République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université Echahid Larbi Tébessi - Tébessa

Faculté des Sciences Exactes et Sciences de la Nature et de la Vie

Département de : Informatique

Filière : Informatique

Spécialité : Réseaux et Sécurité Informatique

Mémoire présenté et soutenu publiquement en vue de
l'obtention du master en Réseaux et Sécurité Informatique

Intitulé :

Mefria : Une application super multiservice

Présenté par : **Tarek Triki, Abdelmounam Kalla**

Membres du jury

- **Dr. Hassiba Laifa** ..... MCB ..... Présidente
- **Dr. Mounir Achouri** ..... MCB ..... Examinateur
- **Dr. Ayoub Yahiaoui** ..... MCB ..... Encadrant
- **Dr. Abderrazak Khediri** ..... MCA ..... Co-encadrant

Année Universitaire : 2024 - 2025

Résumé

Mefria est une application mobile multiservices innovante conçue pour simplifier le quotidien des utilisateurs en centralisant plusieurs services essentiels au sein d'une seule plateforme. L'application propose initialement la prise de rendez-vous pour les coiffeurs et les coiffeuse, et prévoit d'étendre ses fonctionnalités vers la livraison de repas, les courses alimentaires, le transport de personnes, ainsi que la livraison de colis.

Mefria vise à répondre aux besoins croissants des citoyens en matière de praticité, de gain de temps et d'efficacité, tout en soutenant la digitalisation des petits commerçants, artisans et prestataires de services. L'application se distingue par son interface conviviale, son système de réservation intuitif, ainsi que son approche locale et adaptée au contexte algérien.

Ce mémoire explore les aspects techniques, ergonomiques et économiques de l'application, tout en analysant son impact potentiel sur les habitudes de consommation et la transformation numérique des services de proximité.

Abstract

Mefria is an innovative multi-service mobile application designed to simplify users' daily lives by centralizing several essential services within a single platform. The application initially offers appointment scheduling for hairdressers and plans to expand its capabilities to include meal delivery, grocery shopping, passenger transportation, and parcel delivery.

Mefria aims to meet citizens' growing needs for convenience, time savings, and efficiency, while supporting the digitalization of small businesses, artisans, and service providers. The application stands out for its user-friendly interface, intuitive booking system, and local approach, adapted to the Algerian context.

This thesis explores the technical, ergonomic, and economic aspects of the application, while analyzing its potential impact on consumer habits and the digital transformation of local services.

ملخص

مِفرية تطبيق هاتف مبتكر متعدد الخدمات، مصمم لتبسيط الحياة اليومية للمستخدمين من خلال تجميع العديد من الخدمات الأساسية ضمن منصة واحدة. يقدم التطبيق في البداية خدمة جدولة مواعيد مصففي الشعر، ويخطط لتوسيع نطاقه ليشمل توصيل الوجبات، وشراء البقالة، ونقل الركاب، وتوصيل الطرود.

يهدف مِفرية إلى تلبية احتياجات المواطنين المتزايدة من الراحة، وتوفير الوقت، والكفاءة، مع دعم رقمنة الشركات الصغيرة، والحرفيين، ومقدمي الخدمات. يتميز التطبيق بواجهته سهلة الاستخدام، ونظام الحجز البديهي، ونهجه المحلي، المُكيّف مع السياق الجزائري.

تستكشف هذه الأطروحة الجوانب التقنية، وبيئة العمل، والاقتصادية للتطبيق، مع تحليل تأثيره المحتمل على عادات المستهلكين والتحول الرقمي للخدمات المحلية.

Remerciements

Nous remercions tout d'abord Allah Tout-Puissant, qui nous a donné la force, la patience et la volonté pour mener à bien ce travail.

Nous tenons à exprimer notre profonde gratitude à nos encadrants, Dr.Ayoub Yahiaoui et Dr.Abderrazak Khediri, pour leurs précieux conseils, leur disponibilité et leur soutien continu tout au long de ce projet.

Nos plus sincères remerciements s'adressent aussi aux membres du jury pour avoir accepté d'évaluer notre travail, pour leurs remarques constructives et pour le temps qu'ils nous ont consacré.

Nous remercions également l'ensemble des enseignants et du personnel de université Echahid Cheikh Larbi Tebessi , pour la qualité de l'enseignement et leur soutien durant notre formation.

Nos remerciements les plus sincères vont à l'ensemble des membres du laboratoire LAMIS, avec une mention particulière à son directeur, le Professeur Hakim Bendjenna, pour leur accueil, leur accompagnement et leur appui tout au long de notre carriere académique.

Nous tenons également à remercier chaleureusement la société SOMIPHOS pour nous avoir accueillis dans le cadre de notre stage, ainsi que pour les conditions favorables, les conseils professionnels et les connaissances pratiques qu'elle nous a offertes.

Enfin, nous tenons à remercier tous ceux qui ont contribué, de près ou de loin, à la réalisation de ce mémoire et à l'avancement de notre application Mefria.

Dédicace

Nous dédions ce mémoire :

À nos parents, pour leur amour inestimable, leurs sacrifices, leurs encouragements constants et leur confiance en nous.

À nos familles, pour leur soutien moral tout au long de ce parcours académique.

À nos enseignants, pour la transmission de leur savoir et leur accompagnement durant toutes ces années.

À nos amis et collègues, pour leur présence, leur aide et les moments partagés.

À tous ceux qui, de près ou de loin, ont contribué à la réussite de ce projet.

Cette dédicace est le reflet de notre reconnaissance sincère.

Table des matières

1 présentation du projet **1**
- 1.1 Contexte . . . 1
- 1.2 Énoncé du problème . . . 2
- 1.3 Objectifs du projet . . . 2
  - 1.3.1 Objectif général . . . 2
  - 1.3.2 Objectifs spécifiques . . . 3
- 1.4 Approche de la solution . . . 3
  - 1.4.1 Analyse des besoins . . . 3
  - 1.4.2 Conception architecturale . . . 3
  - 1.4.3 Développement itératif . . . 3
  - 1.4.4 Tests et validation . . . 3
  - 1.4.5 Déploiement et maintenance . . . 4
- 1.5 Conclusion . . . 4

2 État de l'art **5**
- 2.1 Introduction . . . 5
- 2.2 Comparaison des applications mondiales . . . 5
- 2.3 Comparaison des applications algériennes . . . 5
- 2.4 Synthèse des comparaisons . . . 6
- 2.5 Apport de *Mefria* . . . 6
- 2.6 Conclusion . . . 7

3 Spécification des besoins et conception **8**
- 3.1 Présentation du projet et spécification des besoins . . . 8
  - 3.1.1 Présentation du projet . . . 8
  - 3.1.2 Spécification des besoins . . . 8
- 3.2 Conception du système . . . 9
  - 3.2.1 Présentation d'UML . . . 9
  - 3.2.2 Diagramme de cas d'utilisation du service des rendez-vous . . . 10
  - 3.2.3 Diagramme de cas d'utilisation du service de livraison . . . 29
  - 3.2.4 Diagramme de cas d'utilisation du service e-commerce . . . 41
  - 3.2.5 Diagramme de cas d'utilisation du service des Artisans . . . 53
  - 3.2.6 Diagrammes de classes des services du système . . . 70
  - 3.2.7 Conclusion . . . 73

4 Réalisation et implémentation de l'application Mefria **74**
- 4.1 Environnement de développement . . . 74
  - 4.1.1 Outils et technologies utilisés . . . 74
  - 4.1.2 Architecture logicielle adoptée . . . 75

4.2 Développement du frontend mobile 75
4.2.1 Interface utilisateur 75
4.2.2 Communication entre client et serveur 75
4.3 Interfaces graphiques de l'application 76
4.3.1 Interface d'accueil 76
4.3.2 Interface d'inscription 77
4.3.3 Interface de connexion 78
4.3.4 Interface principale (page d'accueil) 79
4.3.5 Menu utilisateur et sélection de la langue 80
4.3.6 Interface des informations personnelles 81
4.3.7 Historique des activités 81
4.3.8 Interface des boutiques de coiffure disponibles 83
4.3.9 Coiffeurs disponibles dans la boutique sélectionnée 84
4.3.10 Services disponibles chez le coiffeur sélectionné 85
4.4 Conclusion 86

5 Conclusions et travaux futurs **87**

References **88**

Table des figures

3.1 Diagramme de cas d'utilisation du service des rendez-vous pour le client . . . 10
3.2 Diagramme de séquence – Authentification . . . 12
3.3 Diagramme de séquence – Consultation de l'historique des rendez-vous côté client . . . 13
3.4 Diagramme de séquence – Recherche de magasin . . . 14
3.5 Diagramme de séquence – Ajouter un rendez-vous côté client . . . 15
3.6 Diagramme de séquence – Annuler un rendez-vous côté client . . . 16
3.7 Diagramme de séquence - Affichage des rendez-vous du jour . . . 17
3.8 Diagramme de séquence - Ajout d'un rendez-vous . . . 18
3.9 Diagramme de séquence - Annulation d'un rendez-vous côté agent . . . 19
3.10 Diagramme de séquence - Mise à jour des informations du compte . . . 20
3.11 Diagramme de séquence - Ajouter les dépenses . . . 21
3.12 Diagramme de séquence - Ajouter les promotions . . . 22
3.13 Diagramme de séquence - Modifier les services . . . 23
3.14 Diagramme de séquence - Afficher les revenus, les dépenses et les rendez-vous 24
3.15 Diagramme de séquence - Ajout d'un nouvel agent . . . 25
3.16 Diagramme de séquence - Supprimer un agent . . . 26
3.17 Diagramme de séquence - Modifier les informations du magasin . . . 27
3.18 Diagramme de séquence - Ajouter un magasin . . . 28
3.19 Diagramme de séquence - Supprimer un magasin . . . 29
3.20 Diagramme de cas d'utilisation du service de livraison . . . 30
3.21 Diagramme de séquence - Passer une commande . . . 31
3.22 Diagramme de séquence - Suivre une commande . . . 32
3.23 Diagramme de séquence - Annuler une commande . . . 33
3.24 Diagramme de séquence - Noter le magasin . . . 34
3.25 Diagramme de séquence - Noter le livreur . . . 35
3.26 Diagramme de séquence - Modifier une commande . . . 36
3.27 Diagramme de séquence - Accepter une commande . . . 37
3.28 Diagramme de séquence - Sauter une commande . . . 38
3.29 Diagramme de séquence - Modifier les informations d'un magasin . . . 39
3.30 Diagramme de séquence - Modifier les informations d'un produit . . . 40
3.31 Diagramme de séquence - Vérifier une commande . . . 41
3.32 Diagramme de cas d'utilisation du service e-commerce . . . 42
3.33 Diagramme de séquence - Passer une commande . . . 43
3.34 Diagramme de séquence - Suivre une commande . . . 44
3.35 Diagramme de séquence - Annuler une commande . . . 45
3.36 Diagramme de séquence - Noter le e-commerçant . . . 46
3.37 Diagramme de séquence - Notation d'un produit . . . 47
3.38 Diagramme de séquence - Modification d'une commande . . . 48

3.39 Diagramme de séquence - Modification des produits 49
3.40 Diagramme de séquence - Ajout d'un produit 50
3.41 Diagramme de séquence - Modification des informations du magasin 51
3.42 Diagramme de séquence - Confirmation d'une commande 52
3.43 Diagramme de séquence - Vérification d'une commande 53
3.44 Diagramme de cas d'utilisation du service Artisan 54
3.45 Diagramme de séquence - Affichage de l'historique des missions 55
3.46 Diagramme de séquence - Recherche d'un artisan 56
3.47 Diagramme de séquence - Demande de mission 57
3.48 Diagramme de séquence - Annulation d'une mission par le client 58
3.49 Diagramme de séquence - Affichage des missions du jour 59
3.50 Diagramme de séquence - Ajout d'une mission par l'artisan 60
3.51 Diagramme de séquence - Annulation d'une mission par l'artisan 61
3.52 Diagramme de séquence - Modification des informations du compte 62
3.53 Diagramme de séquence - Ajout de dépenses 63
3.54 Diagramme de séquence - Consultation des statistiques financières 64
3.55 Diagramme de séquence - Modification d'un service 65
3.56 Diagramme de séquence - Ajout d'un nouveau service 66
3.57 Diagramme de séquence - Suppression d'un service 67
3.58 Diagramme de séquence - Ajout d'un nouvel artisan 68
3.59 Diagramme de séquence - Suppression d'un artisan 69
3.60 Diagramme de classes - Module Rendez-vous 70
3.61 Diagramme de classes - Module Livraison 71
3.62 Diagramme de classes - Module E-commerce 72
3.63 Diagramme de classes - Module Artisans 73

4.1 Interface d'accueil de l'application Mefria 76
4.2 Interface d'inscription de l'application Mefria 77
4.3 Interface de connexion de l'application Mefria 78
4.4 Interface principale avec les catégories de services 79
4.5 Menu utilisateur avec options de gestion du compte et de langue 80
4.6 Interface des informations personnelles de l'utilisateur 81
4.7 Interface de l'historique des activités 82
4.8 Interface des boutiques de coiffure à proximité 83
4.9 Interface des coiffeurs disponibles dans la boutique sélectionnée 84
4.10 Interface des services disponibles chez le coiffeur sélectionné 85

Liste des tableaux

2.1 Comparatif des applications/services existants en Algérie 6

Chapitre 1

présentation du projet

Dans un monde de plus en plus connecté, où les besoins des citoyens évoluent rapidement et où la mobilité numérique devient une nécessité, la centralisation des services au sein d'une seule et unique plateforme numérique s'impose comme une réponse logique et moderne. C'est dans cette dynamique que s'inscrit le concept de la «super application», une application mobile multifonctionnelle capable d'intégrer plusieurs services dans une même interface conviviale et unifiée.

Alors que de grandes puissances technologiques comme la Chine, l'Indonésie ou l'Inde ont déjà largement adopté ce modèle à travers des applications emblématiques telles que WeChat[1], Grab [2] ou Gojek [3], le marché algérien reste encore peu exploré dans ce domaine. Les utilisateurs sont souvent contraints d'installer plusieurs applications pour répondre à leurs besoins quotidiens : transport, livraison, réservation, commerce, services à domicile, etc.

Face à ce constat, notre projet — intitulé Mefria — vise à concevoir et développer une super application multiservice adaptée au contexte local, en intégrant une gamme variée de services essentiels pour les citoyens algériens. L'objectif est de proposer une solution innovante, pratique et sécurisée, capable d'accompagner la transformation numérique du pays, tout en s'inscrivant dans une vision plus large de ville intelligente (Smart City), où l'efficacité, l'accessibilité et l'interconnexion des services deviennent un enjeu majeur.

À travers ce mémoire, nous allons explorer les fondements théoriques des super applications, analyser les besoins fonctionnels et techniques du projet, concevoir une architecture logicielle robuste, puis détailler le processus de réalisation de l'application Mefria.

1.1 Contexte

L'évolution rapide des technologies de l'information et de la communication (TIC) a profondément modifié les habitudes de consommation des services au quotidien. L'avènement des smartphones et l'accessibilité croissante à Internet ont permis la démocratisation des applications mobiles dans divers domaines : transport, commerce, réservation, services de proximité, etc.

Dans un premier temps, ces applications se sont développées de manière isolée, chacune répondant à un besoin spécifique. Cependant, cette fragmentation a entraîné une surcharge pour l'utilisateur, contraint d'installer, gérer et naviguer entre plusieurs interfaces pour accomplir ses tâches courantes.

C'est dans ce contexte qu'est apparu le concept de la super application (Super App), introduit notamment en Asie avec des géants comme WeChat en Chine ou Grab en Asie du Sud-Est. Ces plateformes centralisent une diversité de services — messagerie, paiements,

transport, commande de nourriture, etc. — au sein d'une seule application mobile. Le succès de ces modèles a suscité un intérêt mondial croissant pour ce type d'approche intégrée.

En Algérie, bien que le marché numérique soit en pleine expansion, les solutions restent largement fragmentées. Quelques initiatives locales telles que Yassir ou Temtem se sont démarquées dans des secteurs précis (mobilité, livraison), mais aucune plateforme ne propose à ce jour une solution réellement unifiée regroupant plusieurs services essentiels.

