



République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Tunis El Manar
École Nationale d'Ingénieurs de Tunis



Département Technologies de l'Information et de la Communication

Projet de Fin d'Année II

Réalisation d'une application de
covoiturage avec l'architecture
microservices (côté conducteur).

Réalisé par :

Marouane BOUSLAMA

Ahmed HAZGUI

Classe : 2^{ème} Année Informatique

Encadré par :

M. Mohamed Ramzi HADDAD

Année Universitaire 2022/2023

Remerciement

Un immense merci à tous ceux qui nous ont aidé dans ce projet, et nous tenons tout particulièrement à exprimer notre plus grand respect et gratitude à notre encadrant, M. Mohamed Ramzi Haddad, pour son enthousiasme et sa disponibilité.

Nous tenons également à remercier tous les professeurs de l'Ecole Nationale des Ingénieurs de Tunis pour la grande qualité de la formation que nous avons reçu et tous les membres du jury qui ont accepté de juger ce travail.

Table des matières

Table des figures	4
Liste d'acronymes	6
Introduction générale	7
1 Contexte général et spécification des besoins	8
Introduction	8
1.1 Contexte du projet	8
1.2 Objectif du projet	8
1.3 État de l'art : Le covoiturage	8
1.3.1 Présentation du concept de covoiturage	9
1.3.2 Technologie temps réel pour une plateforme de covoiturage	9
1.4 Méthodologie de travail	9
1.5 Spécification des besoins	9
1.5.1 Besoins fonctionnels	10
1.5.2 Besoins non fonctionnels	10
1.5.3 Diagramme de cas d'utilisation globale	10
1.6 Raffinement des cas d'utilisation	11
1.6.1 Raffinement de cas d'utilisation "Gérer les offres"	11
1.6.2 Raffinement de cas d'utilisation "Tarifier les offres"	12
1.6.3 Raffinement de cas d'utilisation "Gérer les réservations"	12
1.6.4 Raffinement de cas d'utilisation "Notifier les conducteurs et les passagers"	13
Conclusion	14
2 Architecture et Conception	15
Introduction	15
2.1 La conception globale de l'application	15
2.1.1 Architecture globale adaptée à l'application	15
2.1.2 Patron d'architecture adopté pour les microservices	21
2.2 La conception détaillée des microservices	24
2.2.1 La conception du microservice "Gestion des offres de covoiturage"	24
2.2.2 La conception du microservice "Tarification"	27
2.2.3 La conception du microservice "Reservations"	30
2.2.4 La conception du microservice "Notifications"	34
conclusion	37
3 Réalisation de l'application	38
Introduction	38
3.1 Technologies utilisées	38
3.1.1 Environnement de développement	38

3.1.2	Librairies et frameworks utilisés	39
3.2	Réalisation des microservices	41
3.2.1	Authentification avec Keycloak	41
3.2.2	Réalisation du microservice "Gestion des offres de covoiturage"	43
3.2.3	Réalisation du microservice "Tarification"	45
3.2.4	Réalisation du microservice "Reservations"	45
3.2.5	Réalisation du microservice "Notifications"	47
conclusion	48
Conclusion générale		49
Références		50

Table des figures

1.1	Diagramme de cas d'utilisation globale.	11
1.2	Cas d'utilisation "Gérer les offres.	12
1.3	Cas d'utilisation "Tarifier les offres".	12
1.4	Cas d'utilisation "Gérer les réservations.	13
1.5	Cas d'utilisation "Notifier les conducteurs et les passagers"	13
2.1	Représentation des différents éléments d'event storming[18].	17
2.2	Contexte de gestion des offres.	18
2.3	Contexte de tarification.	18
2.4	Contexte de gestion des réservations.	19
2.5	Contexte de notifications.	20
2.6	La carte de contextes des différents contextes bornés.	21
2.7	Diagramme de classe générique	22
2.8	Le diagramme d'architecture en oignon. [12]	23
2.9	Diagramme de séquence générique d'un microservice	23
2.10	Domaine du microservice "Gestion des offres de covoiturage"	24
2.11	Les services du microservice "Gestion des offres de covoiturage"	25
2.12	L'infrastructure du microservice "Gestion des offres de covoiturage"	25
2.13	Diagramme de classe du microservice "Gestion des offres de covoiturage"	26
2.14	Domaine du microservice "Tarification"	27
2.15	Les services du microservice "Tarification"	28
2.16	L'infrastructure du microservice "Tarification"	28
2.17	Diagramme de classe du microservice "Tarification"	29
2.18	Domaine du microservice "Reservations"	30
2.19	Les services du microservice "Reservations"	31
2.20	L'infrastructure du microservice "Reservations"	32
2.21	Diagramme de classe du microservice "Reservations"	33
2.22	Domaine du microservice "Notifications"	34
2.23	Les services du microservice "Notifications"	35
2.24	L'infrastructure du microservice "Notifications"	35
2.25	Diagramme de classe du microservice "Notifications"	36
3.1	Interface de création d'un utilisateur	41
3.2	Interface d'affectation des rôles	42
3.3	Interface d'authentification	42
3.4	Page d'accueil de l'application	43
3.5	Requête de création d'une offre	43
3.6	Interface du microservice "Gestion des offres de covoiturage"	44
3.7	Requête de publication ou d'annulation d'une offre	44
3.8	Requête de changement du nombre de places disponibles d'une offre	44
3.9	Offre tarifée par le microservice "Tarification"	45

3.10	Requête pour fixer le prix d'une offre	45
3.11	Voyage récupéré	46
3.12	Requête d'acceptation d'une application	47
3.13	Notification récupérée	47
3.14	Requête pour notifier le conducteur de l'annulation d'une application	48

Liste d'acronymes

- **DDD** : Domain-driven design (Conception pilotée par le domaine).
- **API** : Application Programming Interface (Interface de programmation d'application).
- **DTO** : Data Transfer Object (Objet de transfert de données).
- **UI** : User Interface (interface utilisateur).
- **PL** : Published Language (Langage publié).
- **U** : Upstream (Contexte en amont).
- **D** : Downstream (Contexte en aval).
- **Shared Kernel** : Noyau partagé.

Introduction générale

Le covoiturage est devenu de plus en plus populaire ces dernières années en tant qu'alternative plus rentable et plus écologique aux modes de transport traditionnels. En partageant des trajets, les gens peuvent réduire leur empreinte carbone individuelle, tout en économisant de l'argent sur l'essence et les dépenses liées au véhicule. Le covoiturage peut également réduire les embouteillages et améliorer l'efficacité globale des transports, car un plus grand nombre de personnes peuvent se rendre à leur destination avec moins de véhicules. L'essor du covoiturage a également conduit au développement de diverses plateformes technologiques et applications mobiles qui facilitent la mise en relation et la coordination des trajets.

Le projet en question vise à développer un système logiciel robuste et évolutif capable de gérer efficacement des opérations commerciales complexes. Le système est conçu pour être hautement modulaire, chaque module étant implémenté comme un microservice séparé, afin de permettre une plus grande flexibilité et une plus grande indépendance pendant le développement et le déploiement. Le système est également conçu pour être très adaptable, avec une architecture flexible qui peut s'adapter à l'évolution des besoins et prendre en charge les améliorations futures. Tout au long du projet, l'équipe a utilisé les meilleures pratiques en matière d'ingénierie logicielle, notamment la conception pilotée par le domaine, l'analyse des événements et l'architecture des microservices, afin de s'assurer que le système est hautement maintenable, évolutif et efficace. L'objectif final du projet est de fournir un système logiciel de haute qualité qui réponde aux besoins de ses clients, tout en maintenant un niveau élevé de fiabilité, de sécurité et de performance.

Le rapport est structuré en 3 chapitres le premier définit le cadre général et la spécification des besoins du projet, le deuxième présente l'architecture et la conception de l'application et le troisième montre sa réalisation.

Chapitre 1

Contexte général et spécification des besoins

Introduction

Le chapitre suivant décrit le contexte et les objectifs du projet de conception et de développement d'une plateforme de covoiturage du point de vue des conducteurs. Il présente également l'état actuel des connaissances dans le domaine du covoiturage, les méthodes de travail utilisées pour réaliser ce projet et la spécification des besoins. Ce chapitre donne un aperçu des questions abordées et des différentes approches envisagées pour répondre aux besoins des utilisateurs.

1.1 Contexte du projet

Ce projet intitulé « Conception et développement d'une plateforme de covoiturage (côté conducteur) » est conçu dans le cadre du Projet de Fin d'Année (PFA) de la deuxième année de cycle ingénieur.

1.2 Objectif du projet

L'objectif de ce projet est de concevoir et développer une plateforme de covoiturage pour les conducteurs afin de faciliter leurs déplacements quotidiens en leur offrant une solution économique, écologique et conviviale.

La plateforme permettra aux conducteurs de proposer des trajets à d'autres personnes qui souhaitent se déplacer dans la même direction, en échange d'une participation aux frais de transport. Elle offrira également des fonctionnalités telles que la gestion de l'offre proposée et des réservations ainsi que la communication entre les conducteurs et les passagers. L'objectif final est donc de promouvoir une utilisation plus efficace des ressources de transport existantes.

1.3 État de l'art : Le covoiturage

Cette partie du rapport présente l'état de l'art du covoiturage basé sur les recherches et projets existants dans le domaine. L'objectif est de mettre en évidence les avancées récentes et les tendances actuelles en matière de conception de plateformes de covoiturage, ainsi que les problèmes et les défis qui restent à résoudre.

1.3.1 Présentation du concept de covoiturage

Le covoiturage est un mode de transport économique et écologique qui permet de partager le coût d'un trajet avec d'autres passagers, tout en réduisant les embouteillages et la pollution. Il favorise également l'interaction sociale et la communication entre les passagers.

Cependant, il est important de rappeler que le covoiturage ne doit pas être considéré comme une activité lucrative, mais plutôt comme une contribution économique au coût du transport. En termes de définition, nous parlons de demande lorsqu'un passager demande à être transporté d'un endroit à un autre, tandis qu'un covoituré est un passager du covoiturage qui demande à être transporté. Une offre est faite lorsqu'un conducteur souhaite partager sa voiture avec d'autres pour un trajet spécifique, tandis qu'un covoitureur est le conducteur qui propose un covoiturage. Enfin, pour qu'il y ait covoiturage, certaines contraintes d'appariement doivent être respectées, en particulier la correspondance spatiale et temporelle entre l'offre et la demande, et la capacité respectée en termes de nombre de sièges disponibles dans la voiture.

1.3.2 Technologie temps réel pour une plateforme de covoiturage

La mise en œuvre d'une plateforme de mobilité partagée efficace nécessite l'utilisation d'une technologie en temps réel. Les informations telles que les demandes de trajet, les offres de voitures partagées, les connexions de voitures partagées et les conducteurs de voitures partagées doivent être traitées et mises à jour en temps réel pour garantir une expérience fluide et satisfaisante pour l'utilisateur. La technologie en temps réel permet également d'optimiser la gestion des offres et des demandes en temps réel pour répondre aux besoins des utilisateurs. L'utilisation de la technologie en temps réel est donc essentielle au succès d'une plateforme de covoiturage efficace.

1.4 Méthodologie de travail

La méthodologie adoptée pour le projet est en spirale avec une approche itérative du développement de logiciels qui consiste à travailler sur les différents éléments d'un projet en quatre étapes : spécification, conception, mise en œuvre et test. Cette approche repose sur l'idée que chaque fonctionnalité doit être considérée comme un petit projet indépendant qui doit passer par ces quatre phases avant d'être considéré comme achevé. La spécification définit les besoins et les exigences de la fonctionnalité, la conception détermine la manière dont la fonctionnalité sera mise en œuvre, la mise en œuvre est le codage de la fonctionnalité et les tests garantissent que la fonctionnalité fonctionne correctement et répond aux exigences spécifiées.

1.5 Spécification des besoins

La phase de spécification des exigences est une étape critique dans le processus de développement de projet informatique. Elle détermine les fonctionnalités que le système doit fournir pour répondre aux besoins et aux attentes des utilisateurs. Cette section présente les besoins fonctionnels et non fonctionnels identifiés pour la plateforme de covoiturage. Ces besoins sont identifiés en collaboration avec les parties prenantes, en tenant compte des objectifs du projet et des exigences du marché.

1.5.1 Besoins fonctionnels

La plateforme de covoiturage est caractérisée par deux acteurs principaux qui sont le conducteur et le passager ainsi que les systèmes de notifications, de tarification et de correspondance. Dans la suite, seuls les besoins fonctionnels de la plateforme qui concernent la partie du conducteur seront mentionnés puisque l'attention sera uniquement portée sur la partie du projet qui concerne le conducteur.

Les besoins fonctionnels de cette plateforme sont :

- Le conducteur doit s'authentifier pour accéder à la plateforme.
- Le conducteur peut créer des offres de covoiturage tout en spécifiant les détails de l'offre comme les dates de départ et d'arrivée, les points de départ et d'arrivée et le nombre de places disponibles.
- Le conducteur est capable de modifier les détails de l'offre avant de le publier.
- Le conducteur doit publier l'offre après sa création.
- Le conducteur peut annuler l'offre.
- Après que le système de tarification fixe le prix, le conducteur peut gérer les demandes soumises tout en acceptant ou refusant ces derniers.
- Un système de notification doit notifier les utilisateurs lorsque certains événements se produisent.

1.5.2 Besoins non fonctionnels

Les besoins non fonctionnels sont l'ensemble des contraintes techniques qui caractérisent la plateforme en termes de performances. Les principales exigences sont :

- **Modularité** : Ajout facile de nouvelles fonctionnalités ou remplacement de modules existants.
- **Extensibilité** : permet l'intégration avec d'autres systèmes externes et garantit la flexibilité et l'évolutivité.
- **Tolérance aux pannes** : assurer la continuité du service lorsque d'autres services échouent.
- **Maintenabilité** : Facilité de maintenance et de correction des erreurs dans le code source.

Ces besoins non fonctionnels sont essentiels pour garantir la qualité, la robustesse et la pérennité de la plateforme.

1.5.3 Diagramme de cas d'utilisation globale

La figure 1.1 présente le diagramme général des cas d'utilisation de la plateforme de covoiturage. Chaque acteur dispose d'un ensemble de permissions qui lui sont accordées et d'un ensemble de fonctions qui lui sont associées.

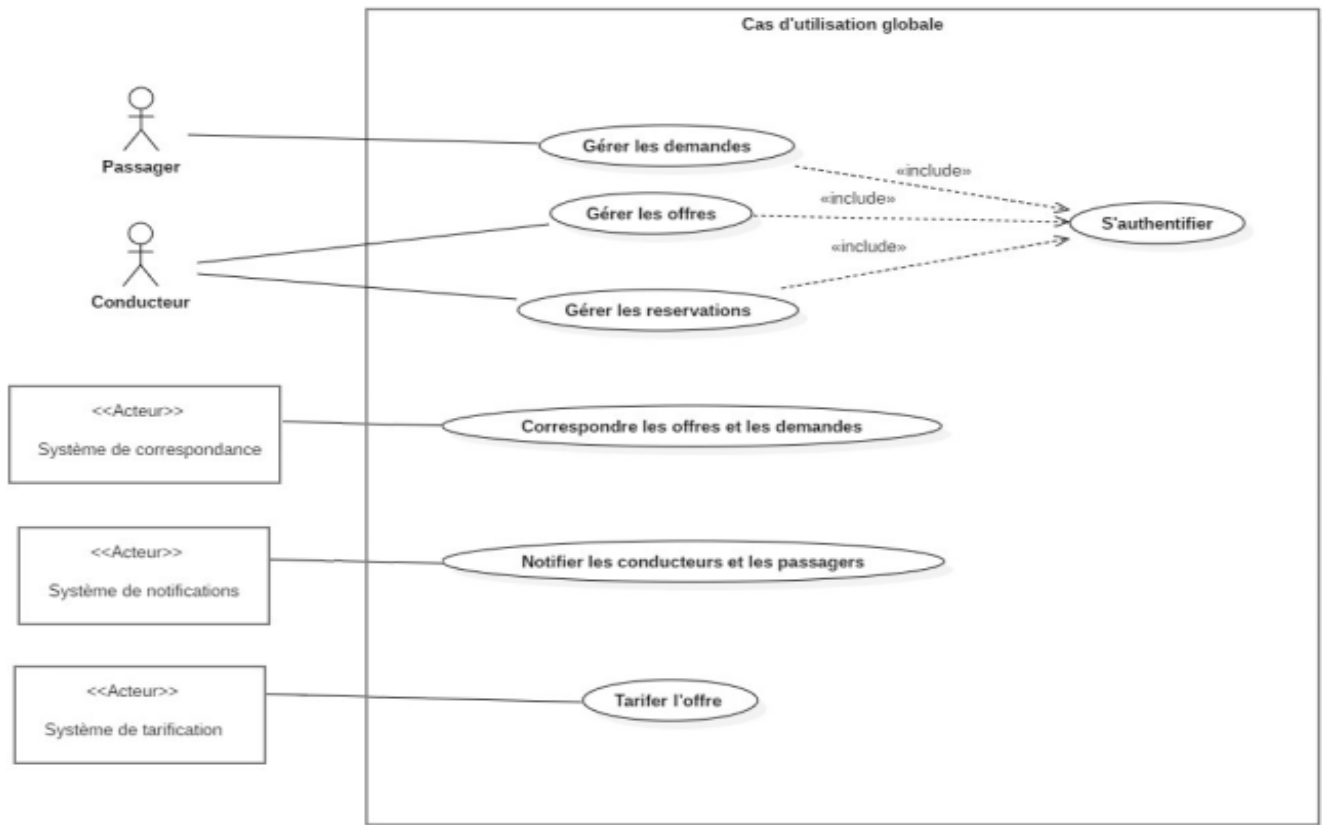


FIGURE 1.1 – Diagramme de cas d'utilisation globale.

