les passagers sur l'état de leurs offres ainsi que la position du conducteur en cas de prise en charge. Et au conducteur de recevoir des demandes.
- le budget dédié au développement couvre les charges de ces dernières.

Here

Dans ce cas le choix est basé sur le prix que propose différents fournisseurs de service de cartographie.

Google a augmenté ses prix de 1400%, et a limité sa version gratuite. C'est pourquoi ICI (Here) est beaucoup plus attrayant que Google, car nous pouvons utiliser le modèle freemium et utiliser 250 000 appels API par jour. Versus 28.000 de Google.

La figure ci-dessous compare les deux fournisseurs en termes de prix.[4]

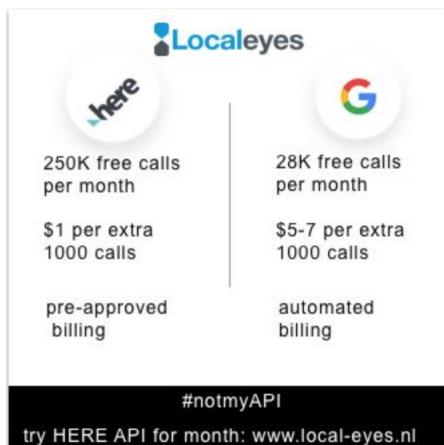


FIGURE 3.5 – HERE Maps API vs Google Maps API

3.3.2 Partie serveur

API Rest

Comme il a été dit dans la partie spécification des besoins notre application devrait être portable et extensible. Les services web REST permet au mieux de répondre à ses exigences et sont plus faciles à implémenter, comparés aux méthodes du type RPC Remote Procedure Call comme SOAP.

Digital Ocean

Puisque notre utilisation du cloud se limite à louer une capacité de calcul et donc pas aux services. On a besoin d'un fournisseur cloud qui gère le matériel serveur, les couches de virtualisation, le stockage, Les réseaux.

Digital Ocean présente une solution qui répond bien à notre besoin d'héberger notre application à moindre coût.[1]

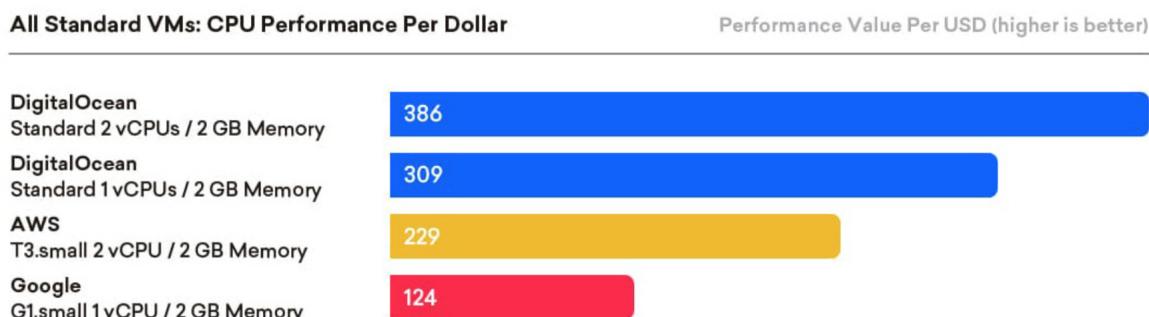


FIGURE 3.6 – Performances CPU par dollar de toutes les machines virtuelles testées

3.3.3 Choix du système de gestion de données

Mysql

- Le projet MySQL a rendu son code source disponible sous les termes de la GNU General Public License, donc il représente une solution à moindre coût, l'autre avantage est qu'il y a une grande communauté derrière, par conséquent, nous pouvons

facilement apprendre et dépanner MySQL à partir de différentes sources comme des blogs, des livres blancs et des livres.

- Nous n'avons pas de données sur le nombre d'utilisateur à venir, donc la mise à l'échelle et la réPLICATION sont des fonctions primordiales et notre SGBD les prennent en charge.
- Les transactions sont simples, les données sur le covoiturage planifié sont le plus souvent demandées en lecture, donc les performances de mysql sont largement suffisantes.