Le projet Mefria s'inscrit dans cette lacune technologique. Il vise à offrir une alternative locale et innovante en proposant une super application multiservice adaptée aux spécificités culturelles, économiques et technologiques du contexte algérien. Ce mémoire explore les fondements de cette idée, ses implications techniques et sociales, et les étapes nécessaires pour sa concrétisation.

1.2 Énoncé du problème

Malgré l'essor du numérique en Algérie et la généralisation des smartphones, les utilisateurs font encore face à une expérience fragmentée dans l'accès aux services quotidiens. Pour répondre à leurs besoins de transport, de réservation, de livraison, d'achat ou de services à domicile, ils doivent souvent jongler entre plusieurs applications indépendantes, aux interfaces et logiques différentes, ce qui engendre :

- Une surcharge numérique liée à la multiplication des applications installées.
- Une perte de temps et d'efficacité dans la recherche et l'utilisation des services.
- Une faible interopérabilité entre les systèmes, rendant difficile la mutualisation des données et des fonctionnalités.
- Une méfiance quant à la sécurité et à la confidentialité des données personnelles, souvent dispersées entre plusieurs prestataires.
- Une limitation de l'inclusion numérique, notamment pour les utilisateurs peu familiarisés avec les outils numériques complexes.

Par ailleurs, le manque d'une solution intégrée limite la capacité de développement de services urbains intelligents, freine la transformation numérique du pays et empêche la mise en place de systèmes de gestion efficaces au niveau local (transport, commerce, artisanat, etc.).

Dans ce contexte, il devient nécessaire de proposer une plateforme unique, centralisée, ergonomique et sécurisée qui regroupe un ensemble de services essentiels dans une seule application mobile. Le projet Mefria répond précisément à cette problématique, en visant à concevoir une super application multiservices adaptée aux réalités et aux besoins du marché algérien.

1.3 Objectifs du projet

1.3.1 Objectif général

Ce projet a pour objectif principal de concevoir et de développer une application mobile appelée Mefria, une super application multiservices destinée à centraliser plusieurs services essentiels (transport, réservation, services à domicile, commerce, etc.) dans une seule plateforme numérique. L'application vise à offrir une solution innovante, accessible et sécurisée, adaptée au contexte technologique, économique et socioculturel algérien.

1.3.2 Objectifs spécifiques

Pour atteindre cet objectif général, les objectifs spécifiques suivants ont été définis :

— Étudier le concept de super application et analyser les modèles existants (WeChat, Gojek, Yassir, etc.).

— Identifier les besoins des utilisateurs algériens en matière de services numériques quotidiens.

— Spécifier les exigences fonctionnelles et non fonctionnelles de l'application Mefria.

— Concevoir une architecture logicielle modulaire et évolutive, permettant l'intégration de multiples services dans un environnement unifié.

— Développer une interface utilisateur simple, ergonomique et intuitive.

— Mettre en œuvre des fonctionnalités de sécurité pour garantir la confidentialité des données et la fiabilité des transactions.

— Tester, valider et déployer une version fonctionnelle de l'application.

1.4 Approche de la solution

Pour répondre efficacement aux besoins identifiés et atteindre les objectifs du projet Mefria, une approche méthodique et progressive a été adoptée. Cette approche repose sur plusieurs étapes clés visant à garantir la qualité, la robustesse et l'adaptabilité de la super application multiservices.

1.4.1 Analyse des besoins

La première étape consiste à collecter et analyser les besoins fonctionnels et non fonctionnels des utilisateurs potentiels, en tenant compte du contexte local algérien. Cette phase inclut des enquêtes, des entretiens et l'étude des applications existantes afin de définir un cahier des charges précis.

1.4.2 Conception architecturale

Sur la base des besoins recueillis, une architecture logicielle modulaire est conçue. Cette architecture favorise la scalabilité, la maintenabilité et la facilité d'intégration de nouveaux services. Elle prend également en compte les exigences de sécurité et de performance.

1.4.3 Développement itératif

Le développement de Mefria est réalisé selon une méthode itérative et incrémentale. Cela permet de développer progressivement les différents modules (transport, services à domicile, e-commerce, etc.) tout en intégrant les retours des utilisateurs pour améliorer continuellement la solution.

1.4.4 Tests et validation

Chaque module développé fait l'objet de tests fonctionnels et de tests de performance afin d'assurer la fiabilité et la qualité globale de l'application. Des tests utilisateurs sont également réalisés pour valider l'ergonomie et l'adéquation aux attentes.

1.4.5 Déploiement et maintenance

Enfin, la solution est déployée sur les plateformes mobiles ciblées avec un suivi régulier pour corriger les éventuels bugs, améliorer les fonctionnalités et adapter l'application aux évolutions des besoins et du marché.

1.5 Conclusion

Le premier chapitre de ce mémoire a permis d'introduire le concept de super application et de poser les fondations du projet Mefria. Dans un environnement numérique en pleine mutation, notamment en Algérie, il devient essentiel de proposer une solution centralisée, ergonomique et sécurisée pour faciliter l'accès aux services quotidiens des citoyens.

À travers une analyse du contexte mondial et local, nous avons mis en évidence la pertinence et l'importance d'une telle initiative dans un marché encore peu exploité. Le projet Mefria s'inscrit ainsi comme une réponse technologique et sociale à la fragmentation des services numériques, avec pour ambition de contribuer activement à la transformation digitale du pays.

Les prochaines sections de ce mémoire seront consacrées à l'étude approfondie des modèles existants, à la définition des besoins utilisateurs, à la conception technique de l'application ainsi qu'au processus de développement et de mise en œuvre de cette super application multiservices.

Chapitre 2

État de l'art

2.1 Introduction

Ce chapitre vise à analyser l'état actuel des applications multiservices, notamment celles proposant des services de réservation, de transport, de livraison et de commerce électronique. Cette étude nous permettra de mieux positionner notre application Mefria dans son contexte technologique et concurrentiel, à la fois au niveau mondial et national.

2.2 Comparaison des applications mondiales

Dans un premier temps, nous allons examiner des applications multiservices à renommée mondiale telles que **Grab**, **Gojek**, **Uber**, ou encore **Glovo**. Ces plateformes ont su s'imposer grâce à une offre centralisée de services variés, allant de la réservation de transport à la livraison de repas, en passant par des services à domicile.

Axes d'analyse

Nous analyserons les aspects suivants :

— **Structure organisationnelle** : modèle d'affaires, partenariats, gouvernance.
— **Technologies utilisées** : langages, frameworks, infrastructures cloud, etc.
— **Expérience utilisateur (UX/UI)** : ergonomie, design, simplicité de navigation.
— **Stratégies d'expansion** : modèles de croissance, localisation, adaptation aux marchés.

Objectifs de la comparaison

Cette comparaison a pour but :

— d'identifier les **bonnes pratiques internationales**,
— de dégager les **fonctionnalités devenues des standards** dans le domaine des super applications.

2.3 Comparaison des applications algériennes

Contrairement aux super applications mondiales qui intègrent plusieurs services dans une seule plateforme, le marché algérien reste fragmenté, avec des applications spécialisées dans des domaines précis. Par conséquent, la comparaison sera divisée par types de services, afin d'analyser les solutions disponibles dans chaque segment et d'identifier les lacunes.

Types de services	Applications / plateformes existantes	Observations
Réservation de rendez-vous	Rahty Doc, Gentlemen Salon	Solutions ciblées sur les salons, médecins ou soins esthétiques. Pas d'intégration multi-services.
Services à domicile (artisans)	Bricoram, Bricool	Portails limités aux petits travaux. Couverture géographique restreinte, pas d'application grand public bien diffusée.
Transport & logistique	Yassir, Heetch, Presto	Acteurs majeurs en milieu urbain. Spécialisation transport, mais pas de gestion unifiée avec d'autres services.
E-commerce	Ouedkniss	Ancienne plateforme de petites annonces. Pas de logistique intégrée ni de paiements in-app.
Paiement électronique	BaridiMob	Application de paiement via CCP. Peu intégrée aux apps de services, usage encore limité chez les commerçants.

TABLE 2.1 : Comparatif des applications/services existants en Algérie

Remarque : Cette structuration fragmentée montre clairement qu'aucune solution algérienne actuelle ne propose une super application unifiée. Cela confirme l'opportunité pour *Mefria* d'occuper ce vide en regroupant ces services dans une seule plateforme ergonomique et adaptée au contexte local.

2.4 Synthèse des comparaisons

À l'issue de ces deux analyses, on observe :

— une **maturité technologique avancée** des super applications mondiales, caractérisée par des stratégies cohérentes et des expériences utilisateurs bien pensées ;

— un **marché algérien dispersé**, où chaque application reste spécialisée dans une verticale sans interconnexion ni intégration.

Cette situation met en lumière de nombreuses **opportunités d'innovation** pour **Mefria**, en capitalisant sur les standards mondiaux tout en répondant aux **besoins spécifiques du marché algérien**.

2.5 Apport de *Mefria*

Mefria apporte une réponse concrète à la fragmentation des services numériques en Algérie.

Ses forces

— Une **plateforme unifiée** intégrant : réservation, artisans, transport, livraison et e-commerce.

— Une **interface locale et simplifiée**, adaptée à différents niveaux de digitalisation.

— Des **modes de paiement flexibles** : électronique et espèces.

— Une **accessibilité géographique élargie** couvrant plusieurs régions du pays.

Ses objectifs

Mefria vise à :

— **centraliser** les services quotidiens dans une seule application,

— **simplifier** l'expérience utilisateur,

— **valoriser** le numérique au niveau national... et pourquoi pas à l'international.

2.6 Conclusion

En conclusion, l'analyse comparative des super applications mondiales et des solutions algériennes a permis de mettre en évidence des écarts significatifs en matière d'intégration, d'expérience utilisateur, et de couverture fonctionnelle. Tandis que les plateformes internationales se distinguent par une offre unifiée et technologiquement avancée, le paysage numérique algérien reste fragmenté, avec des services isolés et peu interconnectés.

Cette fragmentation constitue à la fois une faiblesse du marché actuel et une opportunité stratégique pour une solution innovante comme **Mefria**, capable de centraliser les services essentiels du quotidien dans une seule application. Forte de son orientation locale, de son interface simplifiée, et de sa flexibilité de paiement, *Mefria* s'inscrit comme une réponse pertinente aux besoins non satisfaits du marché.

Ce chapitre a donc posé les fondements nécessaires pour comprendre l'environnement concurrentiel et les attentes des utilisateurs. Les constats dégagés serviront de base aux choix fonctionnels, techniques et stratégiques présentés dans les chapitres suivants.

Chapitre 3

Spécification des besoins et conception

Introduction

La réussite de toute application complexe repose avant tout sur une analyse précise et rigoureuse des besoins des utilisateurs et des contraintes techniques. Cette étape est fondamentale pour orienter la conception du système et assurer sa pertinence, sa qualité et son évolutivité.

Dans ce chapitre, nous présentons tout d'abord le projet Mefria ainsi que la spécification des besoins fonctionnels et non fonctionnels qui en découlent. Nous détaillons ensuite la démarche de conception adoptée, en nous appuyant sur les outils de modélisation UML pour formaliser la structure et le comportement du système. Enfin, nous abordons la construction de la base de données, élément clé pour la gestion des données au sein de l'application.

3.1 Présentation du projet et spécification des besoins

3.1.1 Présentation du projet

Mefria est une application super multiservices développée pour répondre aux besoins variés des utilisateurs dans le contexte algérien. Cette plateforme mobile vise à centraliser plusieurs services essentiels, tels que la réservation de transports, la commande de services à domicile, l'e-commerce local, et la gestion des paiements, en une seule interface unifiée.

L'objectif principal de Mefria est de simplifier la vie quotidienne des utilisateurs en leur offrant un accès rapide, facile et sécurisé à une diversité de services, tout en tenant compte des spécificités culturelles, économiques et technologiques du pays. En favorisant l'intégration de plusieurs services au sein d'une même application, Mefria ambitionne également de contribuer à la transformation digitale et au développement des villes intelligentes en Algérie.

3.1.2 Spécification des besoins

Pour garantir la pertinence et la qualité de l'application, une spécification rigoureuse des besoins est essentielle. Ces besoins sont classés en deux catégories principales : fonctionnels et non fonctionnels.

Besoins fonctionnels

Les besoins fonctionnels correspondent aux fonctionnalités que l'application doit impérativement offrir aux utilisateurs. Parmi les principaux besoins fonctionnels identifiés pour Mefria, on peut citer :

— La gestion des profils utilisateurs (inscription, authentification, gestion des informations personnelles).
— La réservation et le suivi des services de transport.
— La demande et la planification de services à domicile (coiffure, plomberie, nettoyage, etc.).
— La consultation et la commande de produits via la plateforme e-commerce.
— La gestion des paiements sécurisés et des transactions financières.
— La messagerie intégrée pour la communication entre utilisateurs et prestataires.
— La gestion des avis et des évaluations des services.

Besoins non fonctionnels

Les besoins non fonctionnels définissent les contraintes et qualités attendues du système, notamment en termes de performance, sécurité et ergonomie. Les besoins non fonctionnels essentiels pour Mefria comprennent :

— La sécurité des données personnelles et des transactions financières.
— La disponibilité et la fiabilité du service, avec un temps de réponse optimal.
— L'ergonomie et la simplicité d'utilisation pour assurer une bonne expérience utilisateur.
— La compatibilité multi-plateforme (Android, iOS, Desktop).
— La scalabilité pour permettre l'intégration future de nouveaux services.
— La confidentialité et la conformité aux réglementations en vigueur.

3.2 Conception du système

3.2.1 Présentation d'UML

Définition

Le langage UML (Unified Modeling Language) est un langage de modélisation standardisé utilisé pour représenter graphiquement la structure, le comportement et les interactions d'un système logiciel. Il facilite la communication entre les parties prenantes (développeurs, chefs de projet, clients) et permet une meilleure compréhension et documentation du système à développer.

Pourquoi utiliser UML ?

UML permet de :

— Visualiser l'architecture du système avant sa mise en œuvre.
— Identifier clairement les différents composants, leurs interactions et leurs responsabilités.
— Réduire les ambiguïtés dans les exigences et faciliter la maintenance du système.
— Favoriser la modularité et la réutilisabilité du code.

Types de diagrammes utilisés

Dans le cadre du projet Mefria, plusieurs diagrammes UML seront utilisés pour illustrer les différents aspects du système :

— Le diagramme de cas d'utilisation (Use Case) : pour représenter les interactions entre les utilisateurs et le système.

— Le diagramme de classes : pour modéliser la structure statique des objets métiers.

— Le diagramme de séquence : pour illustrer les échanges entre les composants lors d'un scénario particulier.