1.6 Raffinement des cas d'utilisation

Cette partie du projet se concentre sur l'amélioration des cas d'utilisation de la plateforme de covoiturage. L'objectif est de définir précisément les différentes fonctionnalités de la plateforme et l'interaction entre l'utilisateur et le système. Cet raffinement permettra de mieux comprendre les besoins des utilisateurs et de concevoir une plateforme qui réponde efficacement à leurs besoins.

1.6.1 Raffinement de cas d'utilisation “Gérer les offres”

La figure 1.2 montre que le conducteur doit s'authentifier pour avoir accès à son compte où il peut créer, modifier, publier et annuler ces offres de covoiturage.

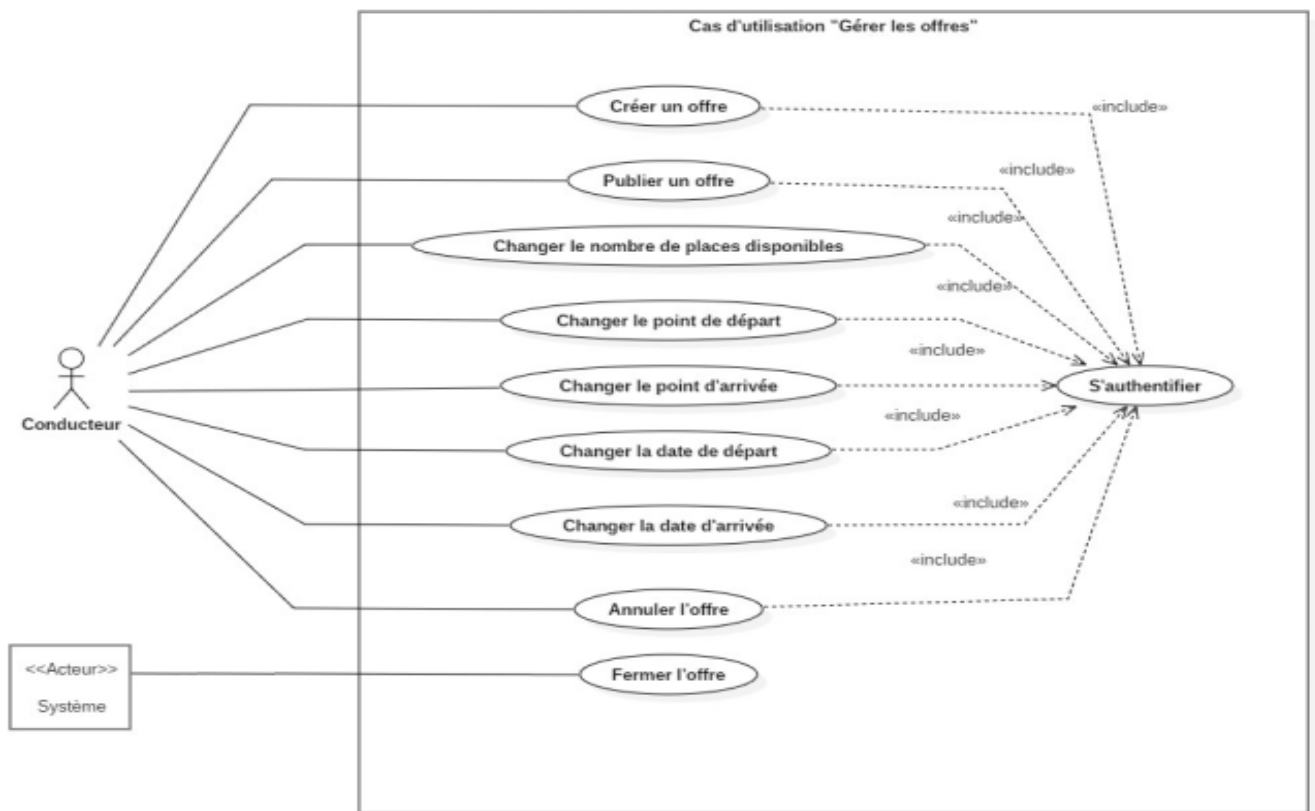


FIGURE 1.2 – Cas d'utilisation "Gérer les offres."

1.6.2 Raffinement de cas d'utilisation "Tarifier les offres"

La figure 1.3 montre que le système de tarification peut fixer le prix d'une offre ou le modifier au cas où le conducteur a modifié les détails de cette offre.

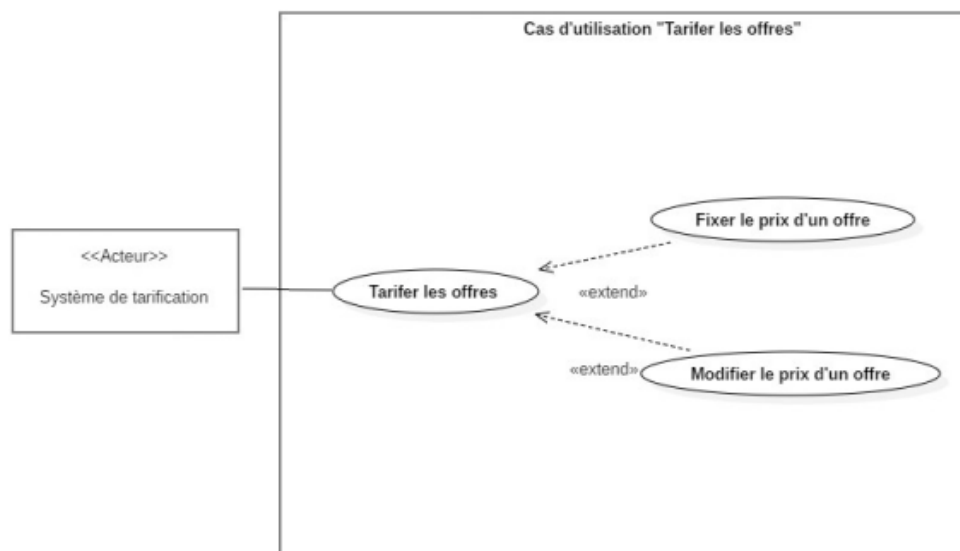


FIGURE 1.3 – Cas d'utilisation "Tarifier les offres".

1.6.3 Raffinement de cas d'utilisation "Gérer les réservations"

La figure 1.4 montre que le conducteur peut créer ou annuler ses propres voyages ainsi qu'accepter ou refuser les demandes qu'il a reçues.

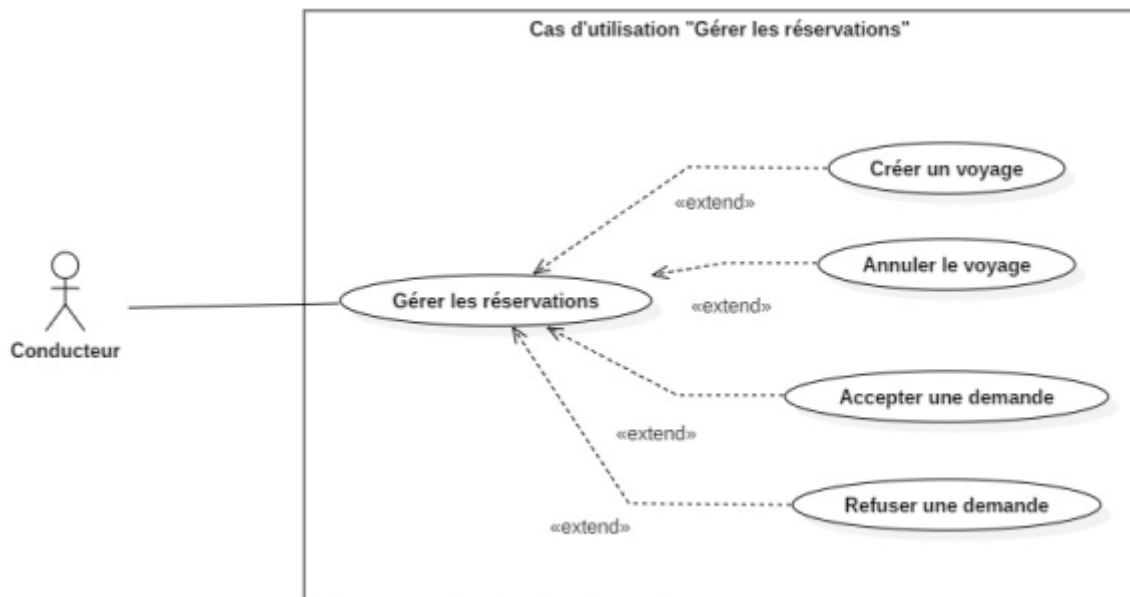


FIGURE 1.4 – Cas d'utilisation "Gérer les réservations".

1.6.4 Raffinement de cas d'utilisation “Notifier les conducteurs et les passagers”

La figure 1.5 montre que le système de notifications envoie des notifications aux utilisateurs à base des événements qui se produisent.

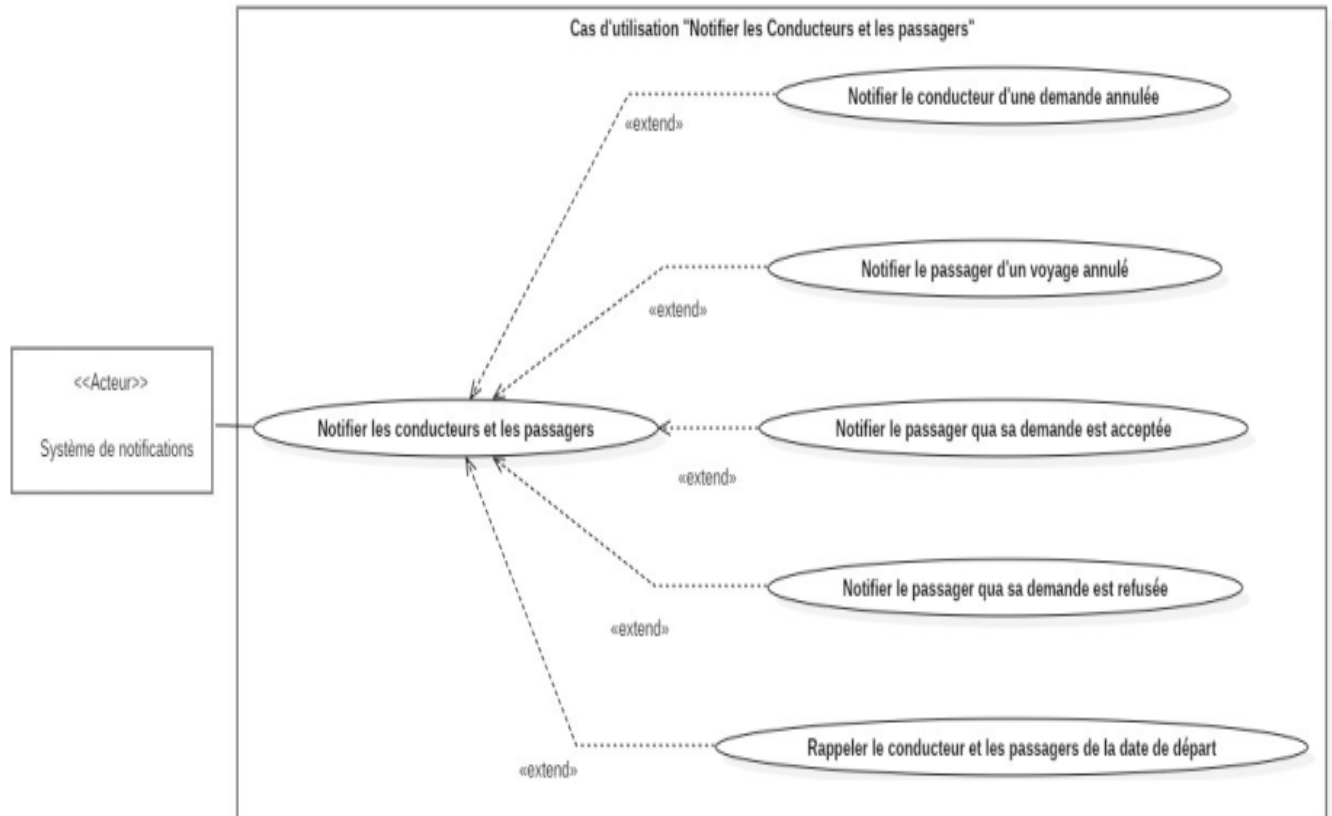


FIGURE 1.5 – Cas d'utilisation "Notifier les conducteurs et les passagers"

Conclusion

En bref, ce chapitre décrit le contexte et les objectifs de notre projet, ainsi que les principales étapes de notre méthodologie de travail. Il résume l'état actuel de l'art en matière de covoiturage et identifie les besoins auxquels la plateforme devait répondre. Ce chapitre prépare le terrain pour comprendre le reste du travail et les choix faits pour répondre aux exigences du projet. Dans le chapitre suivant, nous décrirons en détail la conception de notre solution. Le chapitre suivant détaille la conception de la solution proposée.

Chapitre 2

Architecture et Conception

Introduction

Ce chapitre est consacré à l'architecture et à la conception de la plateforme de covoiturage. Il examine les différentes approches et décisions qui ont été prises pour garantir l'évolutivité, la flexibilité et la maintenabilité de la plateforme. Le chapitre fournit une vue d'ensemble de l'architecture microservices de la plateforme. L'objectif principal de ce chapitre est de fournir une compréhension de haut niveau de l'architecture et de la conception de la plateforme.

2.1 La conception globale de l'application

Après avoir défini la vision globale de l'application, il est temps de se pencher sur sa conception et son architecture.

2.1.1 Architecture globale adaptée à l'application

La phase de sélection de l'architecture adaptée a permis de retenir l'approche de l'architecture microservices pour répondre aux besoins spécifiques de l'application.

2.1.1.1 Choix de l'architecture microservices pour l'application

L'objectif de la plateforme est d'opérer à grande échelle avec une haute disponibilité, une grande évolutivité et une grande tolérance aux pannes. Et donc ça nécessite l'implémentation de l'architecture microservices puisque cette approche permet de décomposer l'application en services plus petits, indépendants et modulaires. Chaque service peut être développé, déployé et mis à l'échelle de manière indépendante, ce qui accroît la flexibilité et réduit le risque d'un point de défaillance de l'application. En cas de défaillance d'un service, le système ne s'effondre pas, seul le service concerné est touché. En outre, les microservices permettent d'ajouter facilement de nouvelles fonctionnalités ou de mettre à jour les fonctionnalités existantes sans affecter le reste du système. En fin de compte, l'architecture microservices fournit une plateforme plus résiliente, plus évolutive et plus facile à maintenir, qui peut s'adapter rapidement aux besoins changeants des utilisateurs[5].

2.1.1.2 Avantages de la conception pilotée par le domaine pour le développement d'une architecture de microservices

La décomposition d'une application en microservices peut être un processus complexe, en particulier lorsqu'il s'agit de domaines complexes. Dans de tels cas, il peut être difficile d'identifier

les limites de chaque microservice et de s'assurer que chaque service est autonome, tout en étant pleinement intégré au système global. C'est là que l'approche de la conception pilotée par le domaine (DDD) entre en jeu. En se concentrant sur la compréhension du domaine et en définissant des contextes délimités, un langage commun et un modèle de domaine clair, il est possible de décomposer plus efficacement l'application en microservices plus petits et plus faciles à gérer[17]. Afin de décomposer notre application, il faut tenir compte aux éléments clés du DDD suivantes :

- **Modèle du domaine** : Le modèle de domaine est la connaissance organisée et structurée du problème. Le modèle de domaine doit représenter le vocabulaire et les concepts clés du domaine du problème et identifier les relations entre toutes les entités du domaine.
- **Langage ubiquitaire** : L'utilisation d'un langage commun dans l'application permet aux personnes qui travaillent sur ce projet d'avoir une compréhension commune du domaine d'activité. Ce langage permet d'éliminer les ambiguïtés et les malentendus, ce qui se traduit par un processus de développement plus efficace et plus efficient.
- **Contexte Borné** : Les contextes délimités permettent de s'assurer que chaque microservice a une responsabilité clairement définie et une compréhension claire des données qu'il gère, ce qui améliore la modularité et la maintenabilité du système.
- **Entité** : un objet du domaine qui a une identité unique et qui reste inchangé dans le temps, représentant souvent quelque chose de tangible ou de conceptuel.
- **Objet valeur** : un objet du domaine qui n'a pas d'identité unique et qui est défini par ses attributs ou ses propriétés, représentant souvent quelque chose qui peut être mesuré ou quantifié.
- **Aggrégat** : Dans le DDD, les agrégats sont des groupes d'objets apparentés qui sont traités comme une seule unité de cohérence. Dans cette application, l'utilisation d'agrégats nous permet d'encapsuler la logique métier et les règles d'intégrité des données dans chaque groupe d'objets apparentés ce qui montre que cette approche respecte plus l'approche orientée objets à la différence de l'approche classique qui se rapproche plus du procédural. Cela simplifie le code et améliore la maintenabilité globale du système. En outre, cela nous permet de mieux gérer le contrôle de la concurrence et la cohérence des données, en veillant à ce que les modifications apportées aux objets apparentés soient effectuées de manière cohérente et atomique.

En appliquant les principes DDD, chaque microservice peut avoir une responsabilité bien définie et une compréhension claire des données qu'il gère. Cette approche permet également une communication plus efficace entre l'équipe de développement et les experts du domaine, ce qui se traduit en fin de compte par un système plus cohérent, plus facile à maintenir et plus évolutif[17].

2.1.1.3 Modélisation des domaines complexes à l'aide de l'Event Storming

Pour parvenir au DDD, nous avons besoin de techniques efficaces pour comprendre et modéliser des domaines complexes. L'une de ces techniques est l'event storming. L'Event Storming est une technique d'atelier collaboratif utilisée dans le DDD pour modéliser des domaines d'activité complexes. Elle implique un groupe de personnes de différents rôles qui travaillent ensemble pour identifier les événements qui se produisent au sein d'un processus d'entreprise et les objets de domaine associés ainsi que leurs relations. Le processus consiste à visualiser le domaine à l'aide de notes autocollantes, représentant la séquence d'événements dans un processus commercial. Cette technique favorise une compréhension commune du langage du domaine, des processus impliqués et des relations entre les différents éléments. Le résultat est un modèle clair et concis du domaine qui peut être utilisé pour guider le processus de développement.