Firestore

- Dans le cas des données sur le covoiturage dynamique (offres, demandes, position des conducteurs...), l'écriture et la lecture en temps réel est le premier facteur à prendre en compte. la raison derrière le choix est que Firebase est essentiellement un service cloud fourni par Google, il fonctionne donc mieux que les autres serveurs et fournisseurs de services backend. (une étude détaillée nous manquait)
- Vu la contrainte temporelle : 4 mois pour développer l'application. Firebase économe beaucoup de temps puisque il offre un ensemble de services prêts à l'emploi, cela veut dire qu'il n'est pas seulement un système de gestion de données mais un logiciel en tant que service SaaS.
- En mode développement la plateforme est gratuite.
- Firebase c'est tout un écosystème donc nous pouvons intégrer gratuitement des services comme Google Analytics qui nous aide à comprendre comment les gens utilisent notre application. L'outil capture automatiquement un certain nombre d'événements et de propriétés utilisateur et permet également de définir nos propres événements personnalisés pour mesurer les éléments qui importent à l'entreprise.

3.4 Quelques interfaces utilisateur

Cette section présente quelques interfaces utilisateur.

(a) L'accueil

9:58 10:02

PEOPLIN

Chercher ...

Regardez aussi cela :

Wassif Meskine
4.5 ★ 15 voitures

Rabat → Marrakech
12:00 → 15:00
29 mars

Arrets : Casablanca, Settat, Benguerir. [Voir plus](#)