3.2.2 Diagramme de cas d'utilisation du service des rendez-vous

Le diagramme ci-dessous présente les différents cas d'utilisation du service de gestion des rendez-vous dans le système Mefria. Il met en évidence les interactions possibles entre les utilisateurs (clients, barbiers, administrateurs, ...) et le système, en précisant les fonctionnalités accessibles à chacun. Ce schéma permet de visualiser les rôles respectifs et les responsabilités liées à la prise, la gestion et l'organisation des rendez-vous :

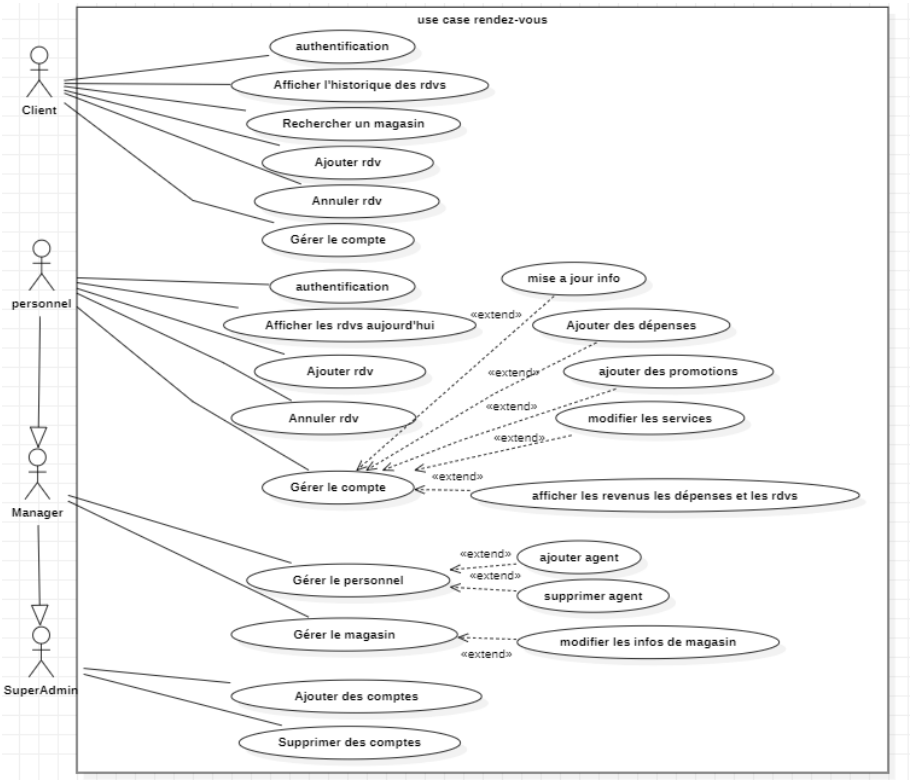


FIGURE 3.1 : Diagramme de cas d'utilisation du service des rendez-vous pour le client

Remarque : L'authentification est requise pour toutes les opérations du système, comme indiqué par la référence (`Authentification`) dans chaque diagramme de séquence.

Acteurs identifiés :

— **Client** : utilisateur final de l'application.

— **Personnel** : employé qui exécute les services (ex. : coiffeur , médecin ...).
— **Manager** : responsable de salon ou d'unité de service.
— **SuperAdmin** : administrateur général du système.

Principaux cas d'utilisation :

— Authentification (tous les rôles)
— Gestion de compte (modifier profil, mot de passe, etc.)
— Ajouter, annuler ou consulter des rendez-vous
— Consultation de l'historique de rendez-vous (client)
— Consultation et gestion des plannings (personnel et manager)
— Gestion de magasin (manager)
— Création et suppression de comptes utilisateurs (SuperAdmin)

Ce diagramme offre une vue d'ensemble des fonctionnalités clés du système Mefria, en illustrant les interactions entre les différents acteurs et les services fournis. Il constitue une base essentielle pour les diagrammes de séquence et de classes à venir.

Descriptions textuelles et diagrammes de séquence des cas d'utilisation

Chaque cas d'utilisation est décrit sous forme de processus en étapes, comme s'il était exécuté par un middleware entre le client et le système. Cette présentation met en valeur la logique de traitement et les vérifications automatiques effectuées par le système. Chaque description est accompagnée d'un diagramme de séquence illustrant l'enchaînement des interactions entre les acteurs et le système.

— **1. Authentification**

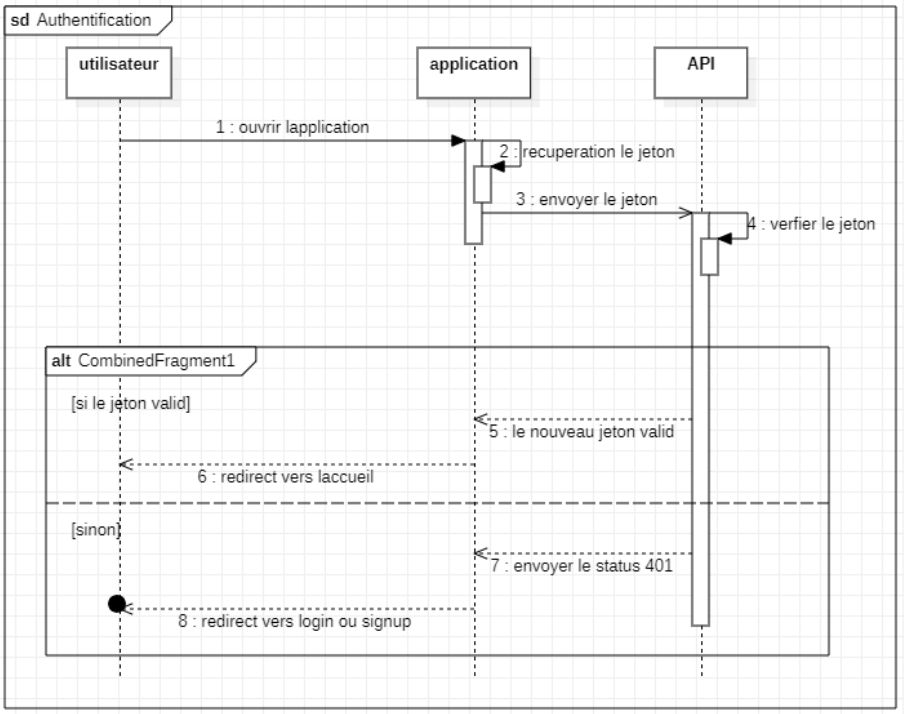


FIGURE 3.2 : Diagramme de séquence – Authentification

1. L'utilisateur ouvre l'application.
2. L'application tente de récupérer un jeton précédemment stocké (localStorage, SecureStore, etc.).
3. Ce jeton est transmis à l'API pour vérification.
4. L'API vérifie la validité du jeton (signature, expiration...).

— **Si le jeton est valide :**

5. L'API retourne un nouveau jeton ou valide l'ancien.
6. L'application redirige l'utilisateur vers la page d'accueil.

— **Sinon (jeton invalide ou expiré) :**

7. L'API renvoie un code d'erreur 401 (non autorisé).
8. L'application redirige l'utilisateur vers la page de connexion ou d'inscription.

— **2. Consultation de l'historique des rendez-vous**

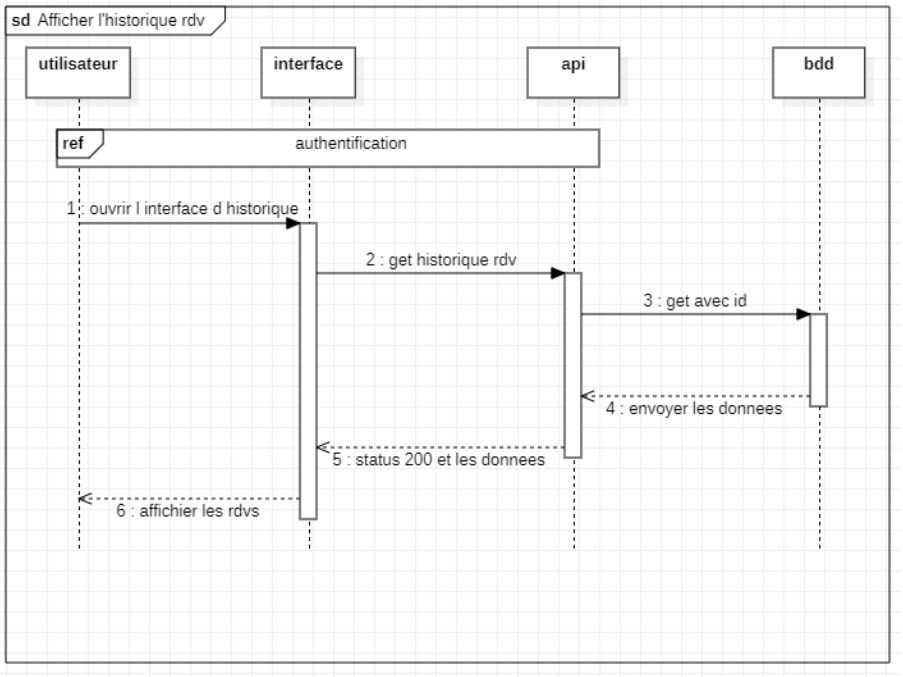


FIGURE 3.3 : Diagramme de séquence – Consultation de l'historique des rendez-vous côté client

1. L'utilisateur ouvre l'interface dédiée à l'historique des rendez-vous.
2. L'interface déclenche une requête vers l'API (`GET historique rdv`) en incluant le jeton d'authentification.
3. L'API interroge la base de données pour récupérer les rendez-vous de l'utilisateur identifié.
4. La base de données renvoie les données correspondantes à l'API.
5. L'API répond avec un code HTTP 200 ainsi que les données des rendez-vous.
6. L'interface affiche la liste des rendez-vous passés à l'utilisateur.

— **3. Recherche de magasin**

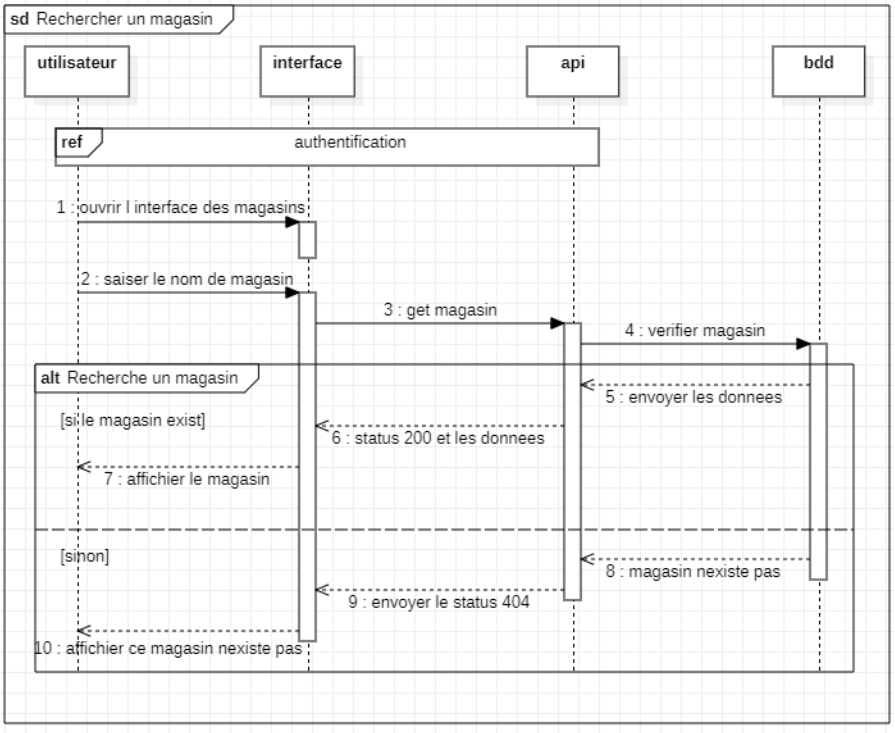


FIGURE 3.4 : Diagramme de séquence – Recherche de magasin

1. L'utilisateur ouvre l'interface de recherche de magasins.
2. Il saisit le nom du magasin recherché.
3. L'interface envoie une requête `GET magasin` à l'API.
4. L'API interroge la base de données pour vérifier l'existence du magasin.
5. La base de données renvoie les données associées si le magasin existe.

— **Si le magasin existe :**

6. L'API retourne un statut HTTP 200 avec les données du magasin.
7. L'interface affiche les informations du magasin à l'utilisateur.

— **Sinon (le magasin n'existe pas) :**

8. La base de données retourne une réponse indiquant que le magasin est introuvable.
9. L'API envoie un code d'erreur HTTP 404 à l'interface.
10. L'interface affiche un message indiquant que le magasin n'existe pas.

— **4. Ajouter un rendez-vous**

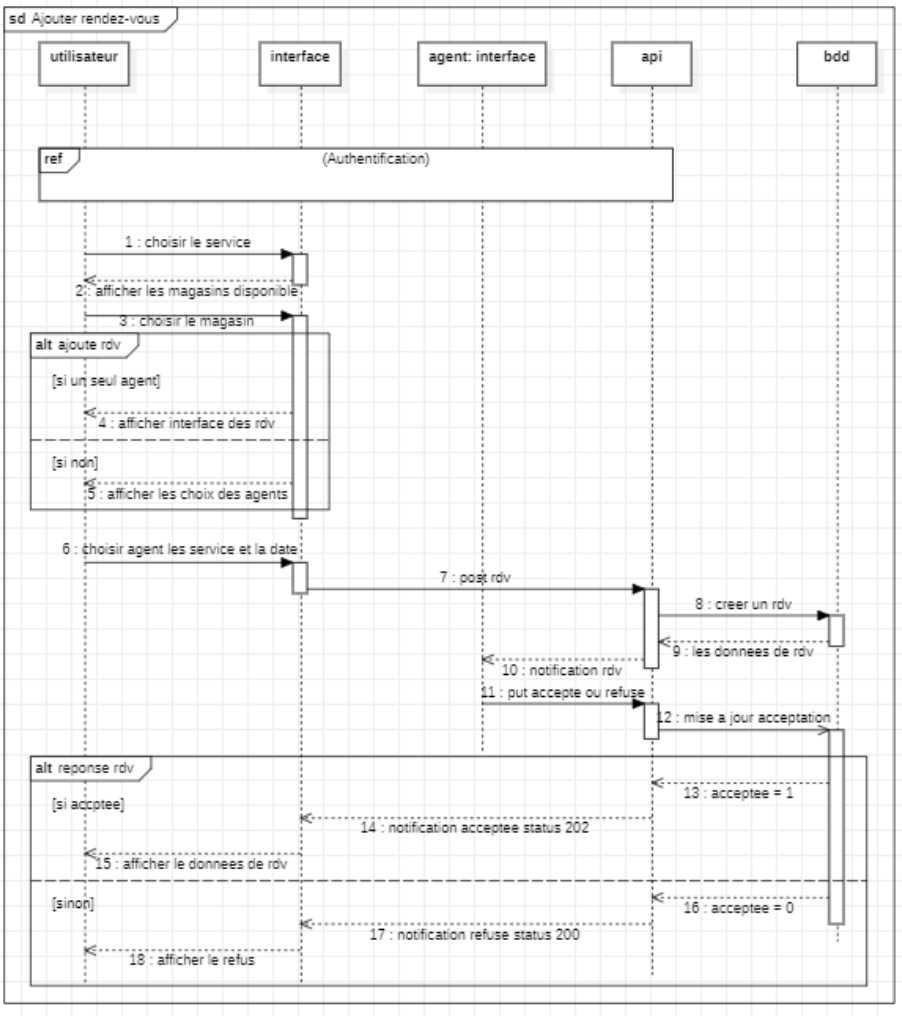


FIGURE 3.5 : Diagramme de séquence – Ajouter un rendez-vous côté client

1. L'utilisateur choisit le service souhaité via l'interface.
2. L'interface affiche les magasins disponibles.
3. L'utilisateur sélectionne un magasin.
   - **S'il n'y a qu'un seul agent associé :**
     4. L'interface affiche directement l'interface de rendez-vous.
   - **Sinon (plusieurs agents disponibles) :**
     5. L'interface affiche les choix des agents disponibles.
4. L'utilisateur choisit un agent, un service, et une date de rendez-vous.
5. L'interface envoie une requête `POST rendez-vous` à l'API.
6. L'API crée un nouveau rendez-vous.
7. L'API envoie les données du rendez-vous à la base de données.
8. La base de données enregistre les informations du rendez-vous.
9. L'API envoie une notification de création de rendez-vous.
10. L'agent peut accepter ou refuser la demande via l'interface agent.

11. L'API met à jour la base de données selon la réponse de l'agent.
12. La base de données retourne le statut du rendez-vous :

— **Si accepté :**

14. L'interface reçoit une notification de succès (status 202).
15. Les données du rendez-vous sont affichées à l'utilisateur.