L'Event storming est un outil puissant qui permet une itération rapide et contribue à garantir que le logiciel résultant est conforme aux exigences du projet[2]. Les événements, les commandes, les acteurs, le context mapping, les politiques et les agrégats sont des termes couramment utilisés dans le DDD :

- **Événements** : il s'agit d'événements ou de faits importants qui se produisent dans le domaine et qui peuvent être enregistrés et traités. Les événements sont immuables et représentent quelque chose qui s'est produit dans le passé.
- **Commandes** : il s'agit de demandes adressées au système pour qu'il effectue une certaine action ou opération. Elles sont mutables et représentent quelque chose qui doit être fait dans le futur.
- **Acteurs** : ce sont les personnes, les systèmes ou les processus qui effectuent des actions et interagissent avec le système. Les acteurs peuvent donner des ordres et réagir à des événements.
- **Règles** : règles et contraintes qui régissent le comportement du système. Les politiques peuvent être implémentées dans le cadre du modèle de domaine afin d'en garantir la cohérence et l'exactitude.
- **Agrégats** : il s'agit de groupes d'objets apparentés qui forment une frontière autour d'une partie du domaine. Les agrégats assurent la cohérence et l'intégrité en garantissant que toutes les modifications apportées aux objets qu'ils contiennent sont effectuées de manière cohérente et atomique tout en gardant le principe de l'encapsulation de l'approche orientée objets.
- **Carte de contextes** : est une technique de DDD utilisée pour gérer les relations et les limites entre les contextes délimités. Il s'agit d'identifier les différents contextes d'un domaine et de définir comment ils interagissent entre eux. Il s'agit notamment de définir le langage utilisé dans chaque contexte, ainsi que les relations, les dépendances ou les points d'intégration entre eux. En créant une carte claire des contextes et de leurs interactions, le context mapping permet de s'assurer que le modèle du domaine est organisé et facile à comprendre, et que la communication entre les membres de l'équipe est rationalisée.

La figure 2.1 montre les notes autocollantes qui présentent les différents éléments de l'event storming :



FIGURE 2.1 – Représentation des différents éléments d'event storming[18].

2.1.1.4 Réalisation de la décomposition du système

L'exercice d'event storming a permis d'identifier les limites de chaque contexte dans l'application et, par conséquent, des microservices clairs et distincts ont été définis pour chaque contexte,

permettant la décomposition de la plateforme en composants plus petits et indépendants qui peuvent être développés, testés et maintenus séparément. La figure 2.2 illustre le résultat de la modélisation du contexte limité de la gestion des offres.

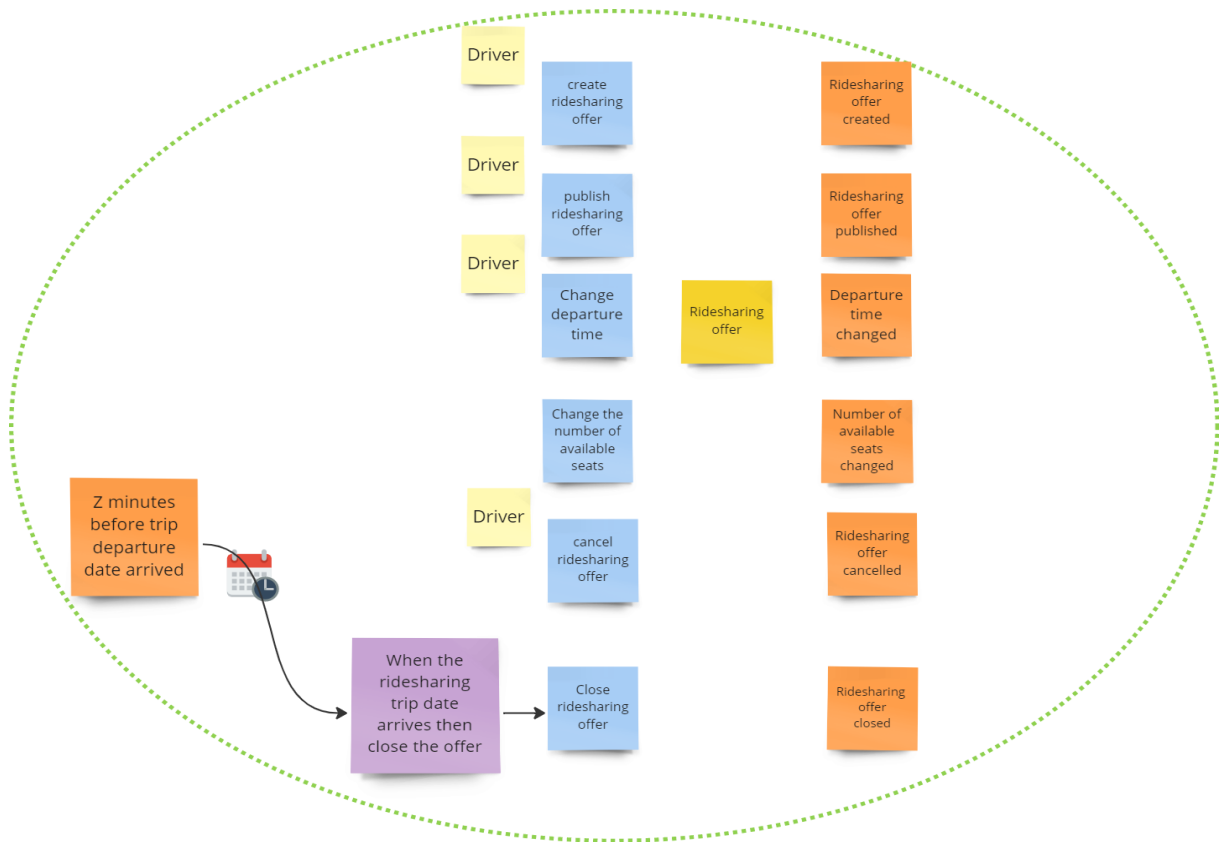


FIGURE 2.2 – Contexte de gestion des offres.

La figure 2.3 présente le contexte de la tarification des offres.

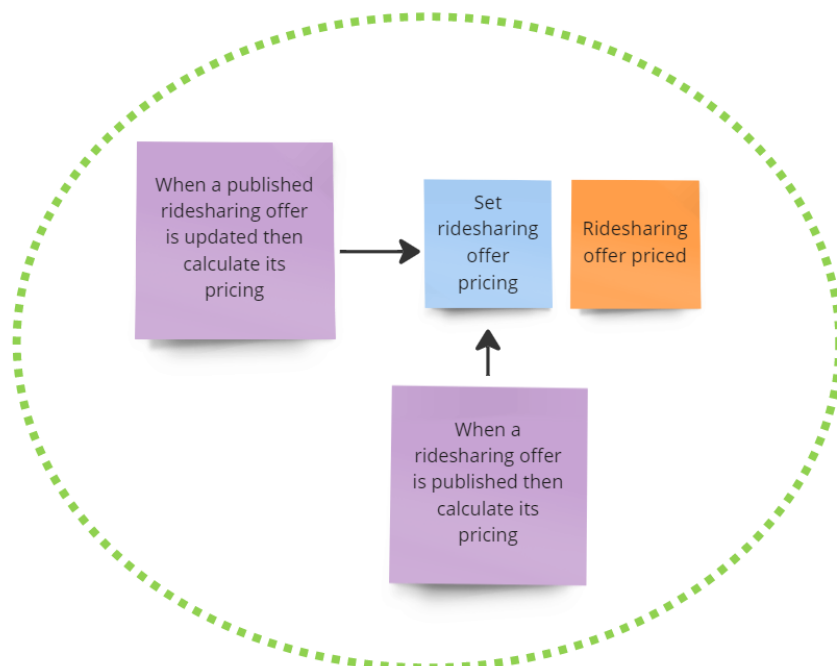


FIGURE 2.3 – Contexte de tarification.

La figure 2.4 présente le contexte qui s'occupe de la gestion des réservations qui permet à un conducteur de gérer les demandes sur son offre.

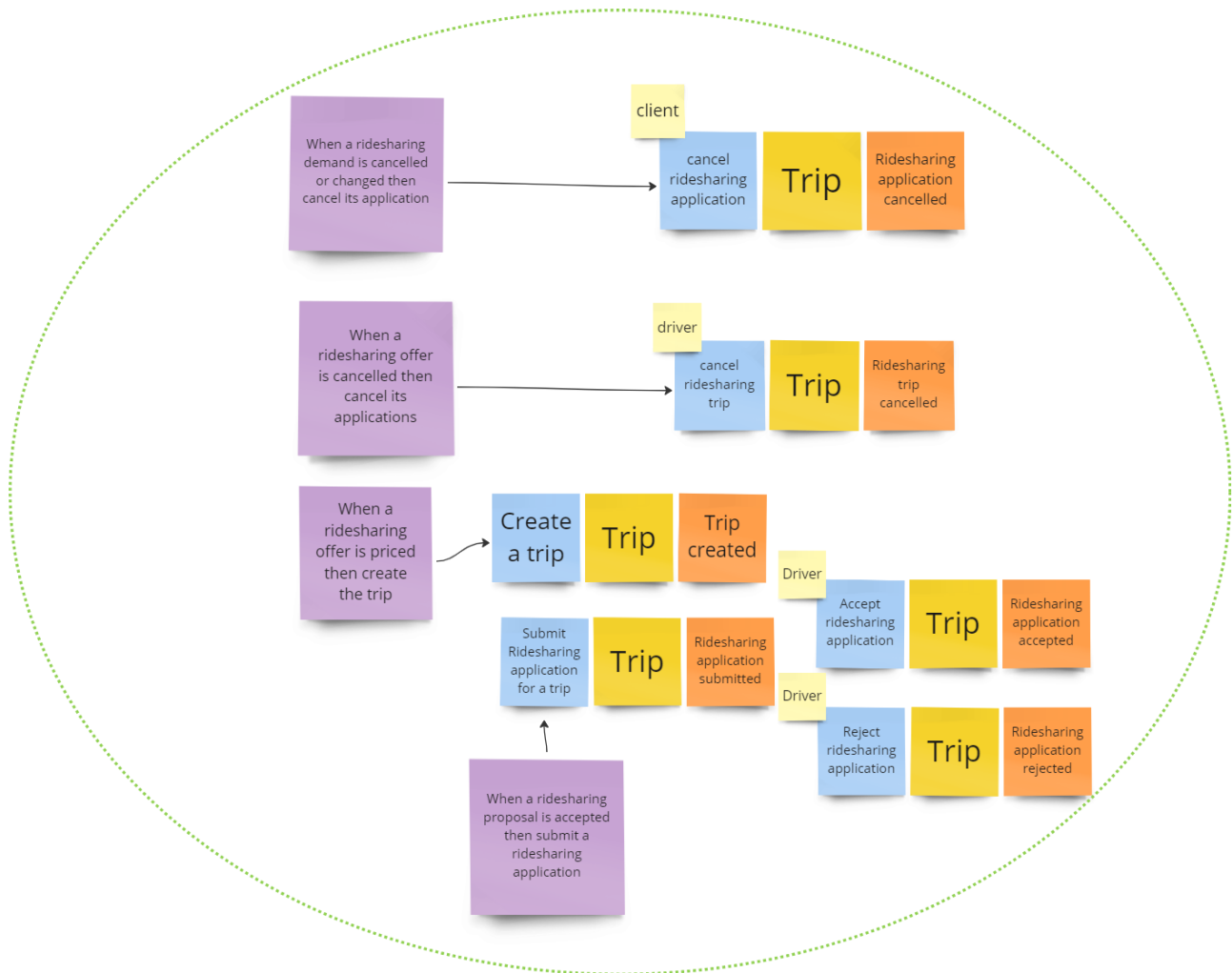


FIGURE 2.4 – Contexte de gestion des réservations.

La figure 2.5 montre comment le contexte de notifications s'occupe d'informer les utilisateurs quand des événements se produisent.



FIGURE 2.5 – Contexte de notifications.

La carte de contextes permet de formaliser le couplage entre les sous-domaines et de décider de sa force dès la phase de conception. Il sert également à déterminer les responsabilités de chaque sous-domaine dans le système global. Il existe différents types de couplages :

- **Noyau partagé** : Un sous-ensemble du modèle de domaine partagé par deux ou plusieurs contextes délimités.
- **Contexte en aval** : Contexte borné qui reçoit un message ou une demande d'un contexte délimité en amont.
- **Contexte en amont** : c'est le contexte borné qui envoie un message ou une demande à un contexte délimité en aval.
- **Couche anti-corruption** : Couche qui traduit les communications entre différents contextes délimités ayant des langages et des modèles différents.
- **Langage publié** : Un langage commun partagé par différents contextes bornés afin d'éviter les malentendus et de faciliter la communication.

La figure 2.6 montre l'ajout de la carte de contextes permettant la communication entre les différents contextes.

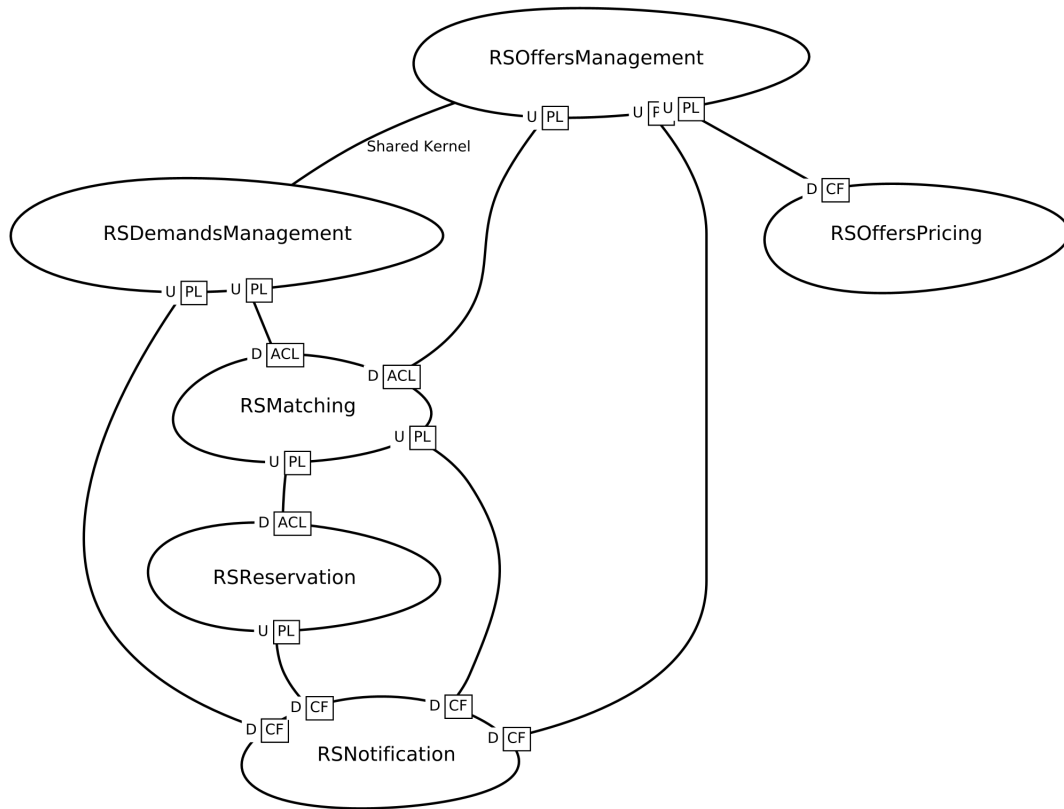


FIGURE 2.6 – La carte de contextes des différents contextes bornés.

Le résultat final est un système plus souple, plus résilient et plus facile à maintenir, qui peut facilement s'adapter à l'évolution des besoins de l'entreprise[10].

2.1.2 Patron d'architecture adopté pour les microservices

Pour garantir la cohérence et la maintenabilité de l'architecture de microservices, il faut adopter une approche méthodique pour concevoir et conditionner chaque microservice de manière uniforme. Cette approche permet de simplifier le développement et la documentation tout en facilitant la collaboration entre les membres de l'équipe.

2.1.2.1 Diagramme de classe générique d'un microservice

Le diagramme de classe illustré dans la figure 2.7 a été utilisé comme référence pour la conception détaillée de chaque microservice.