4 places sont disponibles

80 M.A.D

reserver

(b) Ajouter offre planifiée

Ajouter une offre

General

Adresse de depart heure depart

Adresse d'arrivee heure d'arrivee

25 MON MAY 26 TUE MAY 27 WED MAY 28 THU MAY 29 FRI MAY Prix

Preferences

Fumeur

Sexe All Femme Homme

Bagages 1 2 3

Nombres de places 1 2 3 4

Offres intermediaires

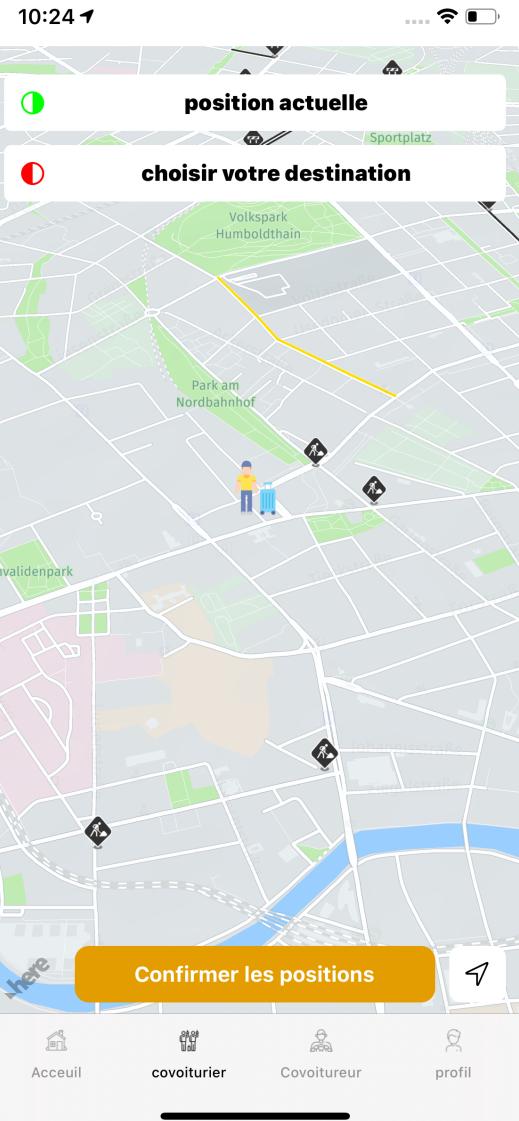
Ville de depart Ville d'arrivee

Heure depart heure d'arrivee Prix

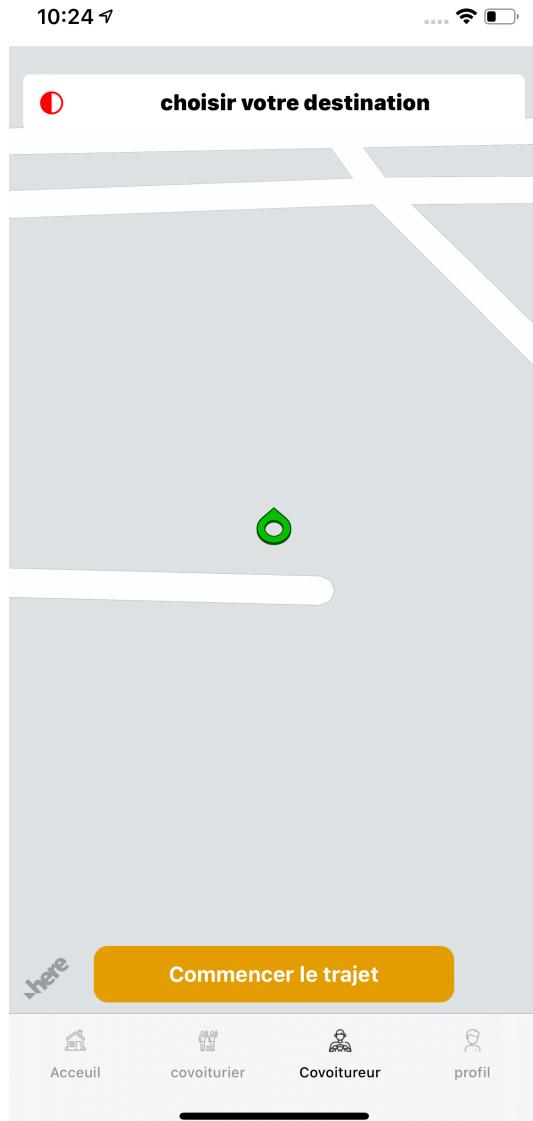
Ajouter l'offre

Acceuil covoiteur Covoitureur profil

FIGURE 3.7 – Les offres planifiés



(a) Conducteur



(b) Passager

FIGURE 3.8 – Les offres instantanés

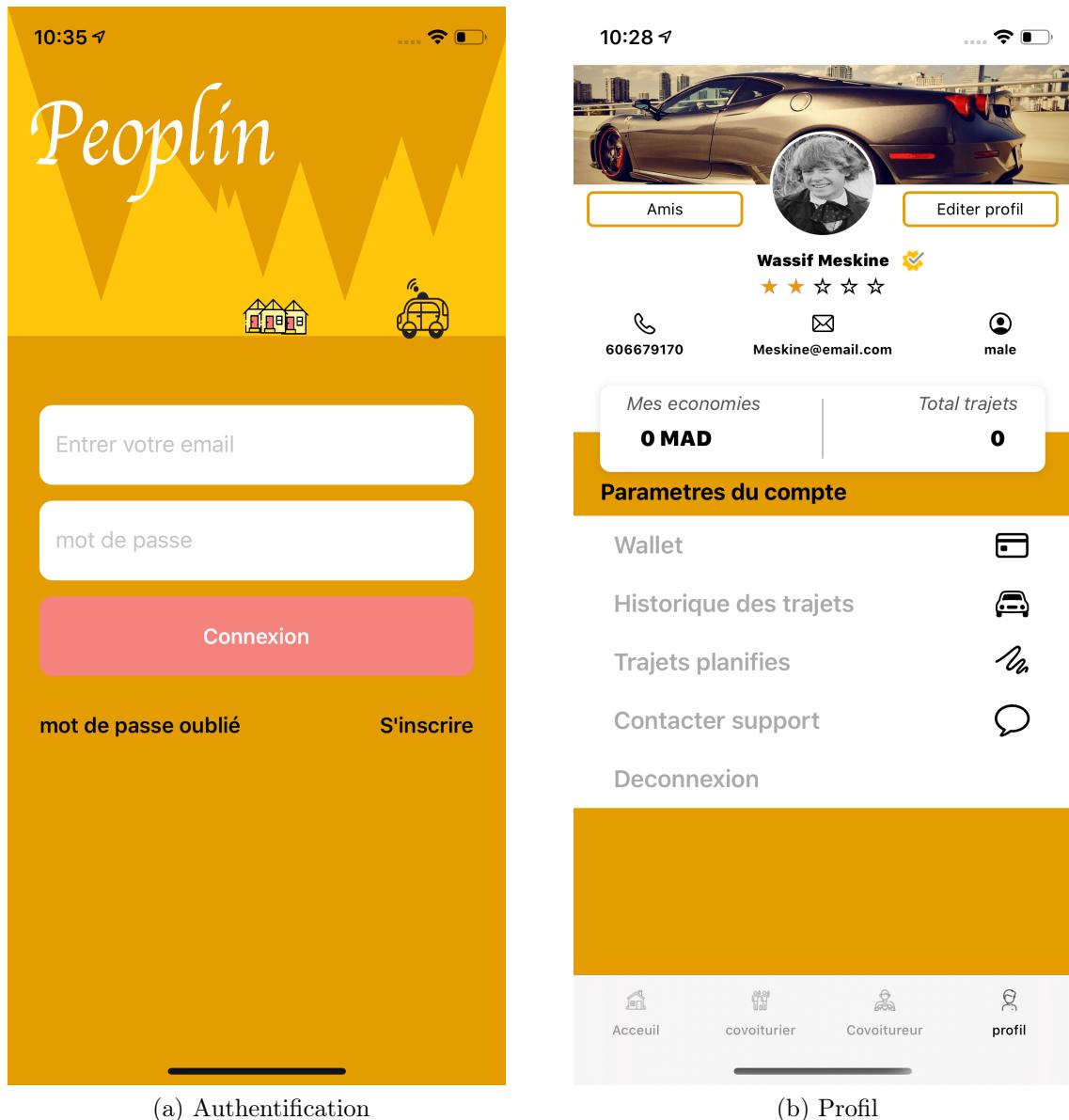


FIGURE 3.9 – Utilisateur

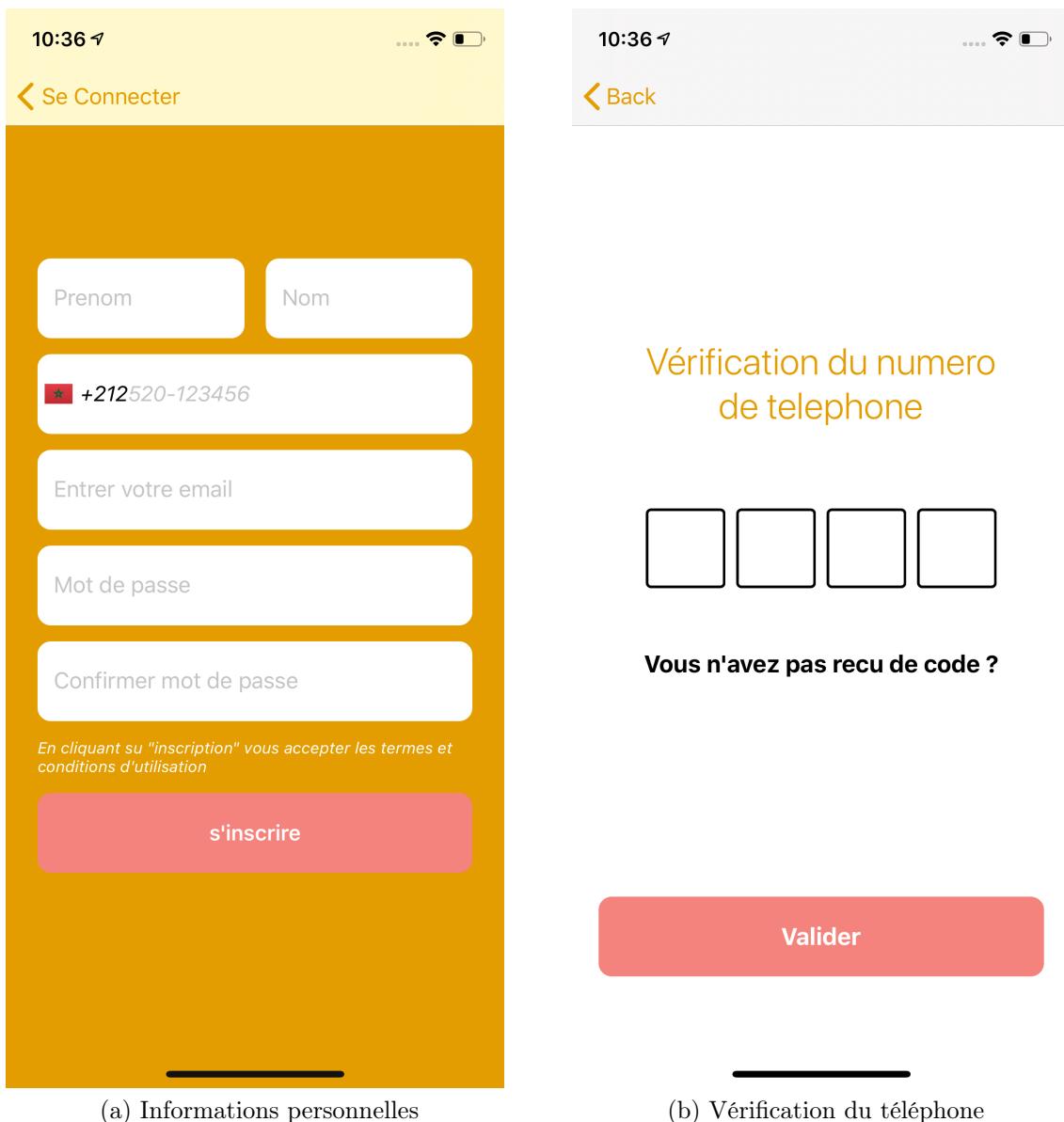


FIGURE 3.10 – L’inscription

Conclusion. Ce troisième chapitre avait pour finalité de donner une vue d’ensemble sur l’architecture, les technologies et le choix technique.

Chapitre 4

Tests et évaluation

4.1 Code testable

Avant même d'écrire des tests, il faut préparer son code à être testable. Synonyme de respecter un ensemble de principes dans le développement.

Cette section présente quelques principes qui nous ont aidé à écrire du code facilement testable, mais pas seulement testable, mais plus facile à gérer et plus flexible en raison de sa meilleure modularité.

Pour parvenir à écrire un code plus robuste, nous avons été amenés à suivre les cinq principes SOLID [3]introduits par Robert C. Martin.[9]