— **Sinon (refusé) :**

17. L'interface reçoit une notification de refus (status 200).
18. Un message de refus est affiché à l'utilisateur.

— **5. Annuler un rendez-vous**

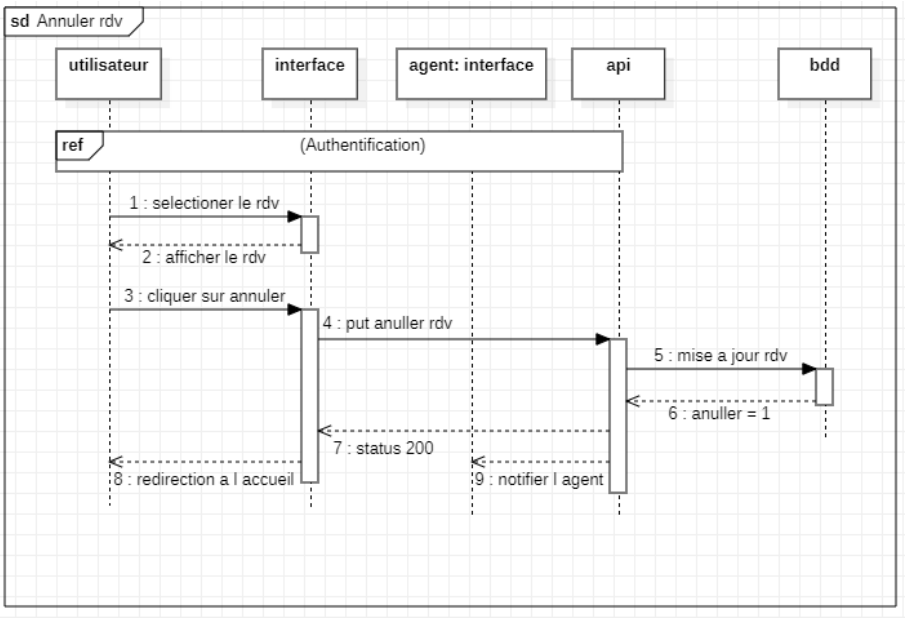


FIGURE 3.6 : Diagramme de séquence – Annuler un rendez-vous côté client

1. L'utilisateur sélectionne le rendez-vous à annuler.
2. L'interface affiche les détails du rendez-vous sélectionné.
3. L'utilisateur clique sur le bouton d'annulation.
4. L'interface envoie une requête `PUT annuler rdv` à l'API.
5. L'API met à jour le rendez-vous dans la base de données.
6. La base de données enregistre l'annulation (`annuler = 1`).
7. L'API retourne un statut HTTP 200 à l'interface.
8. L'interface redirige l'utilisateur vers la page d'accueil.
9. L'API notifie l'agent que le rendez-vous a été annulé.

— **6. Afficher les rendez-vous d'aujourd'hui**

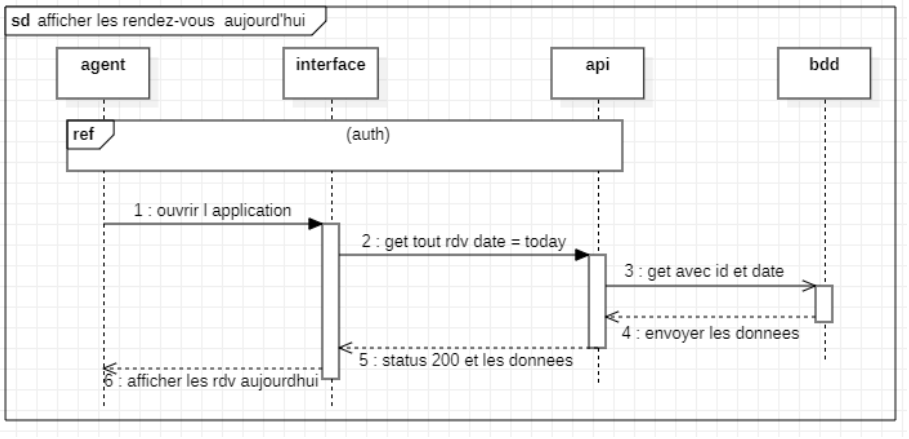


FIGURE 3.7 : Diagramme de séquence - Affichage des rendez-vous du jour

1. L'agent ouvre l'application.
2. L'interface envoie une requête à l'API pour récupérer tous les rendez-vous dont la date est égale à aujourd'hui.
3. L'API interroge la base de données en filtrant par identifiant et date du jour.
4. La base de données renvoie les données des rendez-vous.
5. L'API retourne un statut 200 avec les données à l'interface.
6. L'interface affiche les rendez-vous d'aujourd'hui à l'agent.

— **7. Ajouter un rendez-vous**

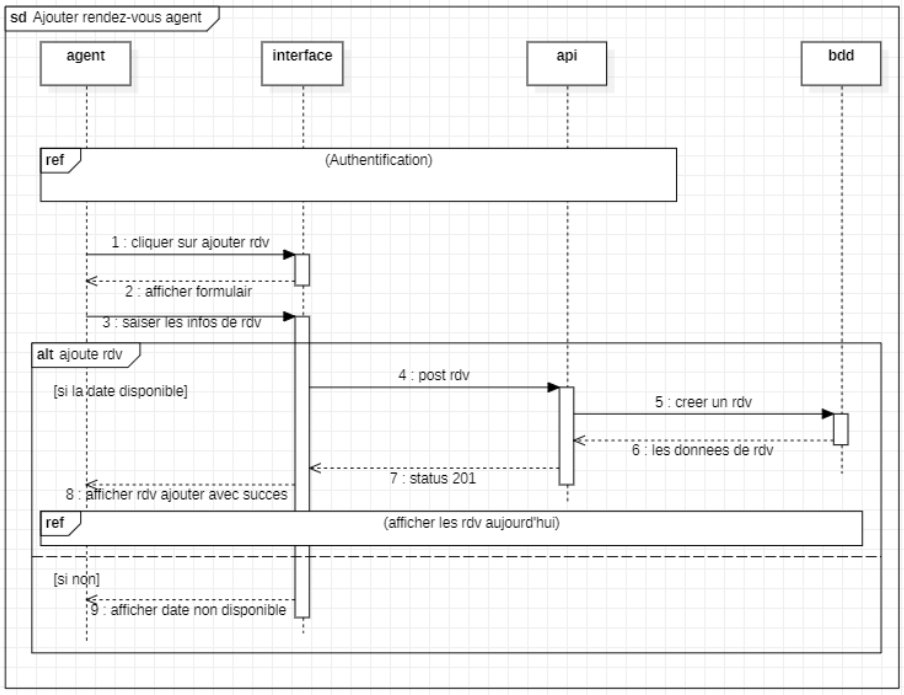


FIGURE 3.8 : Diagramme de séquence - Ajout d'un rendez-vous

1. L'agent clique sur le bouton "Ajouter rendez-vous".
2. L'interface affiche le formulaire de création de rendez-vous.
3. L'agent saisit les informations du rendez-vous (date, heure, client, etc.).

— **Si la date est disponible :**

4. L'interface envoie une requête `POST rendez-vous` à l'API avec les données.
5. L'API vérifie l'authentification de l'agent (ref Authentification).
6. L'API crée le rendez-vous dans la base de données.
7. La base de données retourne les données du rendez-vous créé.
8. L'API retourne un statut HTTP 201 (Created).
9. L'interface affiche un message de succès.
10. Le système actualise l'affichage des rendez-vous (ref Afficher les rdv aujourd'hui).

— **Sinon (date non disponible) :**

11. L'interface affiche un message "Date non disponible".

— **8. Annuler un rendez-vous**

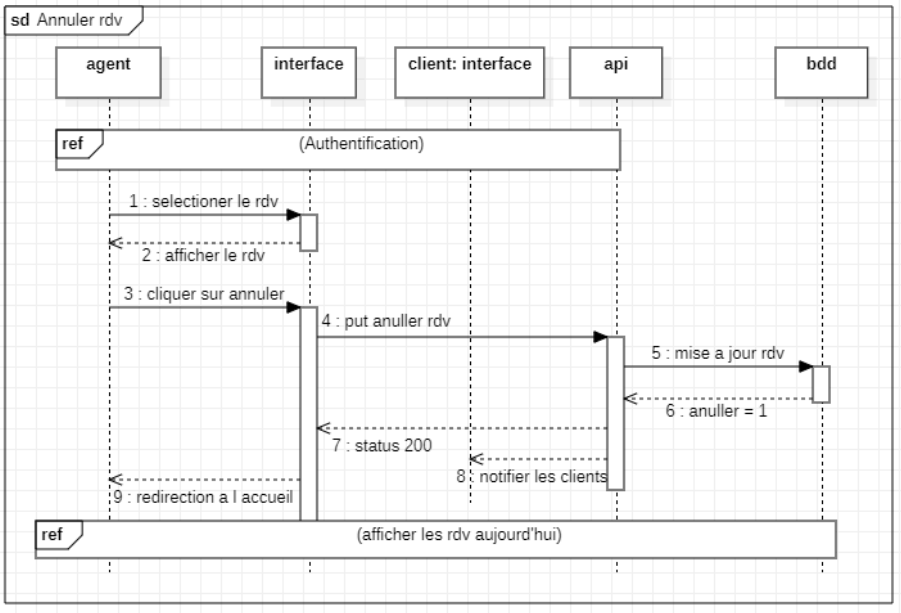


FIGURE 3.9 : Diagramme de séquence - Annulation d'un rendez-vous côté agent

1. L'agent sélectionne le rendez-vous à annuler.
2. L'interface affiche les détails du rendez-vous.
3. L'agent clique sur le bouton « Annuler ».
4. L'interface envoie une requête `PUT annuler rdv` à l'API.
5. L'API met à jour le rendez-vous dans la base de données.
6. La base de données marque le rendez-vous comme annulé (`annuler = 1`).
7. L'API retourne un statut HTTP 200 à l'interface.
8. L'API notifie les clients concernés de l'annulation.
9. L'interface redirige l'agent vers l'accueil.

— **09. Mettre à jour les informations du compte**

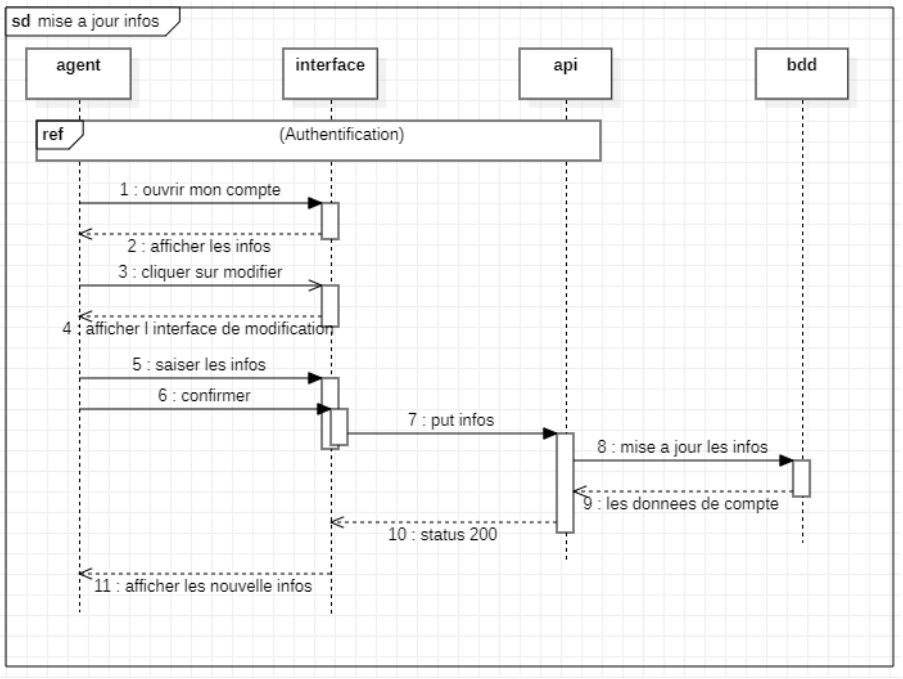


FIGURE 3.10 : Diagramme de séquence - Mise à jour des informations du compte

1. L'agent ouvre la section « Mon compte » dans l'interface.
2. L'interface affiche les informations actuelles du compte.
3. L'agent clique sur le bouton « Modifier ».
4. L'interface affiche un formulaire de modification.
5. L'agent saisit les nouvelles informations (ex. nom, téléphone, email...).
6. Il clique sur le bouton « Confirmer ».
7. L'interface envoie une requête `PUT infos` à l'API.
8. L'API met à jour les informations dans la base de données.
9. La base de données retourne les nouvelles données du compte.
10. L'API renvoie un statut HTTP 200 à l'interface.
11. L'interface affiche les informations mises à jour à l'agent.

— **10. Ajouter les dépenses**

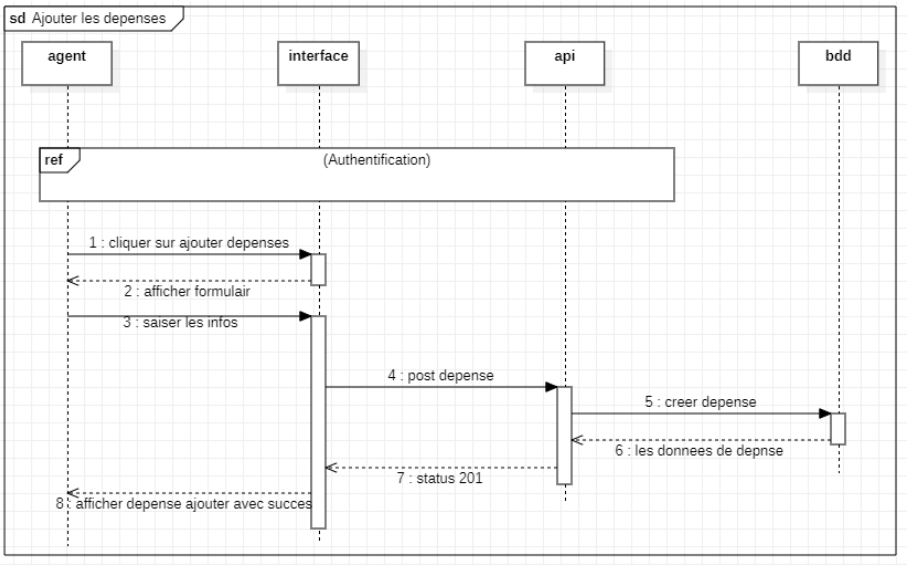


FIGURE 3.11 : Diagramme de séquence - Ajouter les dépenses

1. L'agent clique sur le bouton « Ajouter dépenses » dans l'interface.
2. L'interface affiche un formulaire de saisie.
3. L'agent saisit les informations liées à la dépense (montant, description, etc.).
4. L'interface envoie une requête `POST dépense` à l'API.
5. L'API transmet la requête à la base de données pour créer une nouvelle dépense.
6. La base de données retourne les données de la dépense créée.
7. L'API retourne un statut HTTP 201 à l'interface.
8. L'interface affiche un message de confirmation indiquant que la dépense a été ajoutée avec succès.

— 11. Ajouter les promotions

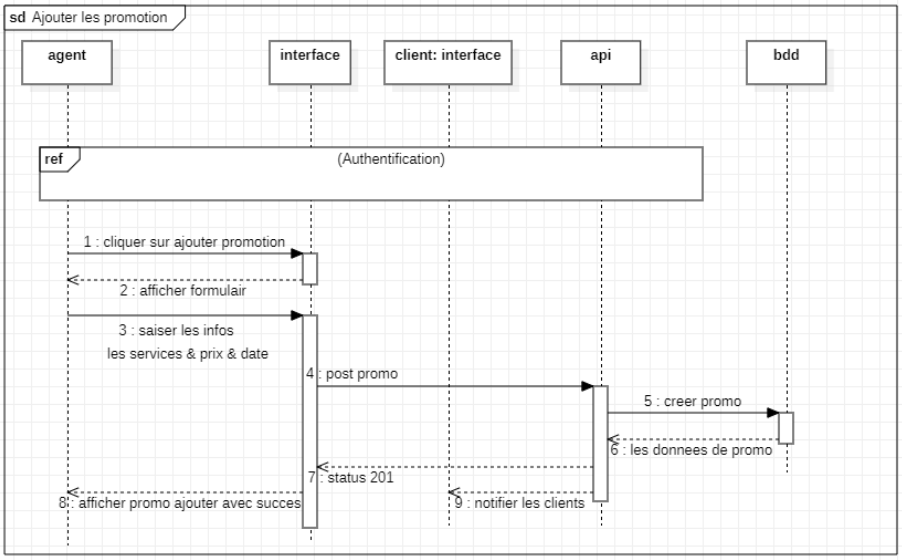


FIGURE 3.12 : Diagramme de séquence - Ajouter les promotions

1. L'agent clique sur le bouton « Ajouter promotion » dans l'interface.
2. L'interface affiche un formulaire de saisie.
3. L'agent remplit le formulaire en indiquant les services concernés, les prix promotionnels et les dates de validité.
4. L'interface envoie une requête `POST promo` à l'API.
5. L'API transmet les informations à la base de données pour créer une nouvelle promotion.
6. La base de données retourne les données de la promotion créée.
7. L'API répond à l'interface avec un statut HTTP 201.
8. L'interface affiche un message de confirmation indiquant que la promotion a été ajoutée avec succès.
9. L'API notifie automatiquement les clients via leur interface des nouvelles promotions disponibles.