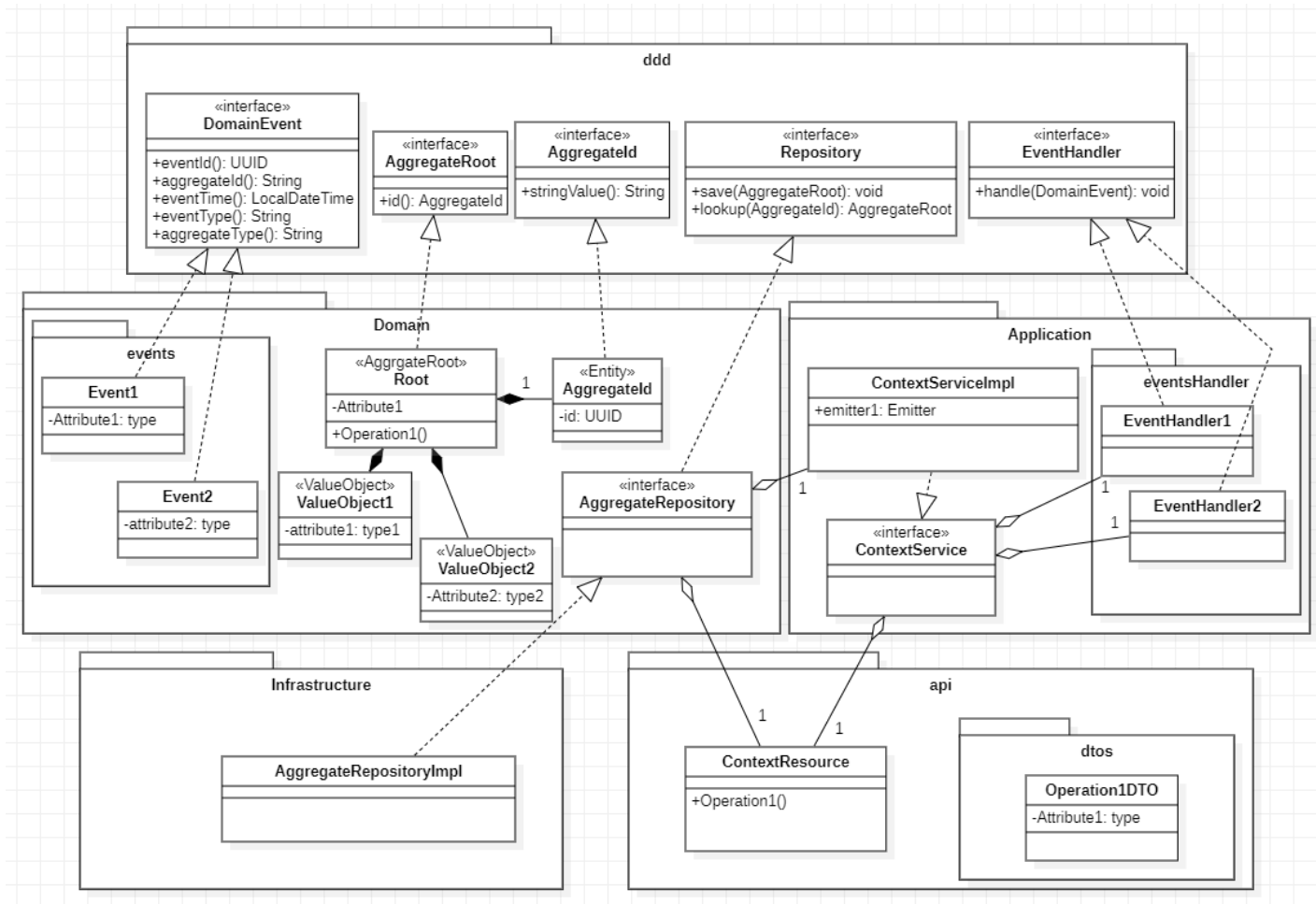


FIGURE 2.7 – Diagramme de classe générique

D'après la figure 2.7, Chaque microservice est décomposé en cinq paquetages différents.

- Le paquetage **Conception pilotée par le domaine (DDD)** : il encapsule les interfaces clés et définit des normes claires pour la conception et la mise en œuvre de nos microservices.
- Le paquetage **API** : est responsable de l'interface utilisateur et de l'exposition des fonctionnalités du microservice aux autres services de l'application grâce aux opérations de la classe ContextResource. Les classes DTO sont utilisées pour encapsuler et transférer des données entre différents composants d'application sans avoir à partager des objets de domaine ou des entités persistantes.
- Le paquetage **Application** : Ce paquetage contient des classes qui mettent en œuvre la logique métier des microservices.
Il contient l'interface ContextServiceImpl qui encapsulent la logique métier du microservice et des classes qui organisent l'interaction entre les objets du domaine tels que les "event Handlers".
- Le paquetage **Domaine** : son rôle est de modéliser les entités, les agrégats, les valeurs d'objets et les événements qui constituent le domaine du microservice.
- Le paquetage **Infrastructure** : il contient les éléments techniques nécessaires à l'exécution des microservices et constitue la couche de persistance des microservices qui gère la connexion et les interactions avec la base de données. Il fournit une implémentation concrète de l'interface AggregateRepository définie dans le package domain.

La figure 2.8 montre le diagramme d'architecture en oignon de l'application.

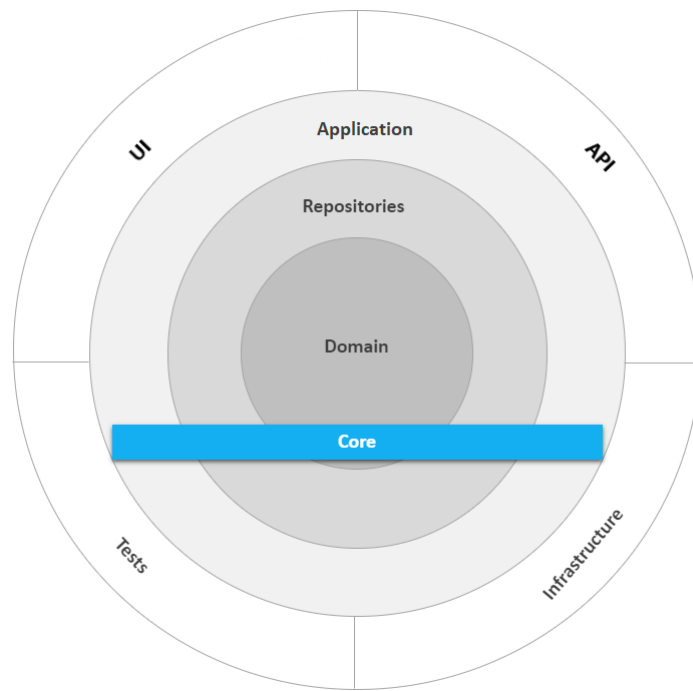


FIGURE 2.8 – Le diagramme d’architecture en oignon. [12]

2.1.2.2 Diagramme de séquence générique d’un microservice

La figure 2.9 représente les interactions entre les différents composants du système. Ce diagramme de séquence met en évidence le flux de messages entre les différents objets, ce qui permet d’illustrer le comportement sous-jacent du système en réponse à certains événements ou demandes.

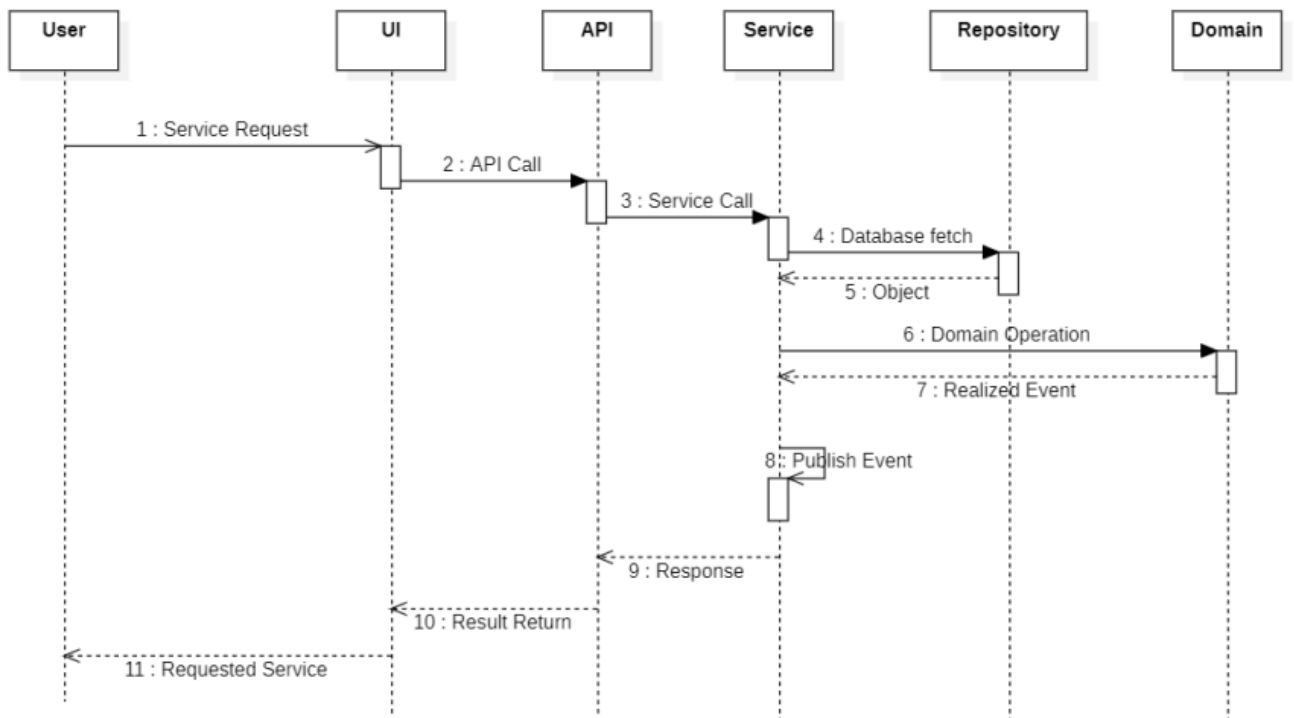


FIGURE 2.9 – Diagramme de séquence générique d’un microservice

2.2 La conception détaillée des microservices

La section précédente a abordé la conception et l'architecture globale de l'application basée sur les microservices. Cette section consiste à approfondir la conception détaillée de chaque microservice.

2.2.1 La conception du microservice "Gestion des offres de covoiturage"

Le microservice "Gestion des offres de covoiturage" est une application autonome qui permet aux conducteurs de créer, de publier, d'annuler et de modifier des offres de covoiturage. Une fois l'offre publiée, cet événement déclenche le microservice de tarification.

2.2.1.1 Diagramme de classe du microservice "Gestion des offres de covoiturage"

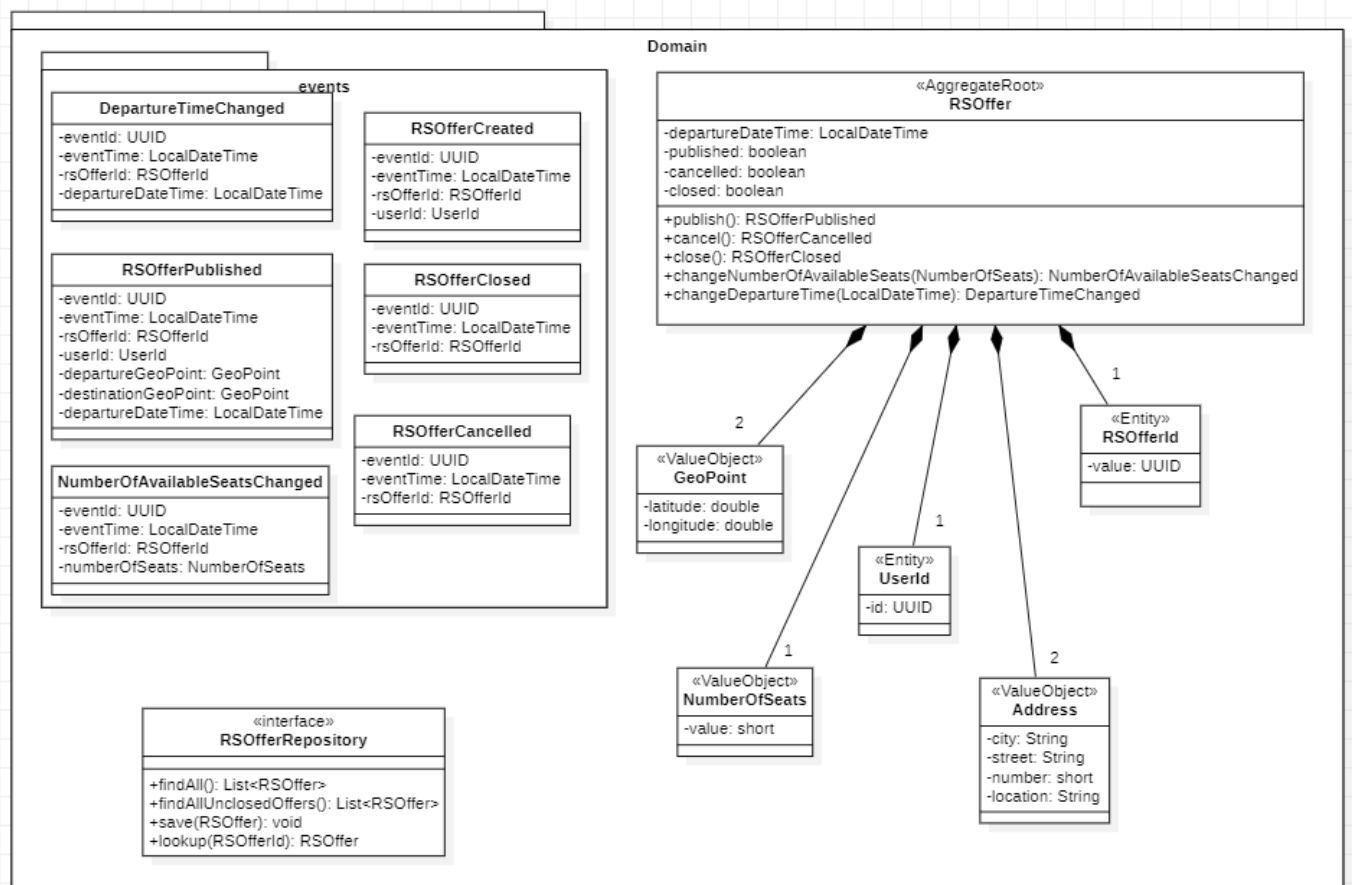


FIGURE 2.10 – Domaine du microservice "Gestion des offres de covoiturage"

La figure 2.10 montre le domaine de ce microservice qui est constitué de l'agrégat **Offre**. La classe **RSOffer** implémente l'interface **AggregateRoot** du paquetage DDD, c'est la racine de l'agrégat, elle est composée par plusieurs objets de valeur tel que les classes **Address**, **NumberOfSeats**, **GeoPoint** et des par entités comme **RSOfferId** et **UserId**.

Toutes les classes du paquetage **events** implémentent l'interface **DomainEvent** du paquetage ddd. Ces événements sont déclenchés par la création, l'annulation, la publication et la fermeture d'une offre ou par des modification tel que le changement de la date de départ de l'offre ou

le nombre de places disponibles puis ils seront consommés par les autres microservices et les systèmes externes.

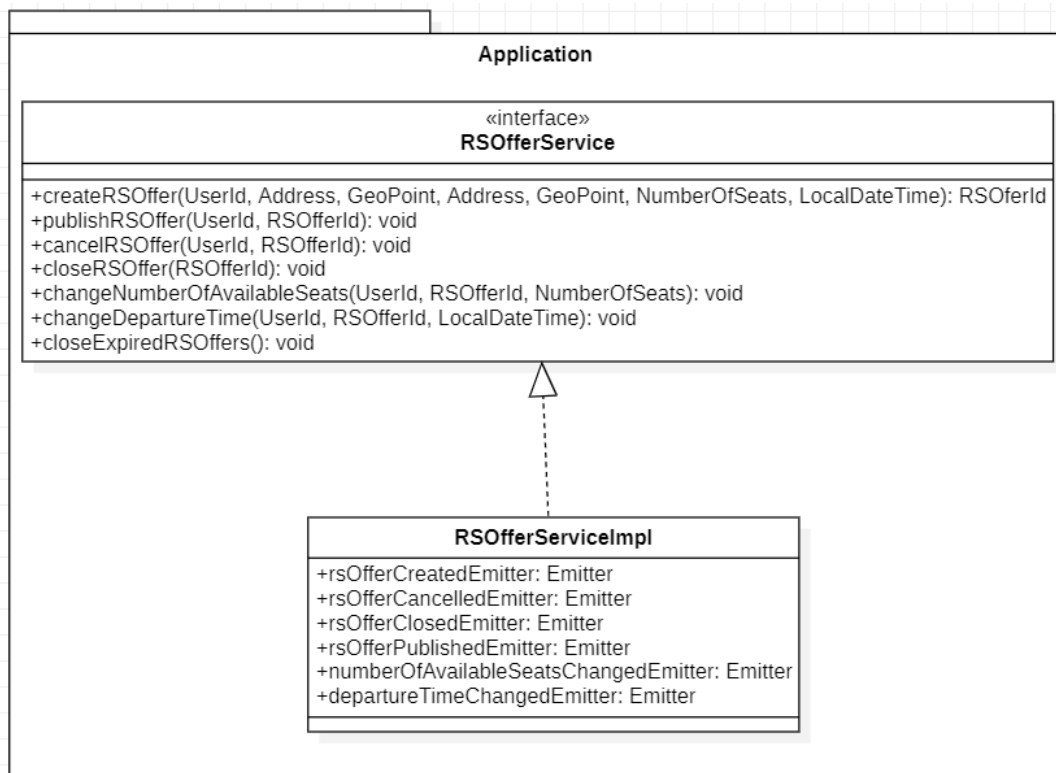


FIGURE 2.11 – Les services du microservice "Gestion des offres de covoiturage"

Les services proposés par ce microservice, comme le montre la figure 2.11 sont la création, l'annulation, la publication, la fermeture et la modification d'une offre.

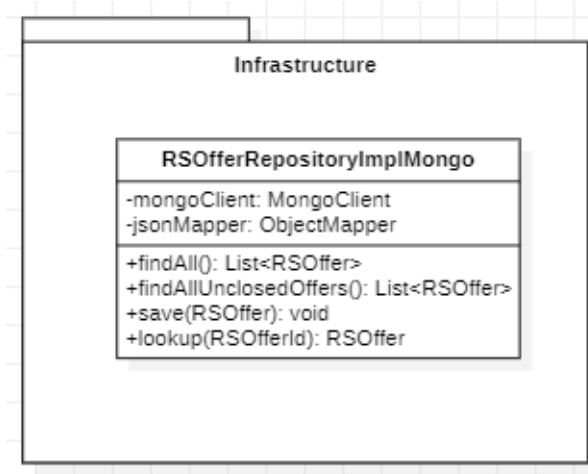


FIGURE 2.12 – L'infrastructure du microservice "Gestion des offres de covoiturage"

La figure 2.12 illustre l'infrastructure de ce microservice, la classe **RSOfferRepositoryImplMongo** implemente l'interface **RSOfferRepository** du paquetage domaine. Elle assure la connexion à la base de données, la sauvegarde, la récupération et la recherche de l'agrégat **RSOffer**.