- **Responsabilité unique** (Single responsibility principle) une classe, une fonction ou une méthode doit avoir une et une seule responsabilité.
- **Ouvert/fermé** (Open/closed principle) une entité applicative (classe, fonction, module ...) doit être ouverte à l'extension, mais fermée à la modification.
- **Substitution de Liskov** (Liskov substitution principle) une instance de type T doit pouvoir être remplacée par une instance de type G, tel que G sous-type de T, sans que cela ne modifie la cohérence du programme
- **Ségrégation des interfaces** (Interface segregation principle) préférer plusieurs interfaces spécifiques pour chaque client plutôt qu'une seule interface générale
- **Inversion des dépendances** (Dependency inversion principle) il faut dépendre des abstractions, pas des implémentations

En plus, comme on a dit dans le chapitre précédent, nous avons adopté dans certaines parties de l'application une architecture MVVM au lieu de MVC pour que nos tests couvrent une grande partie de notre code.

4.2 Intérêts des tests

Dans cette section nous allons répondre à la question, pourquoi écrire des tests ?

Les tests ont pour but de :

- Montrer que notre code fonctionne : si un changement casse quelque chose, un ou plusieurs des tests vont échouer.
- prévenir les bugs et les régressions : tout défaut introduit dans application à l'occasion de l'ajout de nouvelles fonctionnalités, modification de fonctionnalités existantes ou modification d'un composant externe au logiciel lui-même sera signalé par des tests qui vont échouer.