— **12. Modifier les services**

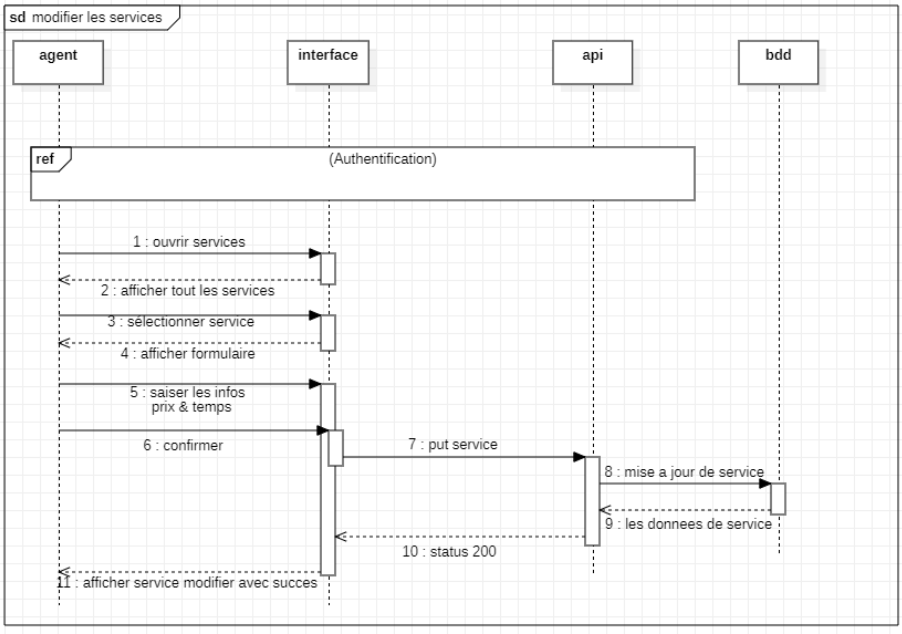


FIGURE 3.13 : Diagramme de séquence - Modifier les services

1. L'agent ouvre la section « Services » depuis l'interface.
2. L'interface affiche tous les services disponibles.
3. L'agent sélectionne le service qu'il souhaite modifier.
4. L'interface affiche un formulaire de modification.
5. L'agent saisit les nouvelles informations telles que le prix et la durée.
6. Il clique sur le bouton « Confirmer ».
7. L'interface envoie une requête `PUT service` à l'API.
8. L'API met à jour les informations du service dans la base de données.
9. La base de données retourne les nouvelles données du service.
10. L'API renvoie un statut HTTP 200 à l'interface.
11. L'interface affiche un message confirmant la modification réussie du service.

— **13. Afficher les revenus, les dépenses et les rendez-vous**

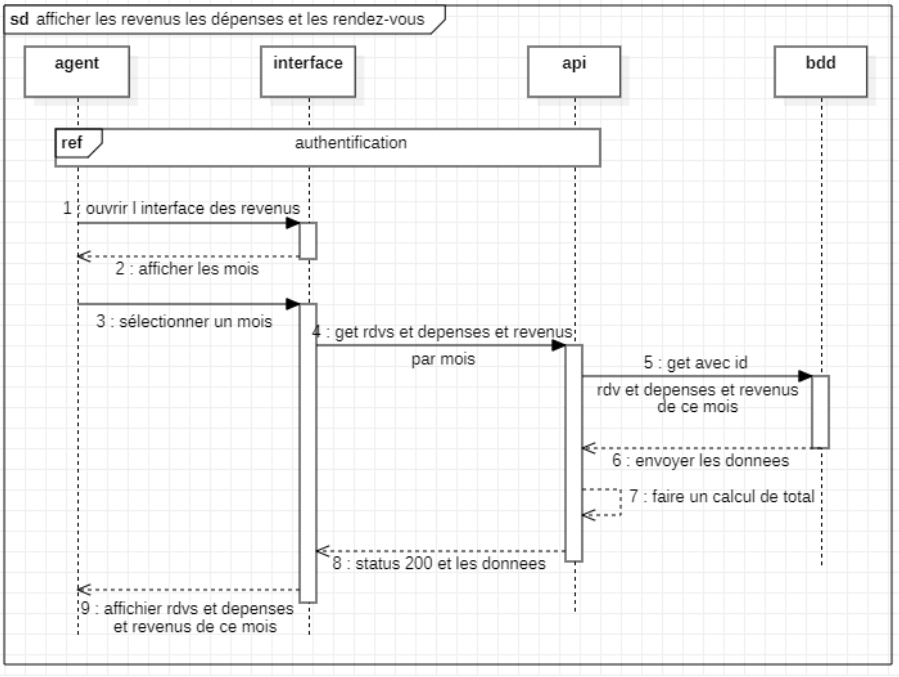


FIGURE 3.14 : Diagramme de séquence - Afficher les revenus, les dépenses et les rendez-vous

1. L'agent ouvre l'interface des revenus.
2. L'interface affiche les mois disponibles.
3. L'agent sélectionne un mois.
4. L'interface envoie une requête `GET` à l'API pour récupérer les rendez-vous, les dépenses et les revenus du mois sélectionné.
5. L'API récupère les informations correspondantes depuis la base de données.
6. La base de données renvoie les rendez-vous, les dépenses et les revenus de ce mois à l'API.
7. L'API effectue un calcul du total (revenus, dépenses).
8. L'API renvoie un statut HTTP 200 ainsi que les données à l'interface.
9. L'interface affiche les rendez-vous, les dépenses et les revenus de ce mois à l'agent.

— **14. Ajouter un agent**

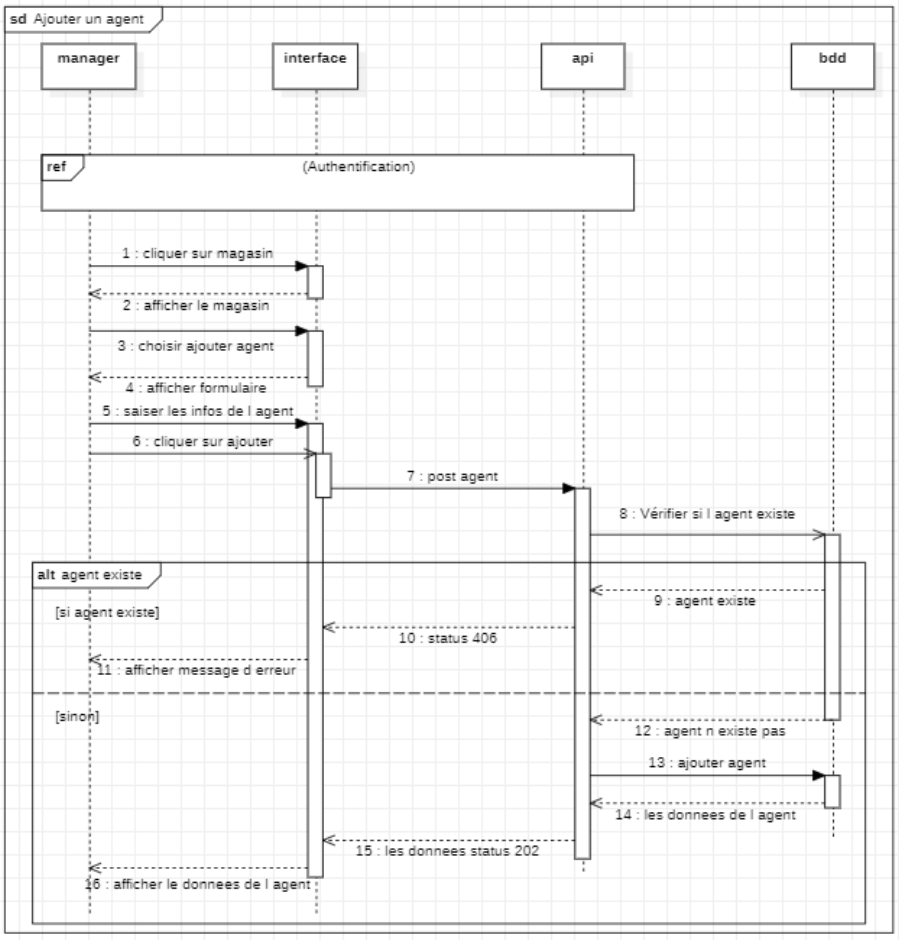


FIGURE 3.15 : Diagramme de séquence - Ajout d'un nouvel agent

1. Le manager clique sur « Magasin » dans l'interface.
2. L'interface affiche la page du magasin.
3. Le manager choisit l'option « Ajouter agent ».
4. L'interface affiche le formulaire d'ajout d'agent.
5. Le manager saisit les informations de l'agent.
6. Il clique sur le bouton « Ajouter ».
7. L'interface envoie une requête `POST agent` à l'API.
8. L'API vérifie si l'agent existe déjà dans la base de données.
9. Si l'agent existe :
   — La base de données informe l'API que l'agent existe.
   — L'API retourne un statut HTTP 406 à l'interface.
   — L'interface affiche un message d'erreur au manager.
10. Sinon :
   — La base de données informe que l'agent n'existe pas.
   — L'API ajoute l'agent dans la base de données.
   — La base retourne les données du nouvel agent.

— L'API envoie un statut HTTP 202 avec les données à l'interface.
— L'interface affiche les données de l'agent au manager.

— **15. Supprimer un agent**

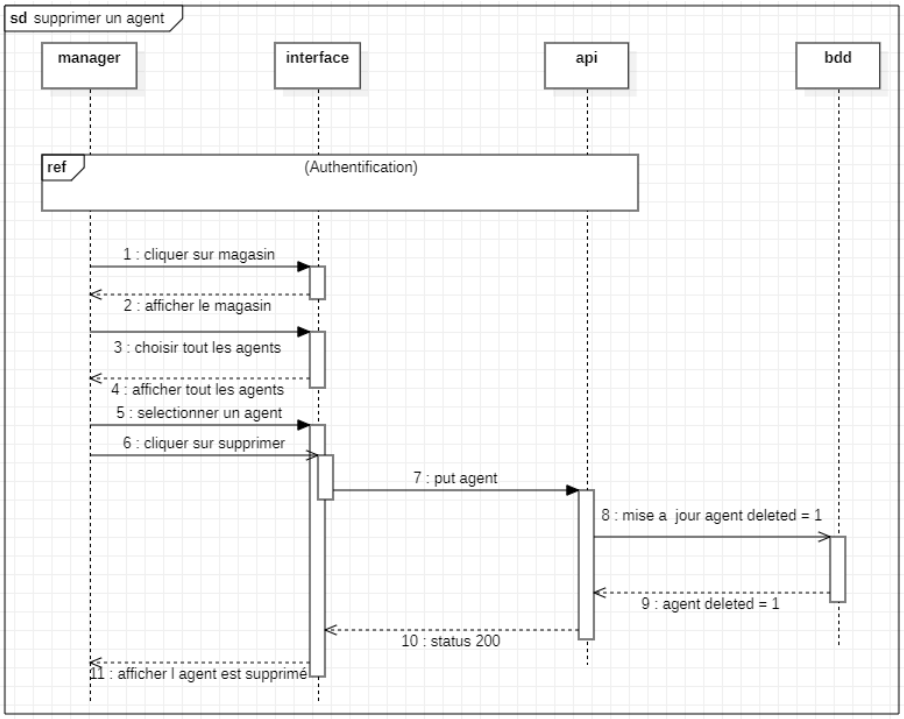


FIGURE 3.16 : Diagramme de séquence - Supprimer un agent

1. Le manager clique sur la section « Magasin » depuis l'interface.
2. L'interface affiche les informations du magasin.
3. Le manager choisit la section « Tous les agents ».
4. L'interface affiche la liste de tous les agents.
5. Le manager sélectionne un agent à supprimer.
6. Il clique sur le bouton « Supprimer ».
7. L'interface envoie une requête `PUT agent` à l'API pour mettre à jour le champ `deleted`.
8. L'API met à jour l'attribut `deleted = 1` de l'agent dans la base de données.
9. La base de données confirme la mise à jour.
10. L'API renvoie un statut HTTP 200 à l'interface.
11. L'interface affiche un message confirmant que l'agent a été supprimé.

— **16. Modifier les informations du magasin**

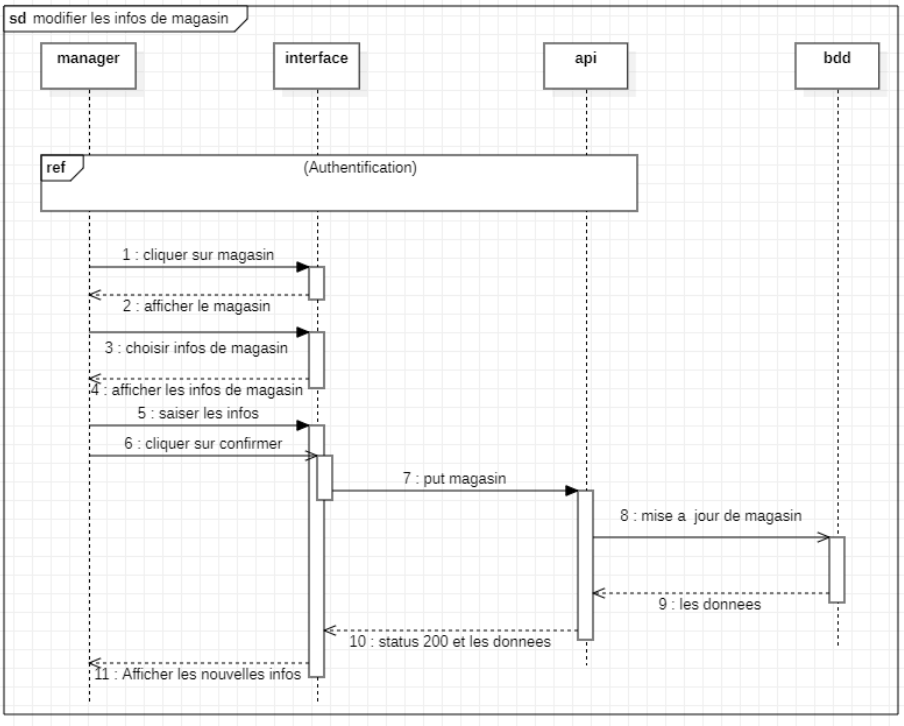


FIGURE 3.17 : Diagramme de séquence - Modifier les informations du magasin

1. Le manager clique sur la section « Magasin » depuis l'interface.
2. L'interface affiche la section du magasin.
3. Le manager choisit l'option « Informations du magasin ».
4. L'interface affiche les informations actuelles du magasin.
5. Le manager modifie les informations du magasin via le formulaire.
6. Il clique sur le bouton « Confirmer ».
7. L'interface envoie une requête `PUT magasin` à l'API.
8. L'API transmet la demande de mise à jour à la base de données.
9. La base de données met à jour les informations du magasin et renvoie les données mises à jour à l'API.
10. L'API renvoie un statut HTTP 200 ainsi que les nouvelles données à l'interface.
11. L'interface affiche les nouvelles informations du magasin.