La figure 2.13 atteste de la conformité de notre implémentation avec les principes et l'approche que nous avons adopté.

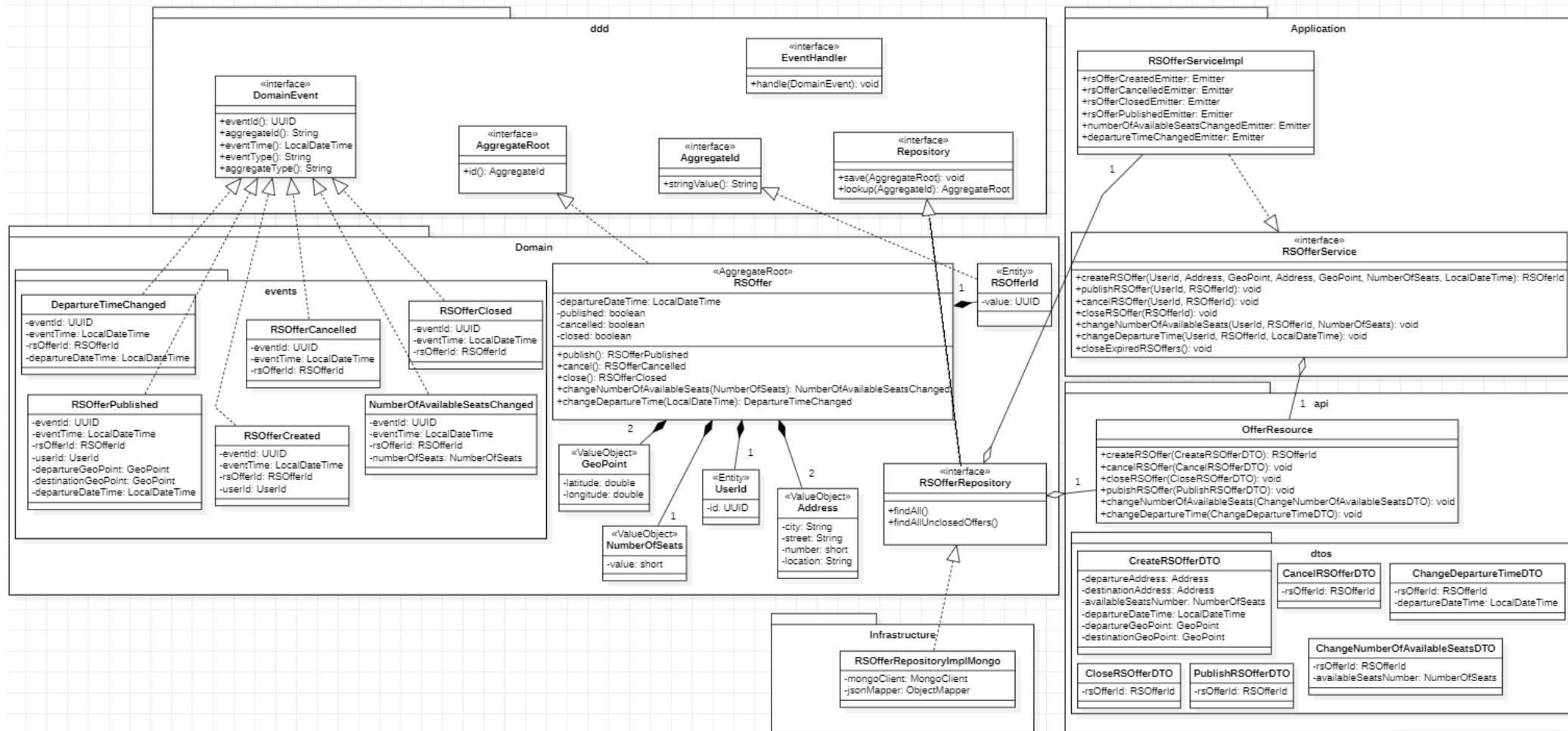


FIGURE 2.13 – Diagramme de classe du microservice "Gestion des offres de covoiturage"

2.2.2 La conception du microservice "Tarifaction"

Le microservice de tarification est un service qui sera déclenché automatiquement après la publication d'une offre dans le microservice de gestion des offres. Son rôle est de déterminer un prix approprié pour l'offre en tenant compte de ses détails tels que l'adresse de départ, l'adresse de destination et le nombre de passagers.

2.2.2.1 Diagramme de classe du microservice "Tarifaction"

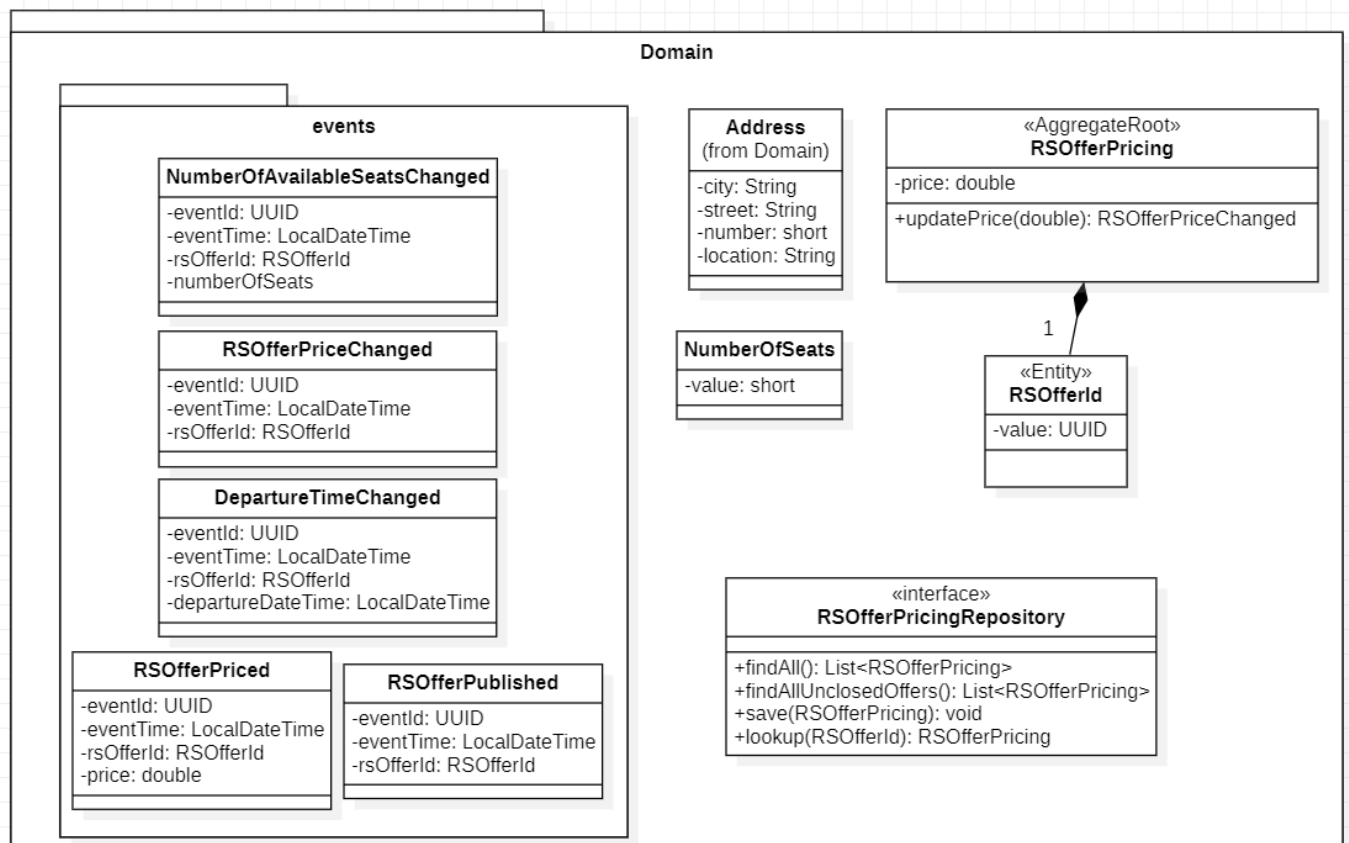


FIGURE 2.14 – Domaine du microservice "Tarifaction"

La figure 2.14 illustre le domaine de ce microservice. La classe **RSOfferPricing** est la racine de l'agrégat, elle implémente l'interface **AggregateRoot** du paquetage ddd. La méthode `updatePrice` permet de tarifier une offre publiée par le microservice de gestion des offres de covoiturage ou bien de mettre à jour le prix d'une offre qui a subi une modification.

Tous les événements implémentent l'interface **DomainEvent** du paquetage ddd et l'interface **RSOfferPricingRepository** hérite de l'interface **Repository** et elle sera implémentée dans l'infrastructure du microservice.

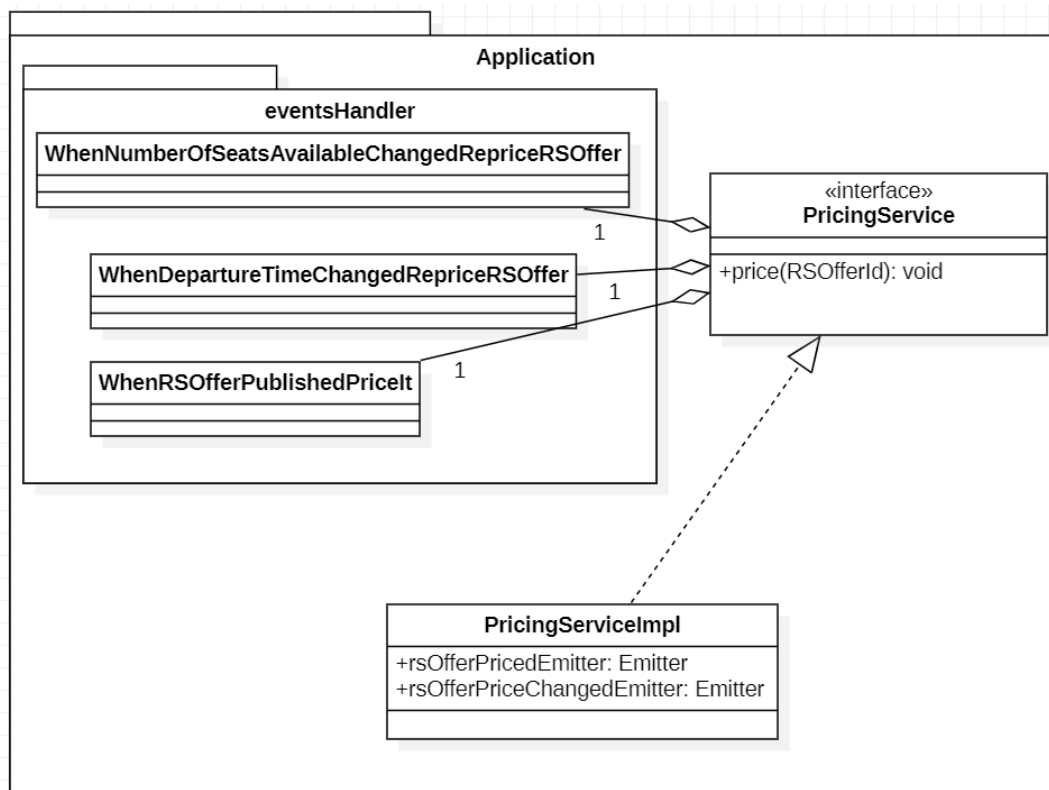


FIGURE 2.15 – Les services du microservice "Tarification"

La figure 2.15 montre le paquetage application de ce microservice qui est uniquement responsable de la tarification des offres. Ainsi, lorsqu'une offre est publiée ou modifiée par le microservice de gestion des offres, les classes du paquetage **eventHandlers** qui implémentent l'interface **EventHandler** du paquetage ddd traiteront automatiquement ces événements pour fixer le prix de l'offre.

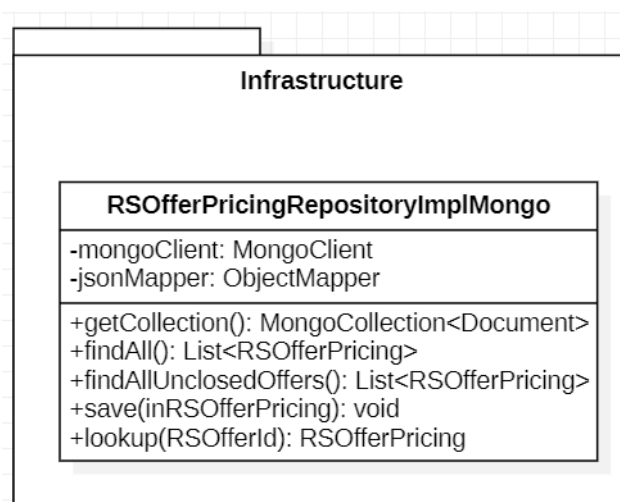


FIGURE 2.16 – L'infrastructure du microservice "Tarification"

La figure 2.16 montre le paquetage infrastructure de ce microservice. **RSOfferPricingRepositoryImplMongo** est une classe concrète qui implémente l'interface **RSOfferPricingRepository** du domaine, elle assure les interactions avec la base de données.

La figure suivante représente le diagramme de classe complet de ce microservice.

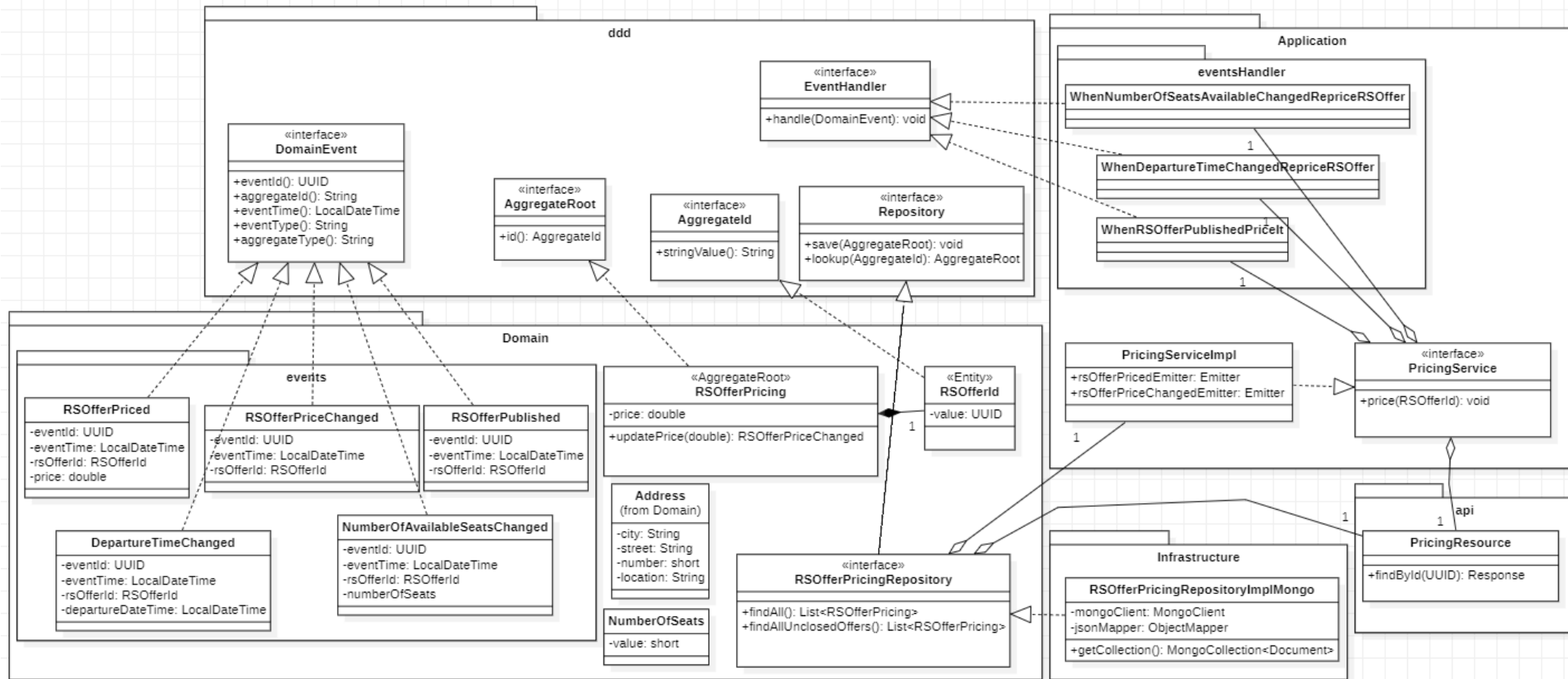


FIGURE 2.17 – Diagramme de classe du microservice "Tarification"

2.2.3 La conception du microservice "Reservations"

Le microservice "Réservations" sera déclenché juste après le microservice de "Tarification". C'est un service de gestion et d'organisation des voyages de covoiturage. Il permet au conducteur de voir les demandes des passagers pour son voyage qui sont envoyées par le microservice "Correspondance", puis il peut accepter ou refuser ces applications. Ce microservice propose aussi une gestion rapide et transparente des annulations, si le conducteur annule son offre de covoiturage, cela entraînera automatiquement l'annulation du voyage et les passagers ayant réservé seront informés de l'annulation. De même, si un passager annule sa demande de réservation, cela entraînera automatiquement l'annulation de l'application pour ce voyage.