- penser les classes : comme on a vu dans la première section, il faut dépendre des abstractions, pas des implémentations. Donc écrire des tests incite à créer des classes dont les responsabilités sont uniques et indépendantes.
- offrir une documentation du code : chaque test va couvrir un cas d'usage d'une des classes. Donc si tout le code est testé, les tests sont une vraie documentation fonctionnelle du programme

4.3 Outils de tests

Afin de parvenir à dénicher les bugs et à écrire un code qui gagnera en robustesse nous avons utilisées un ensemble d'outils cités ci-dessous :

- **LLDB** : un débogueur haute performance de nouvelle génération. Il est construit comme un ensemble de composants réutilisables qui exploitent fortement les bibliothèques existantes dans le plus grand projet LLVM, comme l'analyseur d'expression Clang et le désassembleur LLVM.
- **XCTest** : framework qui permet de créer et exécuter des tests unitaires, des tests

de performances et des tests d’interface utilisateur pour un projet Xcode.

- **Couverture du code** (Code coverage) : indique quel pourcentage du code testé a été exécuté lors du lancement des tests unitaires correspondants. Xcode 7 a introduit une fonctionnalité de couverture du code.
- **SwiftLint** : un outil créé et maintenu par Realm pour améliorer le respect des conventions d’écritures en Swift, c’est à dire il fait une analyse statique du code.