— **17. Ajouter un magasin**

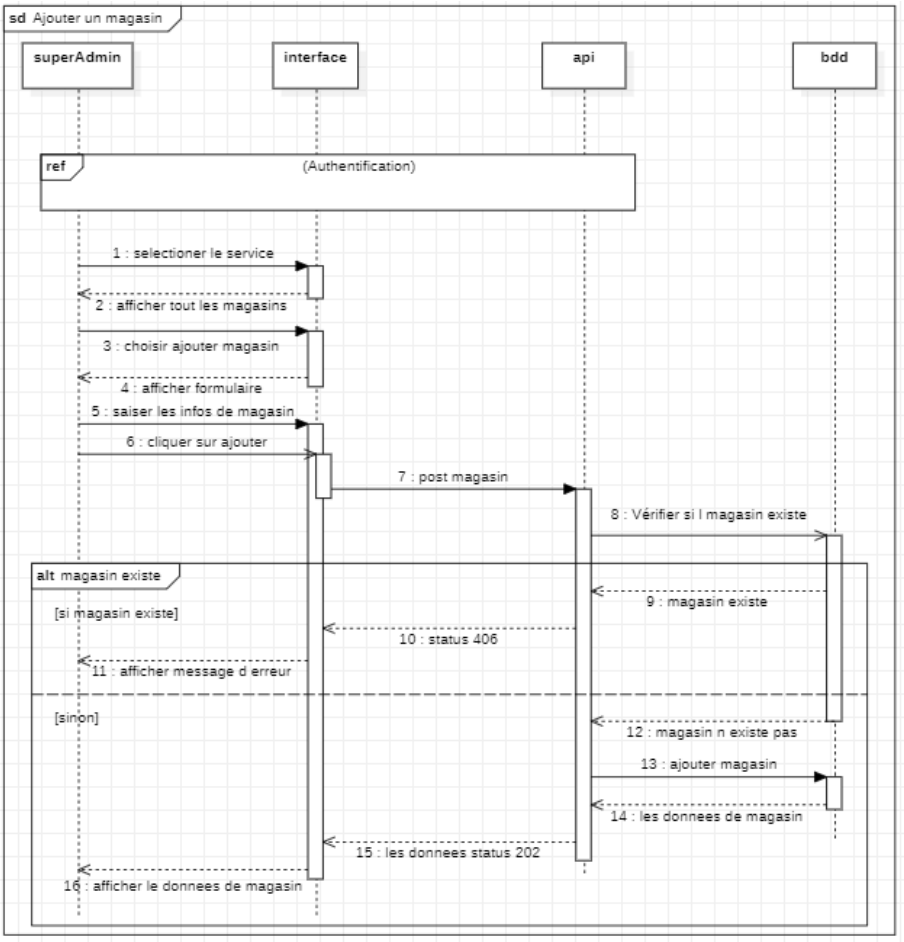


FIGURE 3.18 : Diagramme de séquence - Ajouter un magasin

1. Le super administrateur sélectionne le service concerné.
2. L'interface affiche la liste de tous les magasins associés.
3. Le super administrateur choisit l'option « Ajouter magasin ».
4. L'interface affiche le formulaire d'ajout de magasin.
5. Le super administrateur saisit les informations du magasin.
6. Il clique sur le bouton « Ajouter ».
7. L'interface envoie une requête `POST magasin` à l'API.
8. L'API vérifie si le magasin existe déjà dans la base de données.
9. Si le magasin existe :
   — La base de données indique que le magasin existe.
   — L'API retourne un statut HTTP 406 à l'interface.
   — L'interface affiche un message d'erreur au super administrateur.
10. Sinon :
   — La base indique que le magasin n'existe pas.
   — L'API ajoute le magasin dans la base de données.
   — La base retourne les données du nouveau magasin.

— L'API envoie un statut HTTP 202 avec les données à l'interface.
— L'interface affiche les données du magasin au super administrateur.

— **18. Supprimer un magasin**

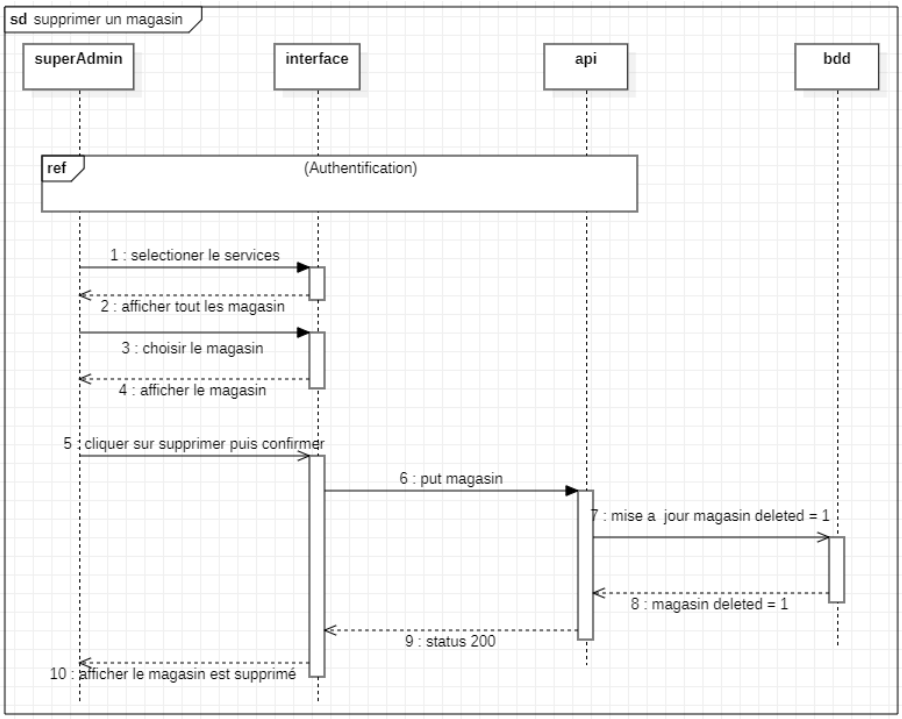


FIGURE 3.19 : Diagramme de séquence - Supprimer un magasin

1. Le superAdmin sélectionne le service « Magasins » depuis l'interface.
2. L'interface affiche la liste de tous les magasins.
3. Le superAdmin choisit le magasin à supprimer.
4. L'interface affiche les détails du magasin sélectionné.
5. Le superAdmin clique sur le bouton « Supprimer » puis confirme l'action.
6. L'interface envoie une requête `PUT magasin` à l'API pour indiquer que le magasin doit être supprimé.
7. L'API met à jour l'attribut `deleted = 1` pour ce magasin dans la base de données.
8. La base de données confirme que `deleted = 1` a bien été enregistré.
9. L'API retourne un statut HTTP 200 à l'interface.
10. L'interface affiche que le magasin a été supprimé.

3.2.3 Diagramme de cas d'utilisation du service de livraison

Le diagramme ci-dessous présente les différents cas d'utilisation du service logistique dans le système Mefria. Il met en évidence les interactions entre les différents acteurs (clients, livreurs, marchands, administrateurs) et le système, en précisant les fonctionnalités accessibles à chacun. Ce schéma permet de visualiser les rôles et responsabilités liés à la gestion des commandes, à la livraison ainsi qu'au suivi des opérations :

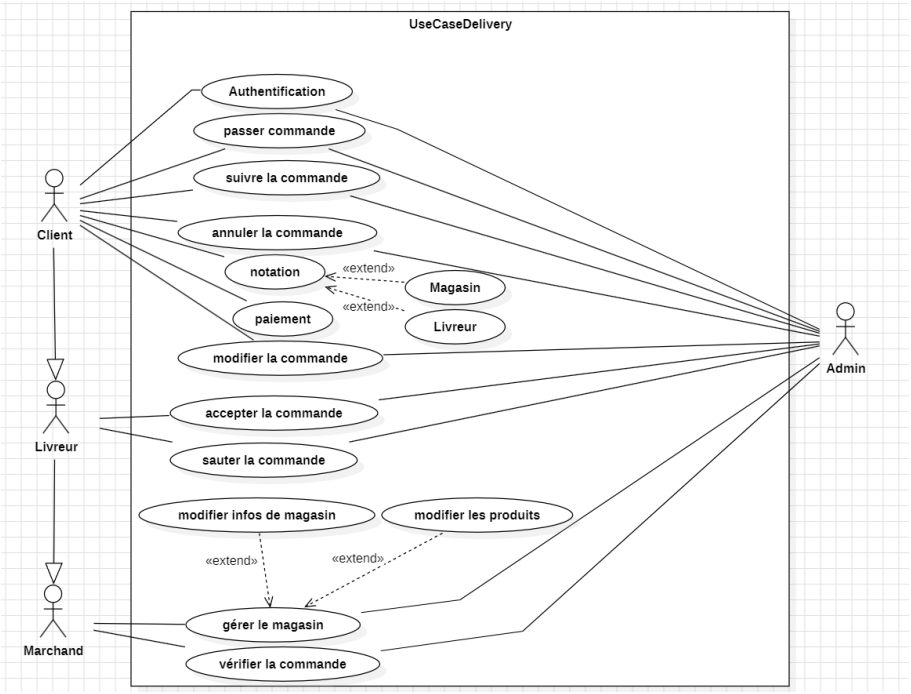


FIGURE 3.20 : Diagramme de cas d'utilisation du service de livraison

Remarque : L'authentification est requise pour toutes les opérations du système, comme indiqué par la référence `(Authentification)` dans chaque diagramme de séquence.

Acteurs identifiés :

- **Client** : utilisateur qui passe et suit les commandes.
- **Livreur** : personne chargée de la livraison des commandes.
- **Marchand** : commerçant qui gère un magasin ou un point de vente sur Mefria.
- **Admin** : administrateur ayant accès à l'ensemble des fonctions de gestion.

Principaux cas d'utilisation :

- Authentification (tous les rôles)
- Passer, suivre, annuler ou modifier une commande (client)
- Paiement et notation du service (client)
- Accepter ou sauter une commande (livreur)
- Modifier les informations ou les produits du magasin (marchand)
- Vérification et gestion des commandes et magasins (admin)

Ce diagramme offre une vue d'ensemble des fonctionnalités du service de livraison de Mefria, en illustrant les interactions entre les utilisateurs et les fonctions logistiques de l'application. Il sert de base aux spécifications techniques détaillées à venir.

Descriptions textuelles et diagrammes de séquence des cas d'utilisation

— **1. Passer une commande**

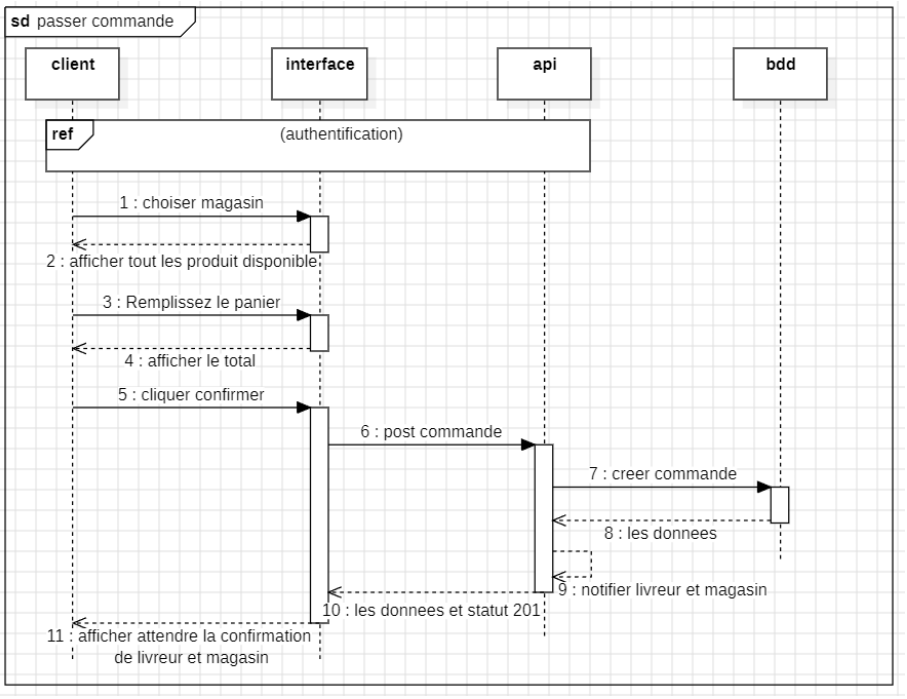


FIGURE 3.21 : Diagramme de séquence - Passer une commande

1. Le client choisit un magasin dans l'interface de l'application.
2. L'interface récupère et affiche tous les produits disponibles dans ce magasin.
3. Le client remplit son panier en sélectionnant les produits souhaités.
4. L'interface calcule et affiche le total de la commande.
5. Le client clique sur le bouton "Confirmer" pour valider la commande.
6. L'interface envoie une requête `POST commande` à l'API pour créer la commande.
7. L'API traite la requête et appelle la base de données pour créer une nouvelle entrée de commande.
8. La base de données enregistre les données de la commande et répond à l'API.
9. L'API retourne un statut HTTP 201 (Created) avec les données de confirmation.
10. L'interface affiche à l'utilisateur un message indiquant que la commande a été envoyée, et qu'il doit attendre la confirmation d'un livreur.

— **2. Suivre une commande**

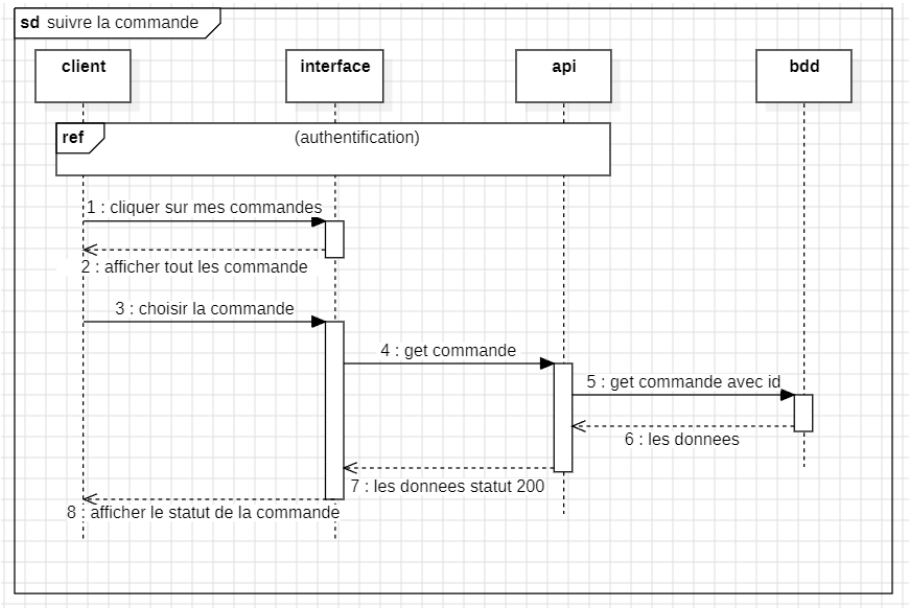


FIGURE 3.22 : Diagramme de séquence - Suivre une commande

1. Le client clique sur « Mes commandes » dans l'interface de l'application.
2. L'interface affiche la liste de toutes les commandes du client.
3. Le client sélectionne une commande spécifique pour en suivre l'état.
4. L'interface envoie une requête `GET commande` à l'API.
5. L'API interroge la base de données avec l'identifiant de la commande.
6. La base de données retourne les informations correspondantes à l'API.
7. L'API répond avec un statut HTTP 200 et les données de la commande.
8. L'interface affiche le statut actuel de la commande au client.