2.2.3.1 Diagramme de classe du microservice "Reservations"

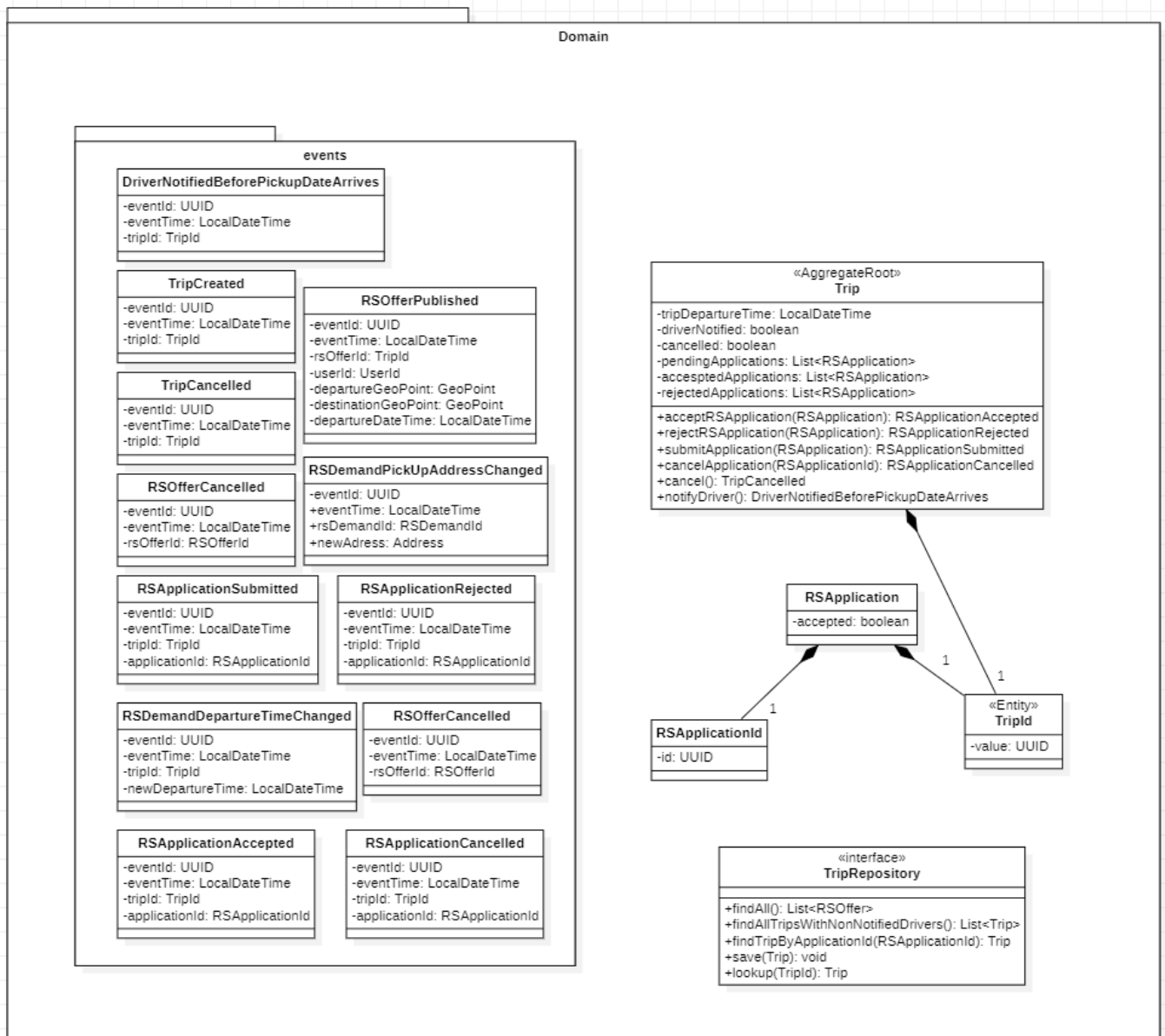


FIGURE 2.18 – Domaine du microservice "Reservations"

La figure 2.18 montre que la racine de l'agrégat du microservice "Reservations" est la classe **Trip** qui implémente l'interface **AggregateRoot** du paquetage ddd. Elle est composée par

l'entité **TripId** qui implémente l'interface **AggregateId** et d'autres détails sur le voyage tel que les applications acceptées, refusées ou en attentes.

Les événements **RSApplicationAccepted**, **RSApplicationRejected**, **RSApplicationCancelled** et **RSApplicationSubmitted** sont déclenchés après l'acceptation, le refus, l'annulation ou la soumission d'une application. les autres événements dépendent de différentes opérations qui peuvent être effectuées par le passager ou le conducteur. Tous ces événements implémentent l'interface **DomainEvent** du paquetage ddd et l'interface **TripRepository** hérite de l'interface **Repository** du ddd.

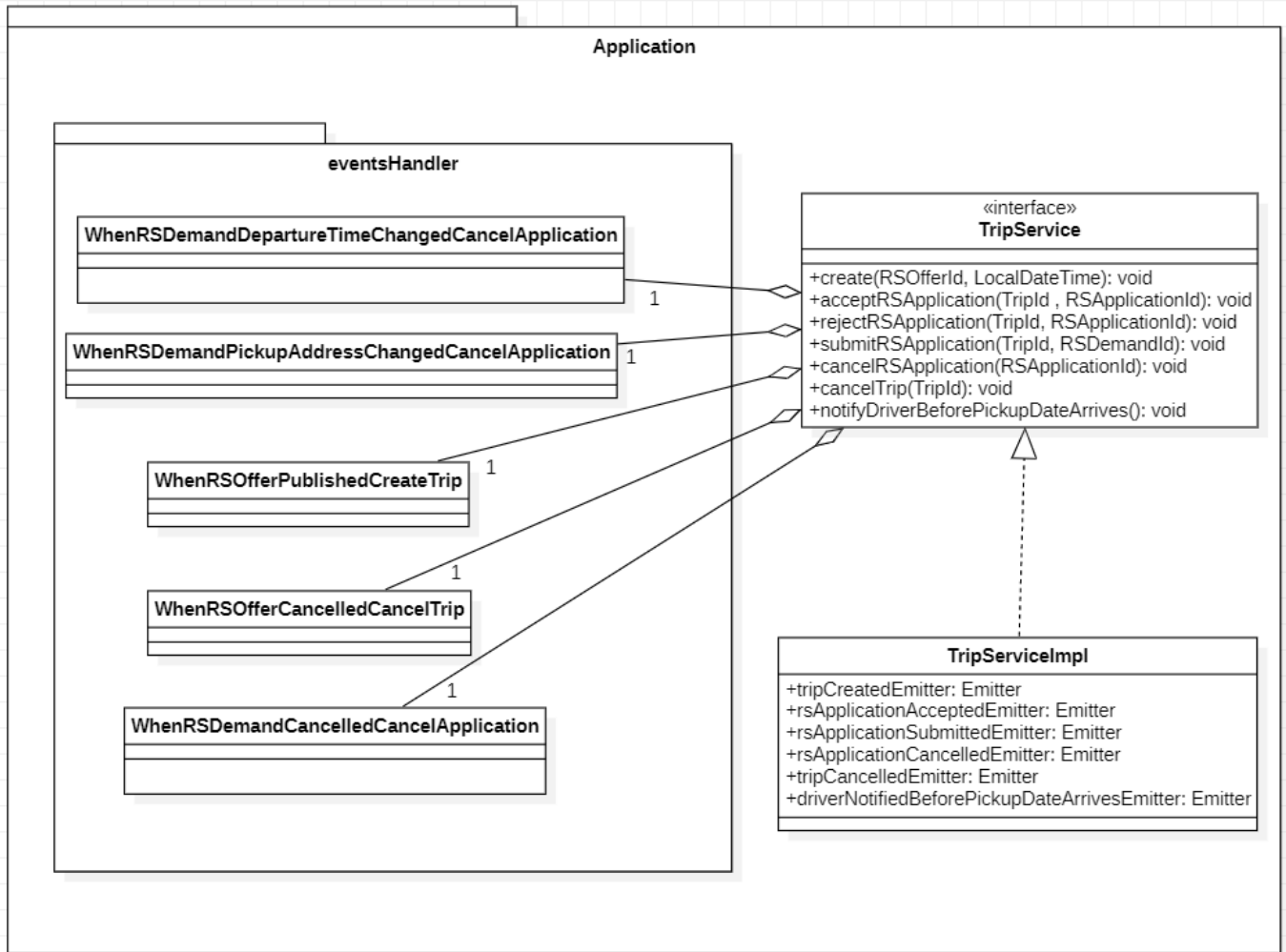


FIGURE 2.19 – Les services du microservice "Reservations"

Comme indiqué dans la figure 2.19, le microservice "Reservations" offre des services au passager tel que soumettre ou annuler une application et d'autres services au conducteur comme accepter ou refuser une application.

Toutes les classes du paquetage eventsHandler implémentent l'interface **EventHandler** du paquetage ddd et assurent l'automatisation de certaines opérations tel que la création d'un voyage après la publication et la tarification d'une offre ou l'annulation d'un voyage après l'annulation de l'offre.

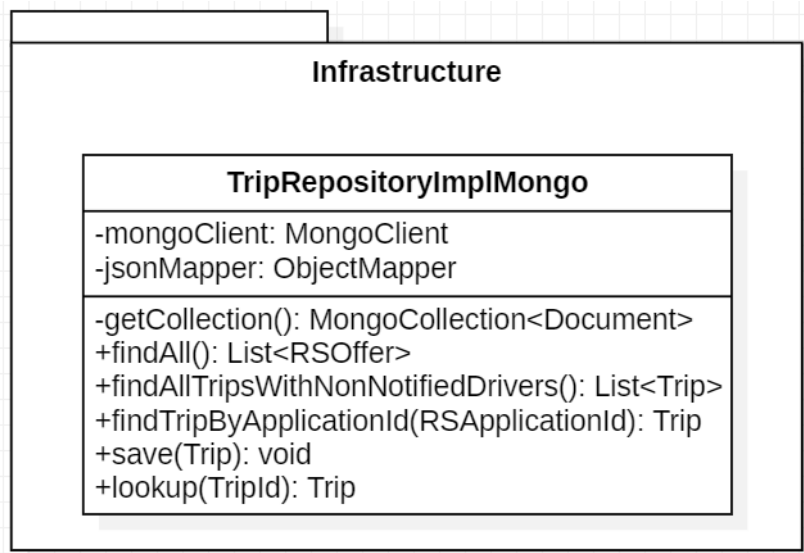


FIGURE 2.20 – L'infrastructure du microservice "Reservations"

La figure 2.20 représente le paquetage infrastructure de ce microservice. La classe **TripRepositoryImplMongo** implémente l'interface **TripRepository** du domaine. Son attribut `mongoClient` permet la connectivité et l'interaction avec une base de données MongoDB ainsi que l'attribut `jsonMapper` permet la transformation d'objets au format JSON.

La figure 2.25 représente le diagramme de classe complet de ce microservice.

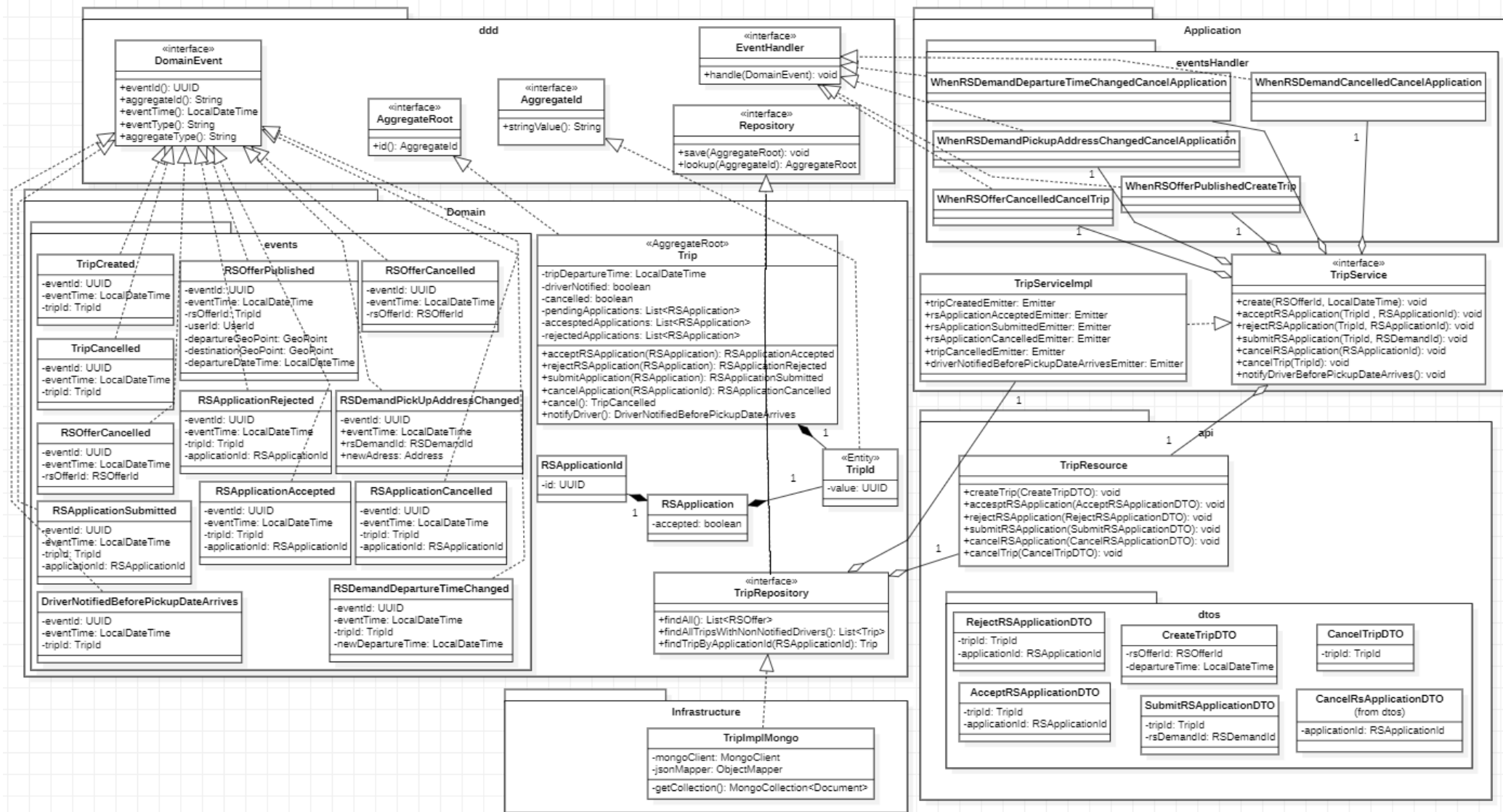


FIGURE 2.21 – Diagramme de classe du microservice "Reservations"

2.2.4 La conception du microservice "Notifications"

Le microservice "Notifications" est un service qui permet de notifier les passagers et les conducteurs des événements importants liés aux voyages en covoiturage. Il est responsable de l'envoi des e-mails de notification en temps réel aux utilisateurs concernés. Lorsqu'un événement important se produit, comme une demande de réservation acceptée ou refusée, une annulation de voyage, ou une modification de l'itinéraire, le microservice "Réservations" utilise le microservice "Notifications" pour envoyer une notification instantanée aux passagers et/ou aux conducteurs concernés. Il est aussi responsable de rappeler le conducteur et les passagers d'un voyage en covoiturage quelques minutes avant l'heure de départ.

2.2.4.1 Diagramme de classe du microservice "Notifications"

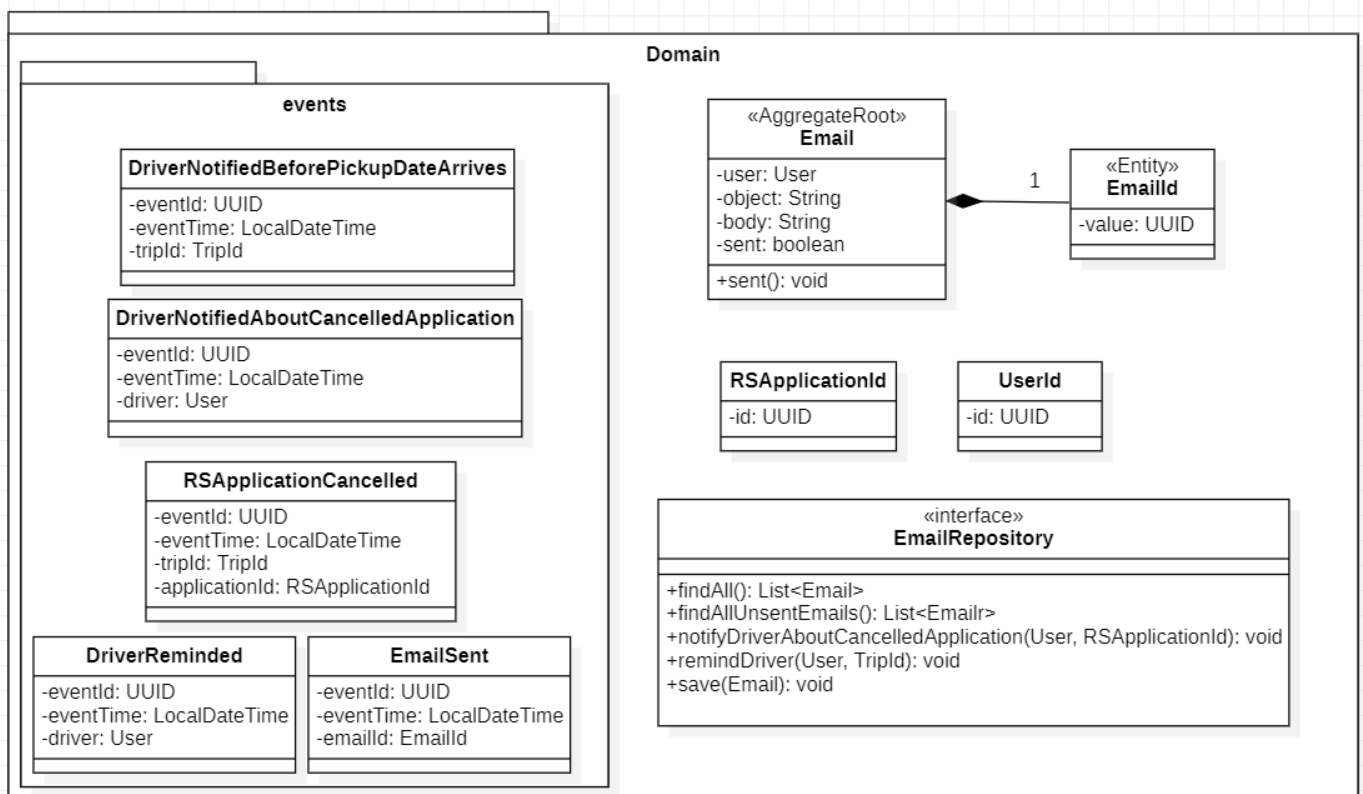


FIGURE 2.22 – Domaine du microservice "Notifications"

La figure 2.22 illustre de paquetage application du microservice "Notifications". La classe **Email** est la racine de l'agrégat, elle implemente l'interface **AggregateRoot** du paquetage ddd est elle contient les détails de la notification à envoyer comme l'objet et corps de l'email. Toutes les classes du paquetage **events** sont des événements spécifiques à ce microservice et elles implementent l'interface **DomainEvent** du ddd alors que l'interface **EmailRepository** hérite de l'interface **Repository** du ddd et offre des méthodes abstraites comme la recherche, la sauvegarde et la récupération des Emails.