4.4 Les types de test

Dans notre application nous avons eu recours à deux types de tests :

- Tests manuels : conduit par les membres de l’équipe et sont destinés pour tester les interfaces utilisateurs, dans notre cas ils remplacent les tests fonctionnels.
- Tests unitaires : destinés à tester chaque bout de code et sont automatisés par XCTest.

4.5 Évaluation

Pour évaluer les performances de notre application, nous avons utilisé Xcode Instruments qui regroupe une collection d’outils pour profiler le comportement d’exécution de l’application. Par exemple, le Time Profiler est un excellent outil pour mesurer les performances et analyser le temps d’exécution des différents threads ou fil d’exécution.

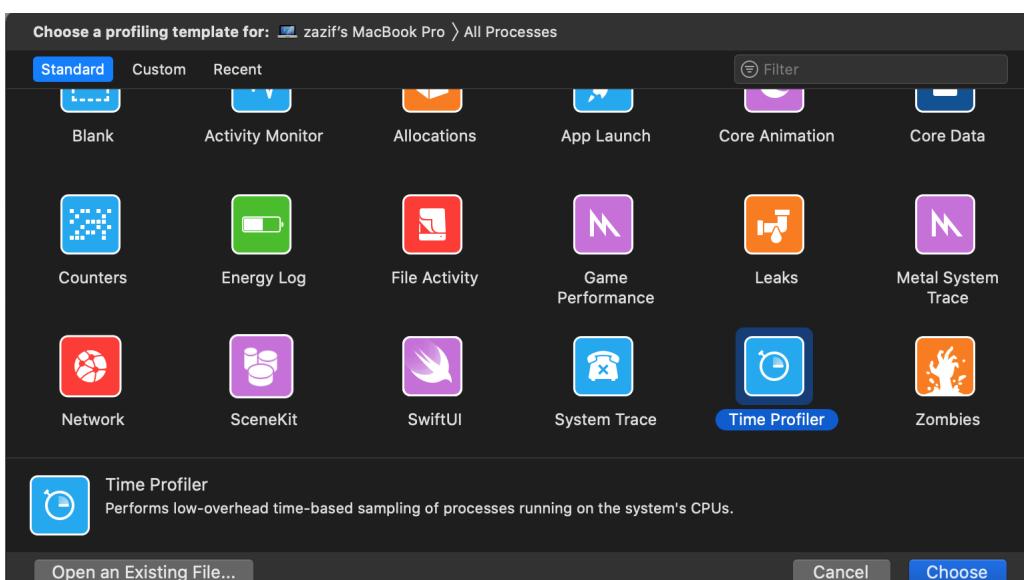


FIGURE 4.1 – Xcode Instruments

Chapitre 5

Résumé et réflexions

Bibliographie

- [1] Cloud Performance Analysis Report. <https://www.digitalocean.com/resources/cloud-performance-report/>.
- [2] Model-View-Controller. <https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>.
- [3] Principes SOLID. [https://fr.wikipedia.org/wiki/SOLID_\(informatique\)](https://fr.wikipedia.org/wiki/SOLID_(informatique)).
- [4] comparaison here et google maps. <https://local-eyes.nl/page/comparing-google-maps-and-here-maps-pricing>, 2020.
- [5] ARNAUD LISSAJOUX. pilier scrum. <https://openclassrooms.com/fr/courses/4511316-perfectionnez-votre-gestion-de-projet-agile>, 2017.
- [6] AYMEN ABBASSIA. implementer mvvm sur ios. <https://soat.developpez.com/tutoriels/ios/apprendre-implementer-modele-view-viewmodel/>, 2015.
- [7] CHEIKH, S. B. *Optimisation avancée au service du covoiturage dynamique*. PhD thesis, 2016.
- [8] FIELDING, R. T., AND TAYLOR, R. N. *Architectural styles and the design of network-based software architectures*, vol. 7. University of California, Irvine Irvine, 2000.
- [9] MARTIN, R. C. *Agile software development : principles, patterns, and practices*. Prentice Hall, 2002.
- [10] SCHREIECK, M., SAFETLI, H., SIDDIQUI, S. A., PFLÜGLER, C., WIESCHE, M., AND KRCMAR, H. A matching algorithm for dynamic ridesharing. *Transportation Research Procedia* 19 (2016), 272–285.