— **3. Annuler une commande**

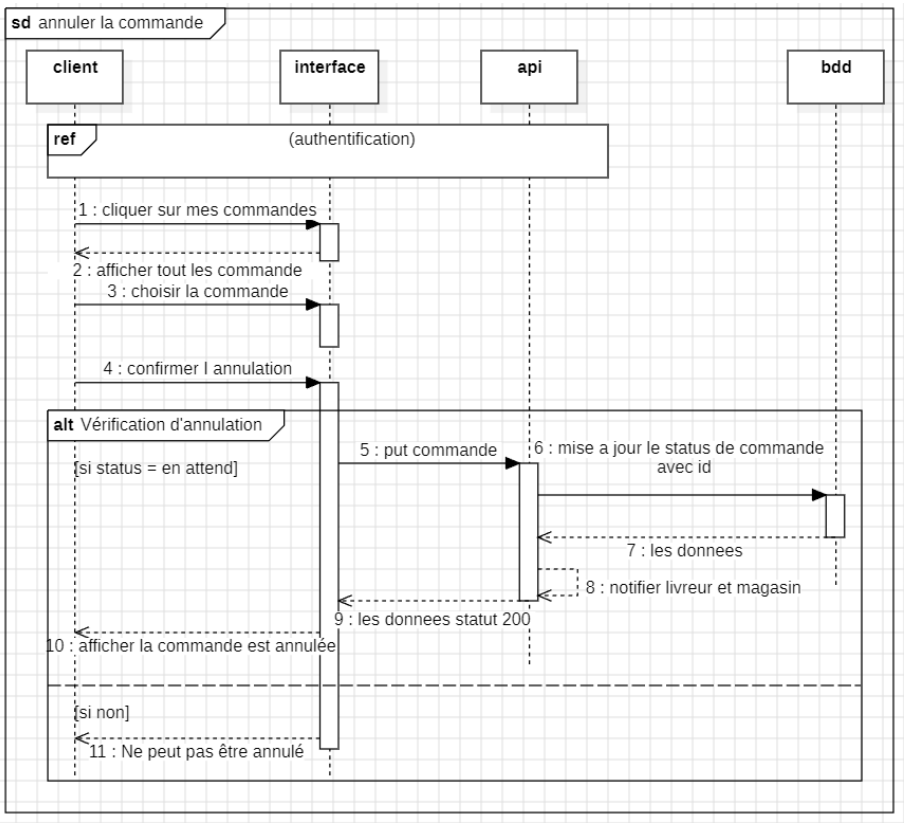


FIGURE 3.23 : Diagramme de séquence - Annuler une commande

1. Le client clique sur « Mes commandes » dans l'interface de l'application.
2. L'interface affiche la liste de toutes les commandes du client.
3. Le client sélectionne une commande spécifique à annuler.
4. Le client confirme l'annulation de la commande.
5. L'interface envoie une requête `PUT commande` à l'API pour mettre à jour le statut.
6. L'API met à jour le statut de la commande dans la base de données en utilisant son identifiant.
7. La base de données retourne les données mises à jour à l'API.
8. L'API répond avec un statut HTTP 200 confirmant la réussite de l'opération.
9. L'interface affiche un message indiquant que la commande a bien été annulée.

— **4. Noter le magasin**

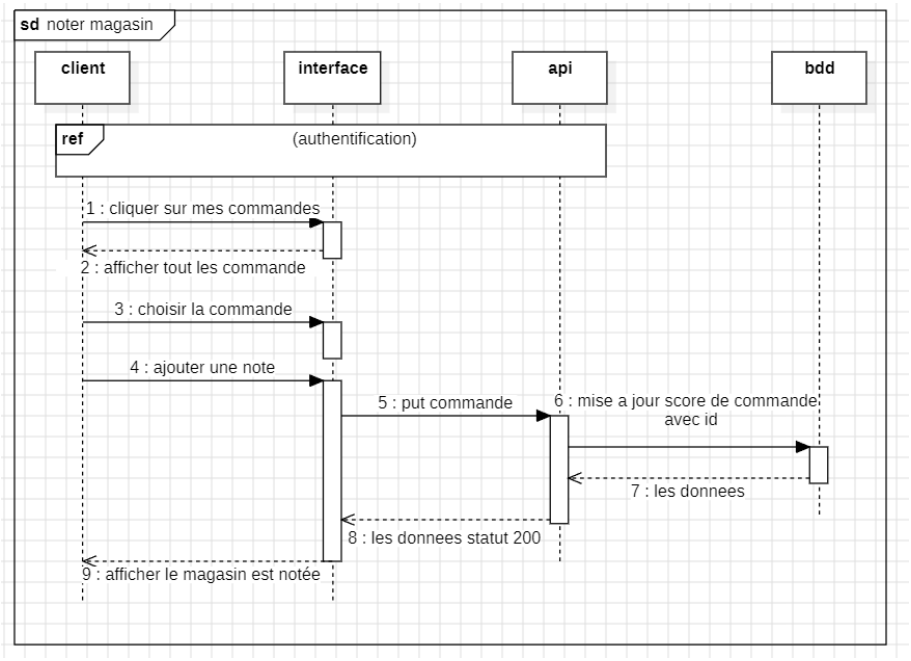


FIGURE 3.24 : Diagramme de séquence - Noter le magasin

1. Le client clique sur « Mes commandes » dans l'interface de l'application.
2. L'interface affiche la liste de toutes les commandes du client.
3. Le client sélectionne une commande spécifique pour évaluer le magasin.
4. Le client ajoute une note (évaluation) pour le magasin.
5. L'interface envoie une requête `PUT commande` à l'API pour mettre à jour le score.
6. L'API met à jour le score du magasin dans la base de données en utilisant l'identifiant de la commande.
7. La base de données retourne les données mises à jour à l'API.
8. L'API répond avec un statut HTTP 200 confirmant la réussite de l'opération.
9. L'interface affiche un message confirmant que le magasin a bien été noté.

— **5. Noter le livreur**

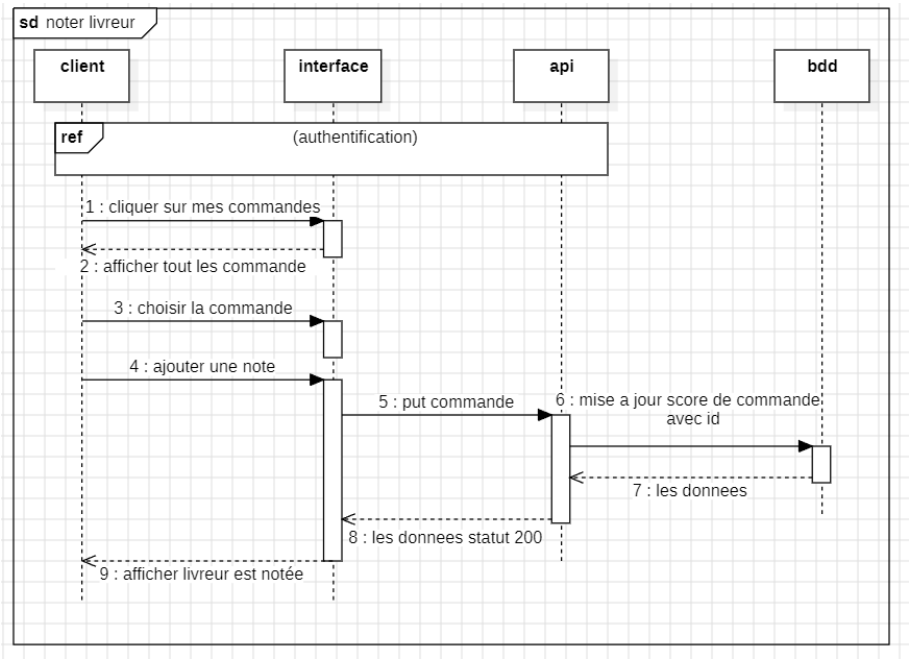


FIGURE 3.25 : Diagramme de séquence - Noter le livreur

1. Le client clique sur « Mes commandes » dans l'interface de l'application.
2. L'interface affiche la liste de toutes les commandes du client.
3. Le client sélectionne une commande spécifique pour évaluer le livreur.
4. Le client ajoute une note (évaluation) pour le livreur.
5. L'interface envoie une requête `PUT commande` à l'API pour mettre à jour l'évaluation.
6. L'API met à jour le score du livreur dans la base de données en utilisant l'identifiant de la commande.
7. La base de données retourne les données mises à jour à l'API.
8. L'API répond avec un statut HTTP 200 confirmant la réussite de l'opération.
9. L'interface affiche un message confirmant que le livreur a bien été noté.

— **6. Modifier une commande**

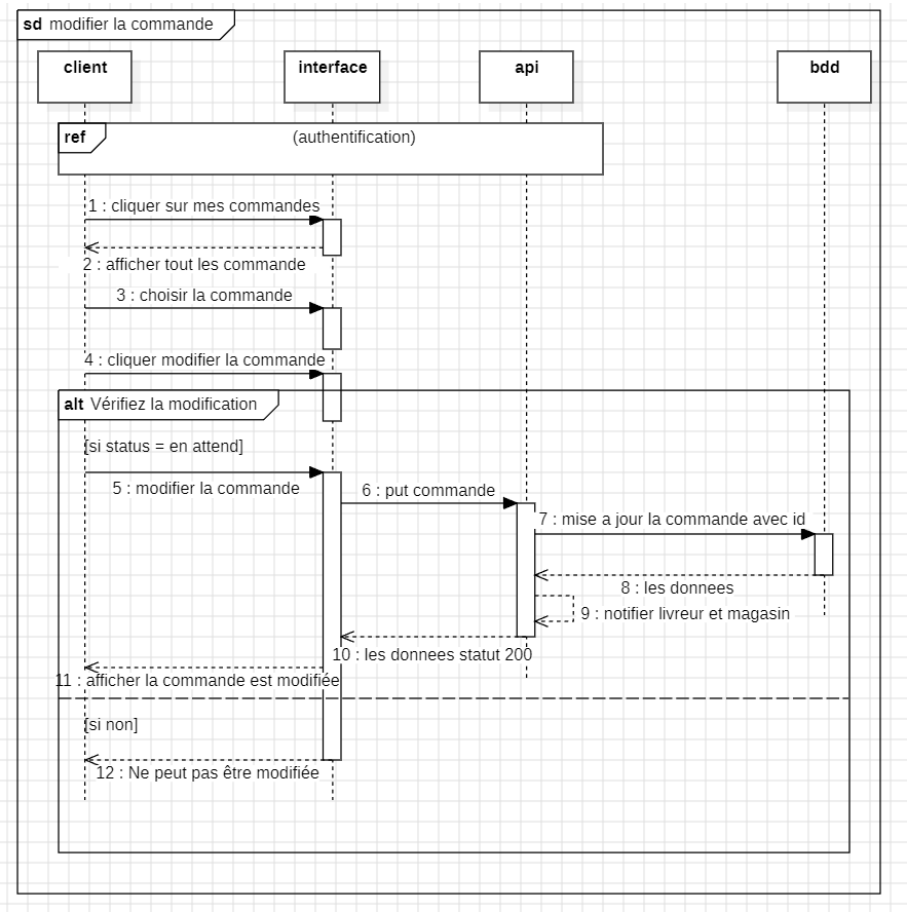


FIGURE 3.26 : Diagramme de séquence - Modifier une commande

1. Le client clique sur « Mes commandes » dans l'interface de l'application.
2. L'interface affiche toutes les commandes du client.
3. Le client choisit la commande qu'il souhaite modifier.
4. Il clique ensuite sur le bouton « Modifier la commande ».
5. Le système vérifie si la commande peut être modifiée :

   — **Si le statut de la commande est « en attente » :**

   5. Le client modifie la commande.
   6. Une requête `PUT commande` est envoyée à l'API avec les modifications.
   7. L'API met à jour la commande avec l'identifiant donné dans la base de données.
   8. Les données mises à jour sont récupérées.
   9. L'API notifie le livreur et le magasin de la modification.
   10. L'API renvoie un statut HTTP 200 à l'interface.
   11. L'interface affiche que la commande a bien été modifiée.

   — **Sinon :**

   12. L'interface informe le client que la commande ne peut pas être modifiée.

— **7. Accepter une commande**

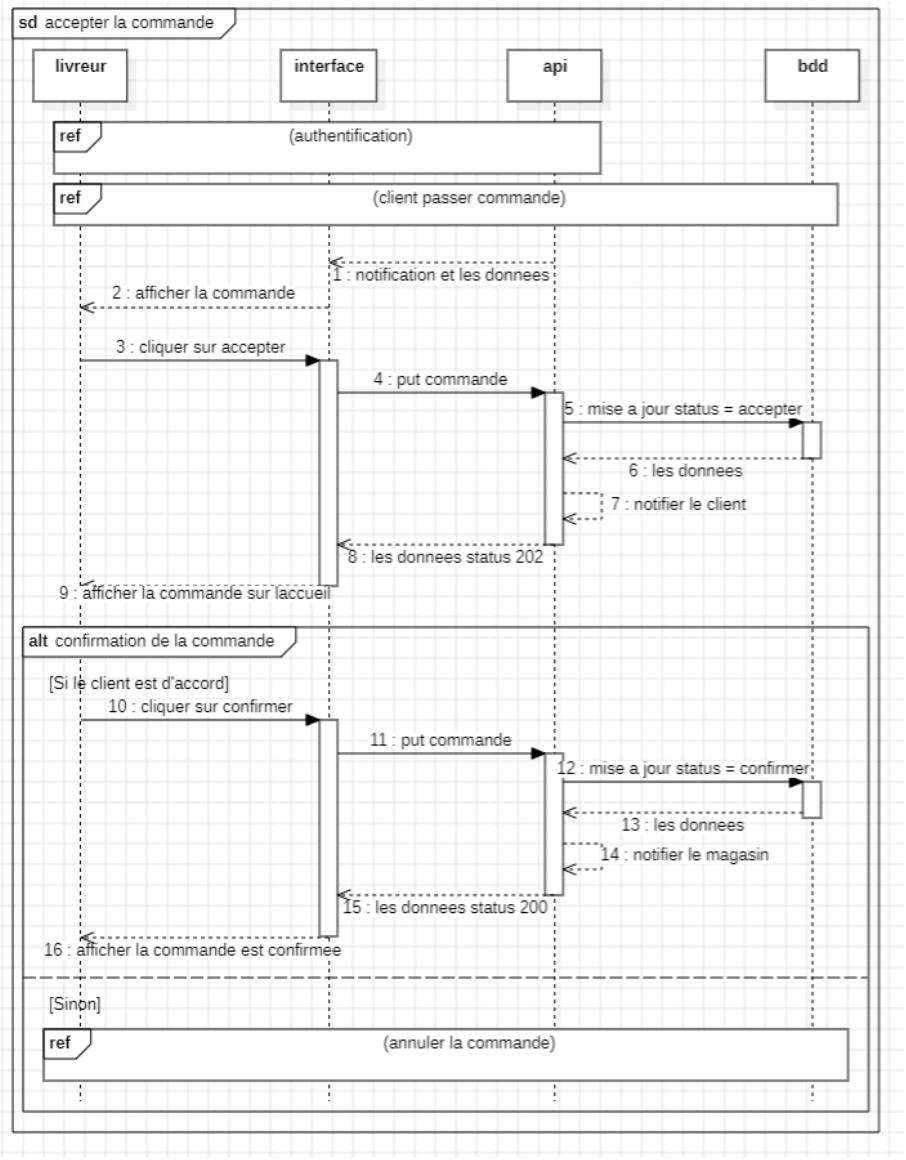


FIGURE 3.27 : Diagramme de séquence - Accepter une commande

1. Le client passe une commande via l'interface de l'application.
2. Le livreur, après authentification, reçoit une notification avec les détails de la commande.
3. L'interface affiche les informations de la commande au livreur.
4. Le livreur clique sur le bouton « Accepter » pour prendre en charge la commande.
5. L'interface envoie une requête `PUT commande` à l'API.
6. L'API met à jour le statut de la commande à `accepter` dans la base de données.
7. L'API récupère les données mises à jour depuis la base de données.
8. L'API notifie le client que la commande a été acceptée.
9. L'API renvoie un statut HTTP 202 à l'interface.
10. L'interface affiche la commande sur l'écran d'accueil du livreur.

— **Si le client est d'accord :**

11. Le livreur clique sur « Confirmer ».
12. Une nouvelle requête `PUT commande` est envoyée à l'API.
13. L'API met à jour le statut de la commande à `confirmé` dans la base de données.
14. Les données mises à jour sont récupérées.
15. L'API notifie le magasin que la commande est confirmée.
16. L'API renvoie un statut HTTP 200 à l'interface.
17. L'interface affiche que la commande est confirmée.