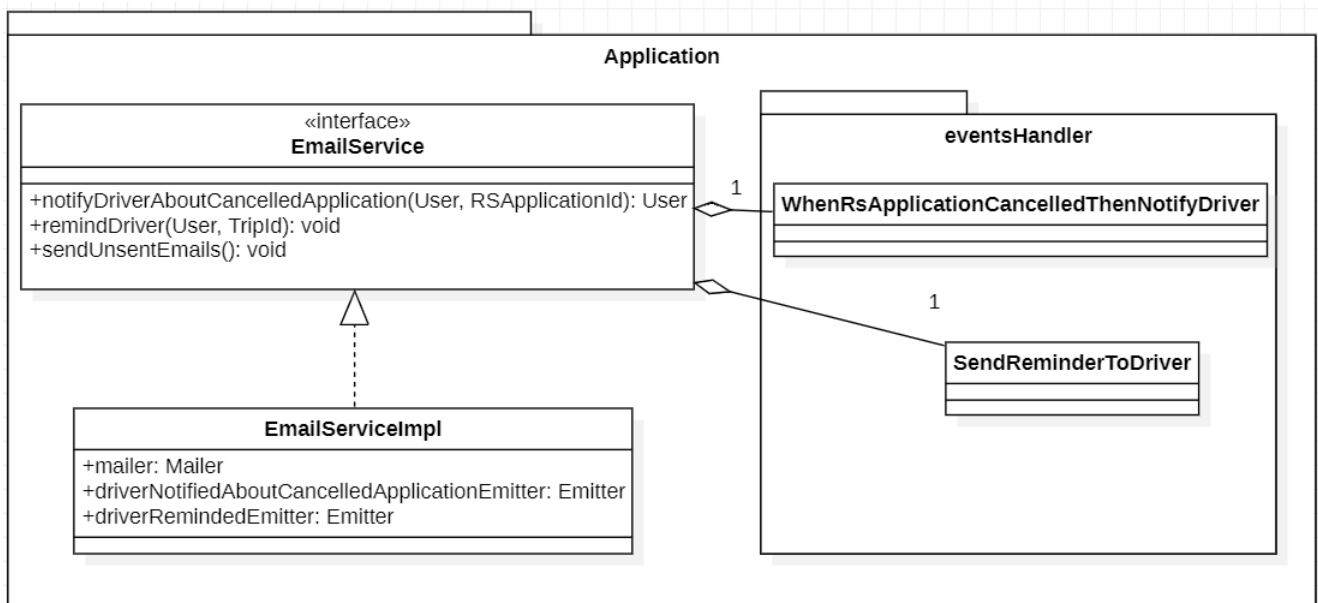


FIGURE 2.23 – Les services du microservice "Notifications"

Comme le montre la figure 2.23, ce microservice est responsable de la notification du conducteur. Il propose des methodes comme remindDriver pour rappeler le conducteur avant quelques minutes du voyage ou notifyDriverAboutCancelledApplication qui le notifie lorsqu’une application est annulée par un passager. Les classes du paquetage **eventsHandler** sont des gestionnaires d’événements qui implémentent l’interface **EventHandler** du paquetage ddd.

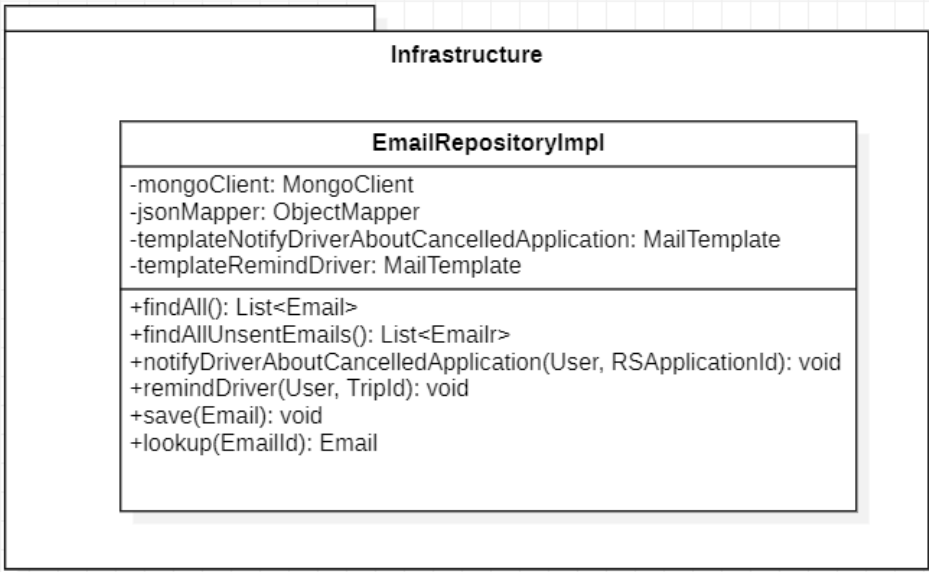


FIGURE 2.24 – L’infrastructure du microservice "Notifications"

La figure 2.24 représente l’infrastructure du microservice "Notifications". La classe **EmailRepositoryImpl** est une implémentation de l’interface **EmailRepository** du doamine qui permet assure la connexion et la manipulation des données dans la base de données et qui stocke des modèles des emails à envoyer.

La figure suivante illustre le diagramme de classe complet de ce microservice tout en respectant la conception et l’approche choisie dans la figure 2.7

FIGURE 2.25 – Diagramme de classe du microservice "Notifications"

Conclusion

Dans ce chapitre, nous avons présenté l'architecture et les approches que nous avons suivies pour développer l'application, ainsi que la conception générale et la conception détaillée de chaque microservice, en fournissant les figures, diagrammes et les explications nécessaires.

Chapitre 3

Réalisation de l'application

Introduction

Maintenant que nous avons présenté les choix architecturaux et la conception détaillée de notre application, nous présentons dans ce chapitre les différentes technologies que nous avons utilisées pour développer l'application. Nous fournirons également des captures d'écran montrant les différentes interfaces et fonctionnalités disponibles pour les utilisateurs.

3.1 Technologies utilisées

Dans cette section, nous décrivons les différents outils et techniques utilisés ainsi que les raisons de nos choix pour la réalisation de ce projet.

3.1.1 Environnement de développement

JAVA

C'est un langage de programmation populaire, robuste, sécurisé et performant que nous maîtrisons très bien, utilisé dans de nombreuses applications d'entreprise et industrielles.

Java est souvent utilisé pour le développement d'applications basées sur une architecture de microservices et DDD parce que c'est un langage de programmation orienté objet, ce qui le rend bien adapté pour la conception de services indépendants et modulaires. De plus, Java est multiplateforme, ce qui signifie que les applications peuvent être déployées sur différentes plateformes, notamment sur des conteneurs Docker, des environnements cloud et des serveurs et bien sur sa gestion efficace de la mémoire et des temps d'exécution rapides, ce qui est important pour les applications en microservices. [6]

Visual Studio Code (VS Code)

C'est un environnement de développement intégré (IDE) open-source léger, flexible et puissant qui offre de nombreuses fonctionnalités et extensions pour le développement d'applications et propose une multitude d'extensions pour les différents langages de programmation tels que Java, Python, C#, etc.

Il est particulièrement bien adapté au développement d'applications avec une architecture microservices et DDD (Domain-Driven Design) pour les raisons suivantes :

- Intégration de Git : VS Code dispose d'une intégration Git native qui facilite la gestion des différents référentiels de code source de l'application. Ceci est important pour les architectures

microservices car chaque service est généralement développé et déployé indépendamment, ce qui nécessite une gestion efficace de plusieurs référentiels de code source.

- Intégration des outils de développement : VS Code s'intègre facilement à de nombreux outils de développement populaires tels que Docker, Kubernetes, etc. Cette intégration est importante pour les architectures microservices car ces outils sont généralement utilisés pour la gestion des conteneurs, le déploiement, la configuration et l'orchestration .

- Débogage : VS Code dispose d'un puissant débogueur intégré qui facilite le processus de débogage des différents microservices de notre application. [11]

MongoDB

Dans une architecture microservice, chaque microservice peut utiliser une technologie de base de données différente des autres, ce qui lui permet de choisir la technologie qui répond le mieux à ses besoins spécifiques.

Pour minimiser la complexité de la gestion et de la synchronisation des données, nous avons choisi d'utiliser MongoDB pour tous les microservices.

MongoDB est une base de données NoSQL populaire qui est conçue pour stocker des données sous forme de documents JSON. Elle est utilisée pour les applications web modernes, les applications mobiles et les analyses de données en raison de sa flexibilité, de sa performance et de sa capacité à s'adapter à des schémas de données en constante évolution. [4]

MongoDB est souvent utilisé pour des applications basées sur des architectures microservices pour plusieurs raisons :

- Flexibilité : MongoDB est une base de données NoSQL, ce qui signifie qu'elle offre une grande flexibilité pour la modélisation des données. Cette flexibilité est particulièrement utile dans les architectures de microservices, où chaque service peut avoir ses propres exigences en matière de stockage de données.

- Haute disponibilité : MongoDB prend en charge la réplication automatique et la récupération en cas des erreurs, ce qui garantit une haute disponibilité des données.

- Performance : MongoDB est conçu pour offrir une haute performance avec des temps de réponse rapides aux requêtes de lecture et d'écriture. [3]

Docker

La conteneurisation offre plusieurs avantages pour le développement et le déploiement d'applications, tels que la portabilité, l'isolation, la facilité de déploiement et de mise à l'échelle, et la gestion des versions simplifiées. Pour profiter de ces avantages, nous avons utilisé Docker. [9]

Docker est un système de conteneurisation open source qui permet la création, le déploiement et la gestion de conteneurs logiciels afin de permettre aux développeurs d'empaqueter, de distribuer et d'exécuter des applications dans des conteneurs légers et portables.[15]

3.1.2 Bibliothèques et frameworks utilisés

Svelte

Svelte est un framework front-end moderne qui utilise une approche de compilation pour créer des applications web de haute performance. Contrairement à d'autres frameworks JavaScript tels que React ou Vue, Svelte compile son code en JavaScript au fur et à mesure qu'il est écrit, plutôt qu'au moment de l'exécution. Cette approche rend les applications plus légères, plus rapides et plus fluides.[7]

En utilisant Svelte avec notre architecture de microservices, nous pouvons créer des interfaces utilisateur distinctes pour chaque microservice, réduisant ainsi la charge de travail du backend. Nous pouvons également créer des interfaces utilisateur réactives qui interagissent avec plusieurs microservices en même temps.

Quarkus

Quarkus est un framework open source pour le développement d'applications Java destinées aux environnements de conteneurs tels que Kubernetes et Docker. Il est conçu pour fournir des temps de démarrage rapides, une faible consommation de mémoire, une grande efficacité de développement et une expérience de développement cloud-native.

Dans les architectures basées sur les microservices et DDD, Quarkus met en œuvre une conception modulaire et une évolutivité efficace pour déployer des microservices de manière indépendante et les exécuter sur des plateformes cloud. Quarkus prend également en charge une variété de technologies populaires, notamment Kubernetes, Apache Kafka et les bases de données NoSQL.[8]

Apache Kafka

Apache Kafka est un système de messagerie et de streaming de données distribuées open source qui permet aux applications de publier, de stocker et de traiter des flux de données en temps réel. Kafka est conçu pour gérer de grands volumes de données et des charges de travail à haute performance, offrant une faible latence, une haute disponibilité et une grande évolutivité.[16]

Kafka est souvent utilisé dans les architectures de microservices pour gérer le flux de données entre les services. En utilisant Kafka comme système de messagerie, chaque microservice peut publier des messages dans un ou plusieurs topics Kafka, qui peuvent ensuite être consommés par d'autres services qui s'y abonnent. Cela permet aux services de communiquer de manière asynchrone sans avoir besoin de connaître l'existence ou la localisation de chaque service avec lequel ils interagissent.

Keycloak

Keycloak est une plateforme open source de gestion des identités et des accès (IAM) qui offre des fonctionnalités de sécurité pour les applications web et mobiles. Elle permet aux développeurs d'ajouter rapidement et efficacement des fonctionnalités d'authentification et d'autorisation à leurs applications. Keycloak prend en charge les protocoles d'authentification et d'autorisation les plus courants tels que OAuth2, OpenID Connect et SAML et peut être facilement intégré.

Keycloak est souvent utilisé dans les environnements microservices et les architectures basées sur le cloud pour fournir une couche de sécurité unifiée et centralisée. [13]

Swagger UI

Swagger UI est une interface utilisateur interactive pour tester et visualiser les API RESTful, qui décrit les end-points, les paramètres, les réponses et les modèles de données d'une API. Elle permet aux développeurs de tester rapidement et facilement diverses fonctionnalités de l'API, de visualiser les résultats de leurs requêtes et d'explorer la documentation de l'API.[14]

En utilisant Swagger dans l'architecture microservice, Nous pouvons tester chaque microservice individuellement, cela nous permet d'identifier rapidement les erreurs et les problèmes de performance dans chaque microservice.[1]

3.2 Réalisation des microservices

Dans cette section, nous utilisons l'interface Swagger (SwaggerUI) pour tester la fonctionnalité de chaque microservice et fournir quelques exemples de requêtes et de captures de l'interface de l'application.

3.2.1 Authentification avec Keycloak

Keycloak assure la sécurité de notre application et offre des fonctionnalités de gestion des utilisateurs et des rôles. Il permet de créer des utilisateurs et des rôles personnalisés, puis affecter chaque utilisateur à un ou plusieurs rôles en fonction de leurs besoins d'accès.

La figure 3.1 représente l'interface de création d'un nouveau utilisateur.

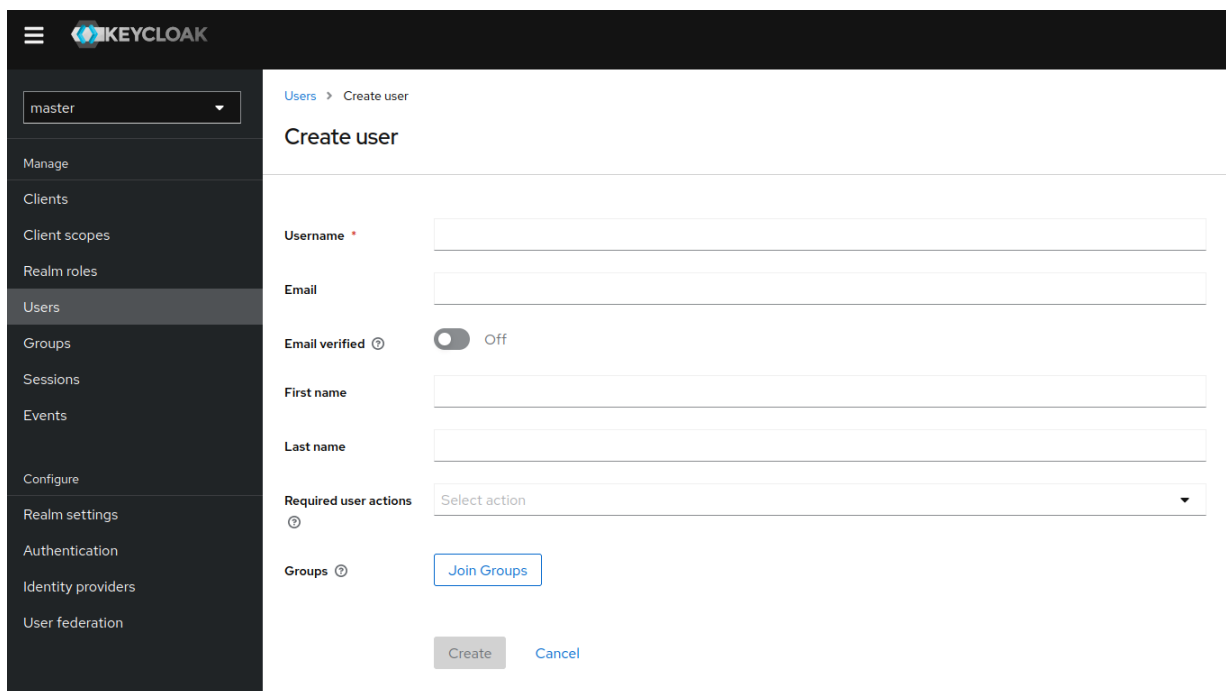
The image shows the Keycloak administration console interface for creating a new user. On the left is a dark sidebar with a menu containing 'master' (selected), 'Manage', 'Clients', 'Client scopes', 'Realm roles', 'Users' (highlighted), 'Groups', 'Sessions', 'Events', 'Configure', 'Realm settings', 'Authentication', 'Identity providers', and 'User federation'. The main content area has a breadcrumb 'Users > Create user' and the title 'Create user'. The form includes fields for 'Username *', 'Email', 'Email verified' (a toggle switch currently set to 'Off'), 'First name', 'Last name', 'Required user actions' (a dropdown menu showing 'Select action'), and 'Groups' (with a 'Join Groups' button). At the bottom are 'Create' and 'Cancel' buttons.

FIGURE 3.1 – Interface de création d'un utilisateur

La figure 3.2 est une capture de l'interface d'affectation des rôles à un utilisateur.

Assign roles to hazgui account



Filter by realm roles

Search by role name

→

1 - 6

<

>

<input type="checkbox"/>	Name	Description
<input type="checkbox"/>	default-roles-rs-platform	\${role_default-roles}
<input type="checkbox"/>	offline_access	\${role_offline-access}
<input type="checkbox"/>	rs-admin	Administrateur de la plateforme
<input type="checkbox"/>	rs-manager	manager ayant le droit de voir le reporting seulement
<input type="checkbox"/>	rs-user	utilisateur de la plateforme
<input type="checkbox"/>	uma_authorization	\${role_uma_authorization}

1 - 6

Assign

Cancel

FIGURE 3.2 – Interface d'affectation des rôles

La figure 3.3 illustre l'interface d'authentification de notre application.

The image shows a login interface for 'RS-PLATFORM'. It features a dark, geometric background. In the center, there is a white rectangular box with a blue border. Inside this box, the text 'Sign in to your account' is displayed. Below this text are two input fields: 'Username or email' and 'Password'. The 'Username or email' field is highlighted with a red border. At the bottom of the white box is a blue button labeled 'Sign In'.

FIGURE 3.3 – Interface d'authentification

Après l'authentification, l'utilisateur accède à notre application et sera dirigé vers la page d'accueil représentée dans la figure 3.4.

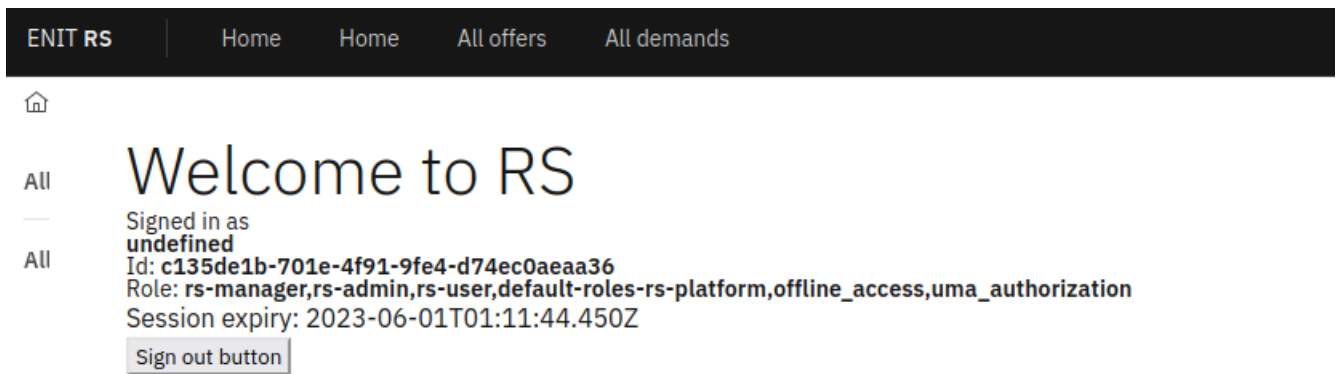


FIGURE 3.4 – Page d'accueil de l'application

3.2.2 Réalisation du microservice "Gestion des offres de covoiturage"

L'API du microservice "Gestion des offres de covoiturage" offre au conducteur plusieurs fonctionnalités tel que la création, la publication, l'annulation et le changement d'une offre. Voici quelques exemples de requêtes et des captures d'écran pour s'assurer que l'API fonctionne correctement et renvoie les résultats attendus.

La figure 3.5 est une requête de création d'une offre.

```
1 {
2   "departureAddress": {
3     "city": "string",
4     "street": "string",
5     "number": 0,
6     "location": "string"
7   },
8   "departureGeoPoint": {
9     "latitude": 0,
10    "longitude": 0
11  },
12  "destinationAddress": {
13    "city": "string",
14    "street": "string",
15    "number": 0,
16    "location": "string"
17  },
18  "destinationGeoPoint": {
19    "latitude": 0,
20    "longitude": 0
21  },
22  "availableSeatsNumber": {
23    "value": 0
24  },
25  "departureDateTime": "2022-03-10T12:15:50"
26 }
```

FIGURE 3.5 – Requête de création d'une offre

On peut vérifier dans la figure 3.6 que l'offre a été bien créer et que le conducteur peut la publier ou l'annuler.

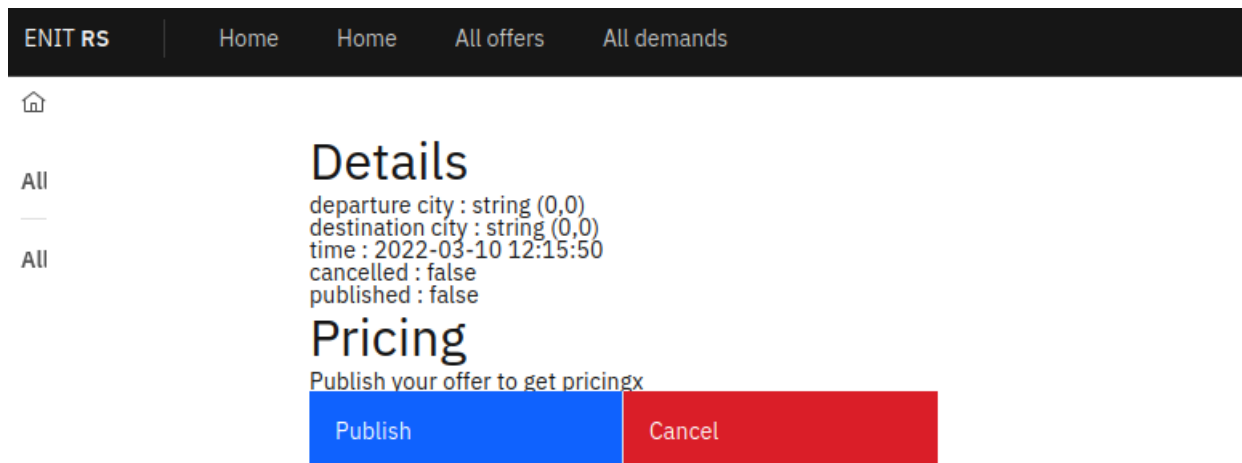


FIGURE 3.6 – Interface du microservice "Gestion des offres de covoiturage"

Pour publier ou annuler une offre, il suffit de mentionner son identifiant tel qu'il est présenté dans la figure 3.7

```
1 {  
2   "rsOfferId": {  
3     "value": "3fa85f64-5717-4562-b3fc-2c963f66afa6"  
4   }  
5 }
```

FIGURE 3.7 – Requête de publication ou d'annulation d'une offre

Par contre, pour la modification de l'offre, nous avons besoin de son identifiant et du nouveau attribut à changer. La figure 3.9 nous montre la Requête de changement du nombre de places disponibles d'une offre.

```
1 {  
2   "rsOfferId": {  
3     "value": "3fa85f64-5717-4562-b3fc-2c963f66afa6"  
4   },  
5   "availableSeatsNumber": {  
6     "value": 0  
7   }  
8 }
```

FIGURE 3.8 – Requête de changement du nombre de places disponibles d'une offre

3.2.3 Réalisation du microservice "Tarification"

Le microservice "Tarification" est responsable de calculer le prix de l'offre et il exécute cette tâche automatiquement dès qu'un événement de publication ou de modification d'une offre se produit dans le microservice "Gestion des offres de covoiturage" comme mentionné dans la figure 3.9, le prix de l'offre est fixé après sa publication.

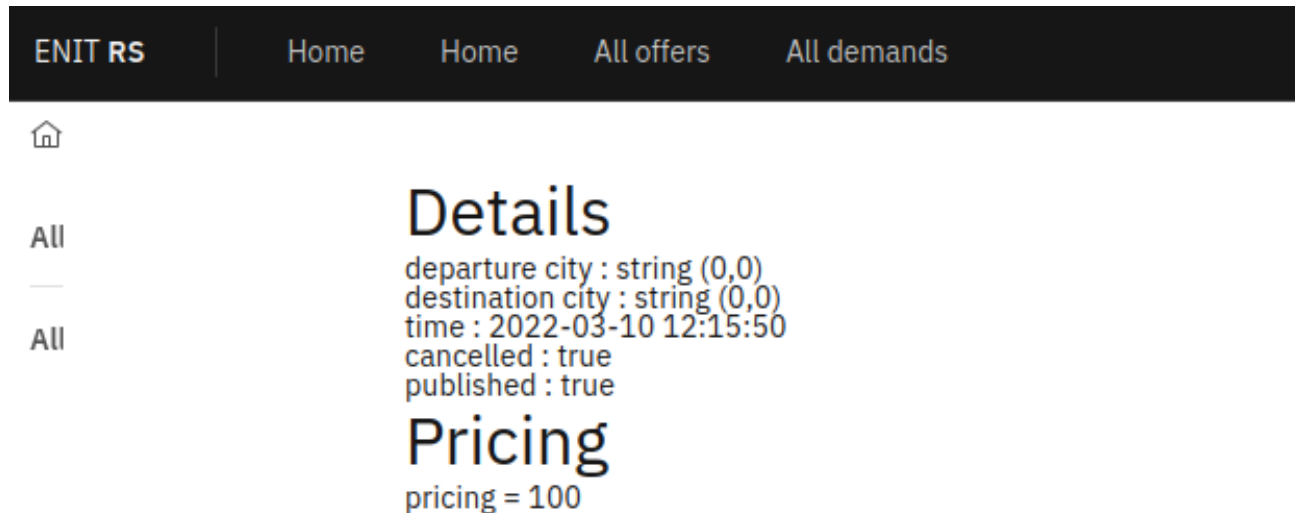


FIGURE 3.9 – Offre tarifiée par le microservice "Tarification"

Le document 3.11 représente une requête pour fixer le prix d'une offre qui contient le prix calculé et l'identifiant de l'offre.

```
1 {  
2   "price": 87.69642291259643 ,  
3   "rsOfferId": {  
4     "value": "904ba787-8414-4920-be50-4695479a7861"  
5   }  
6 }
```

FIGURE 3.10 – Requête pour fixer le prix d'une offre

3.2.4 Réalisation du microservice "Reservations"

L'interface du microservice "Réservations" permet au conducteur de voir les voyages de ses offres et les applications en attente, puis décider de les accepter ou de les rejeter.

La figure 3.12 illustre un voyage récupéré par son identifiant.

```
1 {
2   "tripId": {
3     "value": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
4   },
5   "pendingApplications": [
6     {
7       "rsApplicationId": {
8         "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
9       },
10      "tripId": {
11        "value": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
12      },
13      "accepted": false
14    }
15  ],
16  "acceptedApplications": [
17    {
18      "rsApplicationId": {
19        "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
20      },
21      "tripId": {
22        "value": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
23      },
24      "accepted": true
25    }
26  ],
27  "rejectedApplications": [
28    {
29      "rsApplicationId": {
30        "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
31      },
32      "tripId": {
33        "value": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
34      },
35      "accepted": false
36    }
37  ],
38  "tripDepartureTime": "2022-03-10T12:15:50",
39  "cancelled": false,
40  "passengerNotified": true,
41  "driverNotified": true
42 }
43 ]
```

FIGURE 3.11 – Voyage récupéré

Pour accepter ou refuser une application à un voyage, il faut mentionner l'identifiant du voyage et l'identifiant de l'application comme le montre la figure 3.13

```

1 {
2   "tripId": {
3     "value": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
4   },
5   "applicationId": {
6     "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
7   }
8 }

```

FIGURE 3.12 – Requête d'acceptation d'une application

3.2.5 Réalisation du microservice "Notifications"

Le microservice "Notifications" est chargé d'envoyer des notifications au conducteur. Les notifications sont automatisées et envoyées automatiquement lorsqu'un événement d'annulation d'une demande ou l'arrivée de l'heure de départ d'un voyage se produit à partir du microservice "Réservations".

La figure 3.14 illustre une notification récupérée à partir de son identifiant.

```

1 {
2   "emailId": {
3     "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
4   },
5   "user": {
6     "userId": {
7       "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
8     },
9     "nom": "string",
10    "prenom": "string",
11    "emailAddress": "string"
12  },
13  "object": "string",
14  "body": "string",
15  "sent": true
16 }

```

FIGURE 3.13 – Notification récupérée

Pour notifier le conducteur d'une application annulée, nous avons besoin de l'identifiant du conducteur et de l'identifiant de l'application comme indiqué dans la figure 3.15


```
1 {
2   "driver": {
3     "userId": {
4       "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
5     },
6     "nom": "string",
7     "prenom": "string",
8     "emailAddress": "string"
9   },
10  "rsApplicationId": {
11    "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
12  }
13 }
```

FIGURE 3.14 – Requête pour notifier le conducteur de l’annulation d’une application

Conclusion

Ce chapitre a permis de donner un aperçu de la mise en œuvre de notre application en présentant différentes captures pour illustrer son interface utilisateur et ses différentes fonctionnalités, ainsi que les technologies utilisées pour la réaliser.

Conclusion générale

Dans le cadre de ce projet, nous avons développé une application de covoiturage basée sur des microservices qui est très flexible et adaptable, ce qui nous permettra de répondre rapidement aux changements et de développer de nouvelles fonctionnalités à l'avenir.

Ce rapport contient les différentes étapes de la mise en œuvre de l'application, en commençant par la spécification des besoins, puis l'architecture choisie, la conception générale et la conception détaillée, jusqu'aux technologies utilisées et la réalisation.

Ce projet a été une très bonne expérience et nous a permis d'apprendre beaucoup de choses, notamment nous avons amélioré la communication, la collaboration et le travail d'équipe. En outre, nous avons utilisé plusieurs outils et techniques qui nous ont permis de mettre en œuvre des compétences déjà acquises et d'en développer de nouvelles.

Ce projet a également mis l'accent sur l'importance d'appliquer de bonnes pratiques d'ingénierie logicielle pour garantir la qualité, la fiabilité et la sécurité. Grâce à des méthodes telles que la conception axée sur le domaine d'application, l'analyse des événements et l'architecture des microservices, l'application a été développée de manière structurée, efficace et cohérente avec les meilleures pratiques.

Références

- [1] Ahmad ALSHAMMARI, Abdulrahman ALQAHTANI et Meshari ALDOSSARY. « Microservices Architecture: Overview, Benefits, and Best Practices ». In : *Journal of Information Technology Research* 14.2 (2021), p. 1-22. DOI : 10.4018/JITR.20210401.0a4.
- [2] Alberto BRANDOLINI. « Introducing event storming ». In : *blog, Ziobrando's Lair* 18 (2013).
- [3] Kristina CHODOROW. « MongoDB: The Definitive Guide ». In : (2013). URL : <https://www.oreilly.com/library/view/mongodb-the-definitive/9781449344795/>.
- [4] Kristina CHODOROW et Shannon DIROLF. « MongoDB: The Definitive Guide ». In : *O'Reilly Media, Inc.* (2013). Consulté le 10 avril 2023. URL : <https://www.oreilly.com/library/view/mongodb-the-definitive/9781449344764/>.
- [5] Lorenzo DE LAURETIS. « From monolithic architecture to microservices architecture ». In : *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE. 2019, p. 93-96.
- [6] John DOE. « Why Java is Often Used for Microservices and DDD Applications ». In : *Medium* (2021). URL : <https://medium.com/>.
- [7] Anthony GORE. *Svelte 3 Up and Running: Build Reactive Web Apps in a Fraction of the Time*. O'Reilly Media, 2020.
- [8] Red HAT. *Quarkus: Supersonic Subatomic Java*. <https://quarkus.io/>. [Accessed: 10-April-2023]. 2021.
- [9] Red HAT. *Why Use Containerization?* Accessed: 2022-04-11. URL : <https://www.redhat.com/en/topics/containers/why-use-containerization> (visité le 11/04/2022).
- [10] Stefan KAPFERER et Olaf ZIMMERMANN. « Domain-specific Language and Tools for Strategic Domain-driven Design, Context Mapping and Bounded Context Modeling. » In : *MODELSWARD*. 2020, p. 299-306.
- [11] MICROSOFT. « Visual Studio Code ». In : (2021). Site web consulté le 10 avril 2023. URL : <https://code.visualstudio.com/>.
- [12] Jeffery PALERMO. « The Onion Architecture ». In : (2020). Site web consulté le 18 avril 2023. URL : <https://thecodereaper.com/2020/05/02/the-onion-architecture/>.
- [13] Stian PEACOCK. *Keycloak: Identity and Access Management for Modern Applications*. Packt Publishing Ltd, 2020.
- [14] Caio Ribeiro PEREIRA. *API Development with Node.js*. Packt Publishing Ltd, 2019.
- [15] Nigel POULTON. *Docker Deep Dive*. Nigel Poulton, 2018.
- [16] Gwen SHAPIRA, Neha NARKHEDE et Todd PALINO. *Kafka: The Definitive Guide*. O'Reilly Media, Inc., 2017.
- [17] Vaughn VERNON. *Implementing domain-driven design*. Addison-Wesley, 2013.

- [18] Vaughn VERNON. « Symbols/Icons for Event Storming ». In : (2022). Site web consulté le 17 avril 2023. URL : <https://kalele.io/symbolsicons-for-event-storming/>.