— **Sinon :**

18. Le livreur annule la commande via un scénario alternatif.

— **8. Sauter une commande**

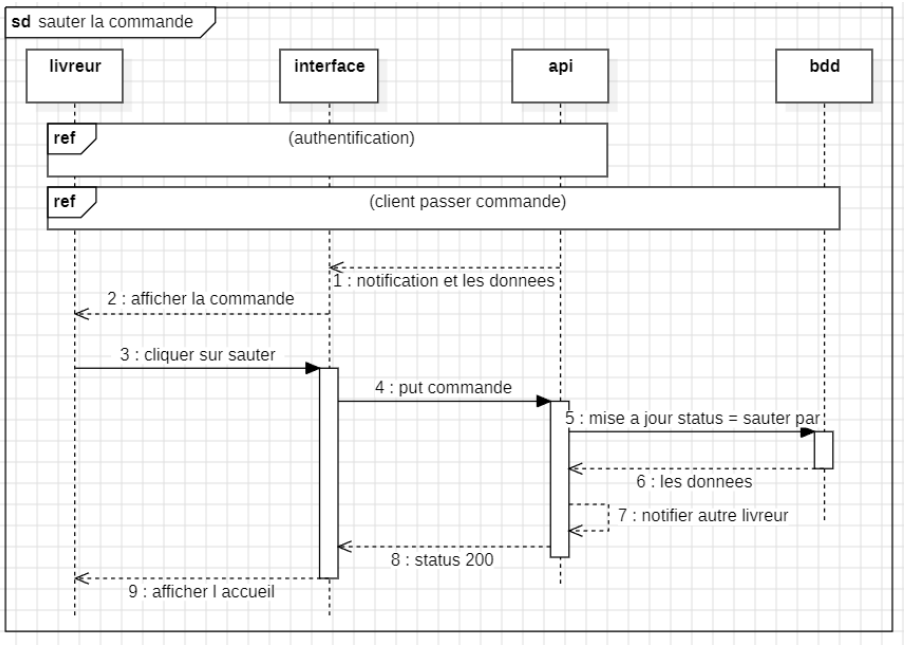


FIGURE 3.28 : Diagramme de séquence - Sauter une commande

1. Le livreur reçoit une notification contenant les données d'une nouvelle commande.
2. L'interface affiche les informations de la commande au livreur.
3. Le livreur clique sur le bouton « Sauter » pour ne pas prendre en charge la commande.
4. L'interface envoie une requête `PUT commande` à l'API.
5. L'API met à jour le statut de la commande dans la base de données avec la valeur `sauter par`.
6. L'API récupère les données mises à jour depuis la base de données.
7. L'API notifie un autre livreur disponible pour prendre la commande.
8. L'API renvoie un statut HTTP 200 à l'interface.
9. L'interface redirige le livreur vers l'écran d'accueil.

— **9. Modifier les informations d'un magasin**

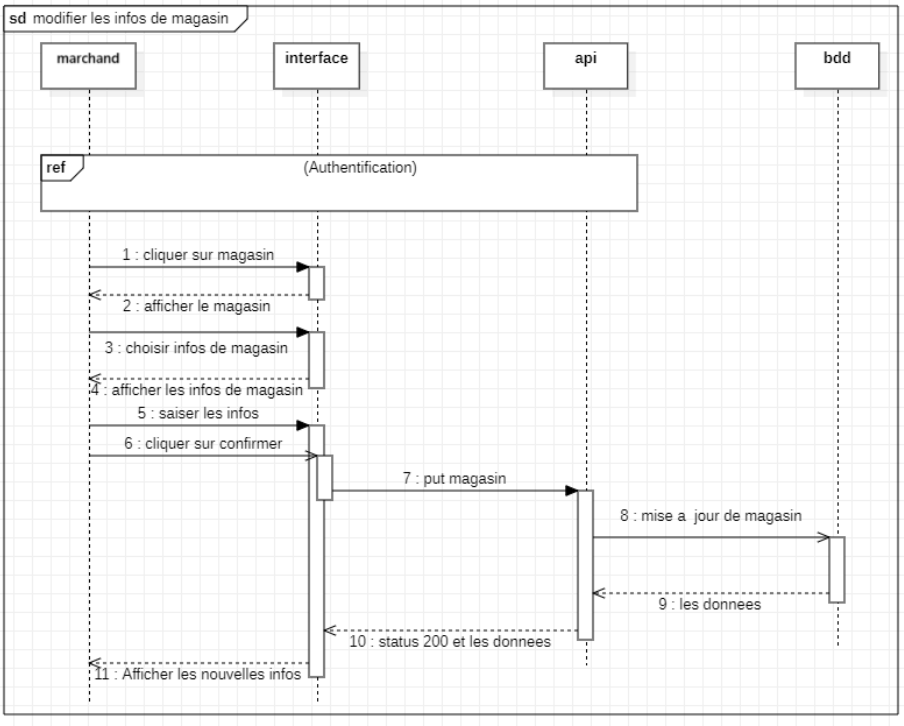


FIGURE 3.29 : Diagramme de séquence - Modifier les informations d'un magasin

1. Le marchand clique sur le bouton « Magasin » dans l'interface.
2. L'interface affiche les informations du magasin.
3. Le marchand choisit l'option « Infos de magasin ».
4. L'interface affiche les informations détaillées du magasin.
5. Le marchand saisit les nouvelles informations à modifier.
6. Le marchand clique sur le bouton « Confirmer ».
7. L'interface envoie une requête `PUT magasin` à l'API.
8. L'API met à jour les informations du magasin dans la base de données.
9. La base de données retourne les nouvelles données mises à jour.
10. L'API renvoie un statut HTTP 200 ainsi que les nouvelles données à l'interface.
11. L'interface affiche les nouvelles informations du magasin au marchand.

— **10. Modifier les informations d'un produit**

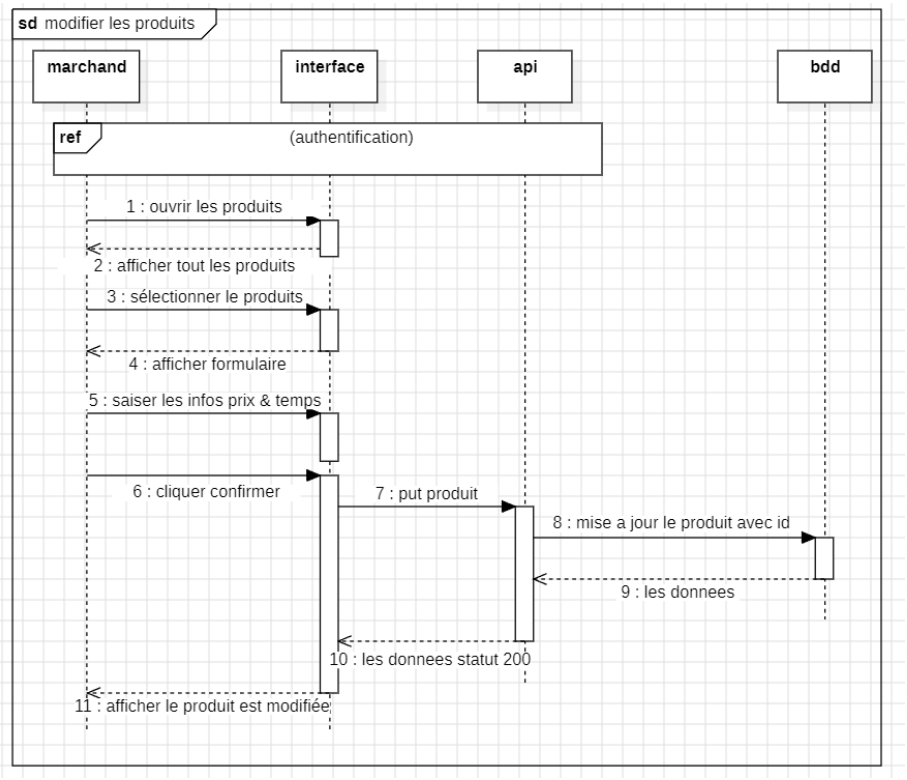


FIGURE 3.30 : Diagramme de séquence - Modifier les informations d'un produit

1. Le marchand clique sur le bouton « Produits » dans l'interface.
2. L'interface affiche la liste de tous les produits.
3. Le marchand sélectionne le produit à modifier.
4. L'interface affiche un formulaire de modification.
5. Le marchand saisit les nouvelles informations (prix et temps).
6. Le marchand clique sur le bouton « Confirmer ».
7. L'interface envoie une requête `PUT produit` à l'API.
8. L'API met à jour le produit dans la base de données via son identifiant.
9. La base de données retourne les nouvelles données du produit.
10. L'API renvoie un statut HTTP 200 et les nouvelles données à l'interface.
11. L'interface informe le marchand que le produit a été modifié avec succès.

— **11. Vérifier une commande**

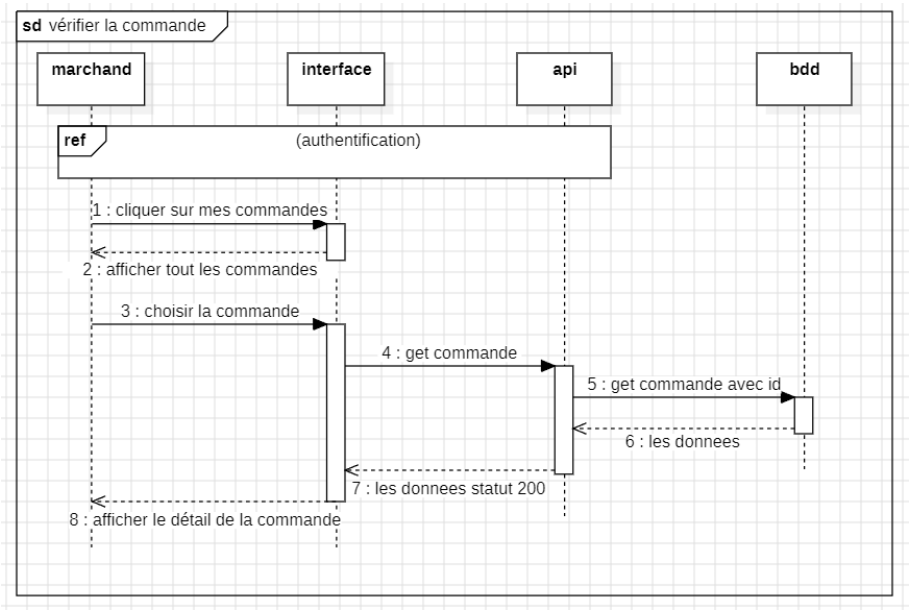


FIGURE 3.31 : Diagramme de séquence - Vérifier une commande

1. Le marchand clique sur le bouton « Mes commandes » dans l'interface.
2. L'interface affiche la liste de toutes les commandes.
3. Le marchand choisit une commande à consulter.
4. L'interface envoie une requête `GET commande` à l'API.
5. L'API interroge la base de données pour récupérer la commande à partir de son identifiant.
6. La base de données renvoie les données de la commande à l'API.
7. L'API retourne un statut HTTP 200 et les données à l'interface.
8. L'interface affiche le détail de la commande sélectionnée au marchand.

3.2.4 Diagramme de cas d'utilisation du service e-commerce

Le diagramme ci-dessous présente les différents cas d'utilisation du module e-commerce du système Mefria. Il met en évidence les interactions possibles entre les acteurs (clients, e-commerçants, administrateurs) et le système, en détaillant les fonctionnalités offertes telles que la gestion des produits, des commandes, des magasins et des paiements. Ce schéma permet de visualiser les rôles et responsabilités associés à chaque utilisateur dans le processus d'achat en ligne, de gestion commerciale et d'administration.

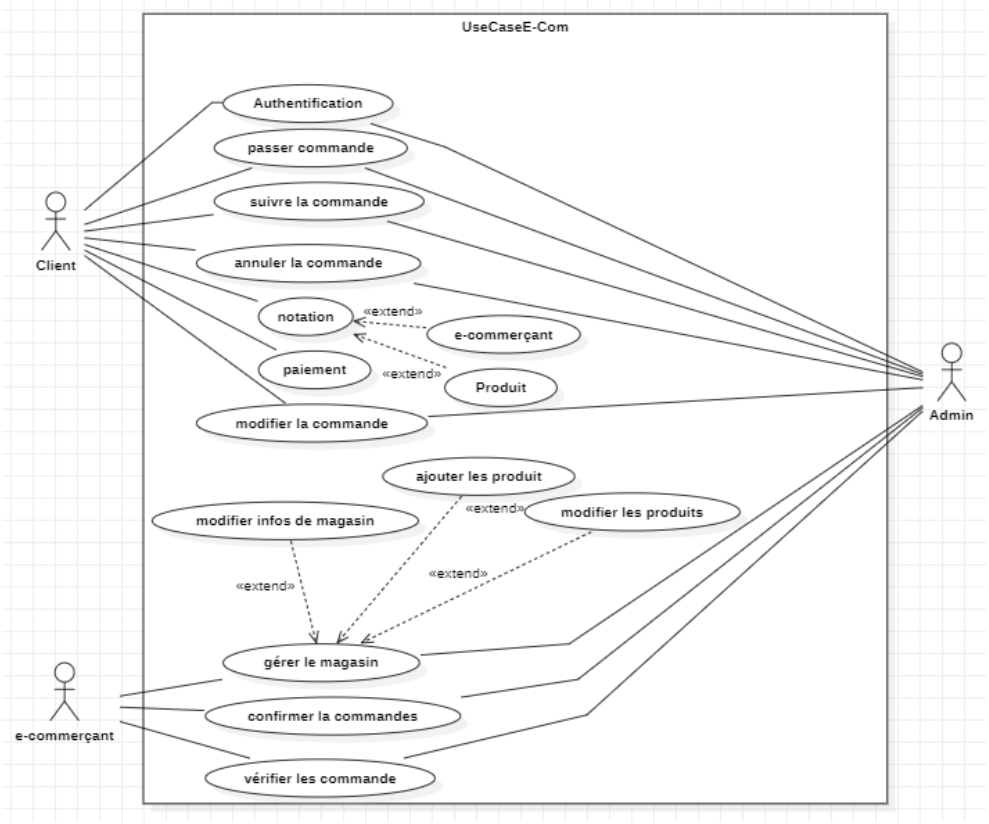


FIGURE 3.32 : Diagramme de cas d'utilisation du service e-commerce

Acteurs identifiés :

- **Client** : utilisateur final effectuant des achats et des paiements en ligne.
- **E-commerçant** : marchand responsable de la gestion de son magasin et de ses produits.
- **Admin** : administrateur supervisant l'ensemble du système, pouvant modifier ou supprimer des données globales.

Principaux cas d'utilisation :

- Authentification (tous les rôles)
- Passer, modifier ou annuler une commande (client)
- Suivi des commandes (client)
- Notation des produits et des commerçants (client)
- Gestion de magasin et des produits (e-commerçant)
- Confirmation et vérification des commandes (e-commerçant)
- Supervision des opérations e-commerce (admin)

Ce diagramme offre une vue d'ensemble des fonctionnalités clés du module e-commerce de Mefria, en illustrant les interactions entre les différents acteurs et les services fournis. Il constitue une base essentielle pour les diagrammes de séquence et de classes à venir.

Remarque : L'authentification est une condition préalable pour toutes les opérations décrites dans ce module, comme représenté dans les diagrammes de séquence associés.

Descriptions textuelles et diagrammes de séquence des cas d'utilisation

Chaque cas d'utilisation est décrit sous forme de processus en étapes, comme s'il était exécuté par un middleware entre le client et le système. Cette présentation met en valeur la logique de traitement et les vérifications automatiques effectuées par le système. Chaque description est accompagnée d'un diagramme de séquence illustrant l'enchaînement des interactions entre les acteurs et le système.

Les descriptions suivantes détaillent le comportement du module e-commerce du système Mefria, en commençant par les actions les plus fréquentes comme la consultation ou la modification de commande, jusqu'aux processus métier comme la gestion de magasin et l'administration des produits.

— **1. Passer une commande**

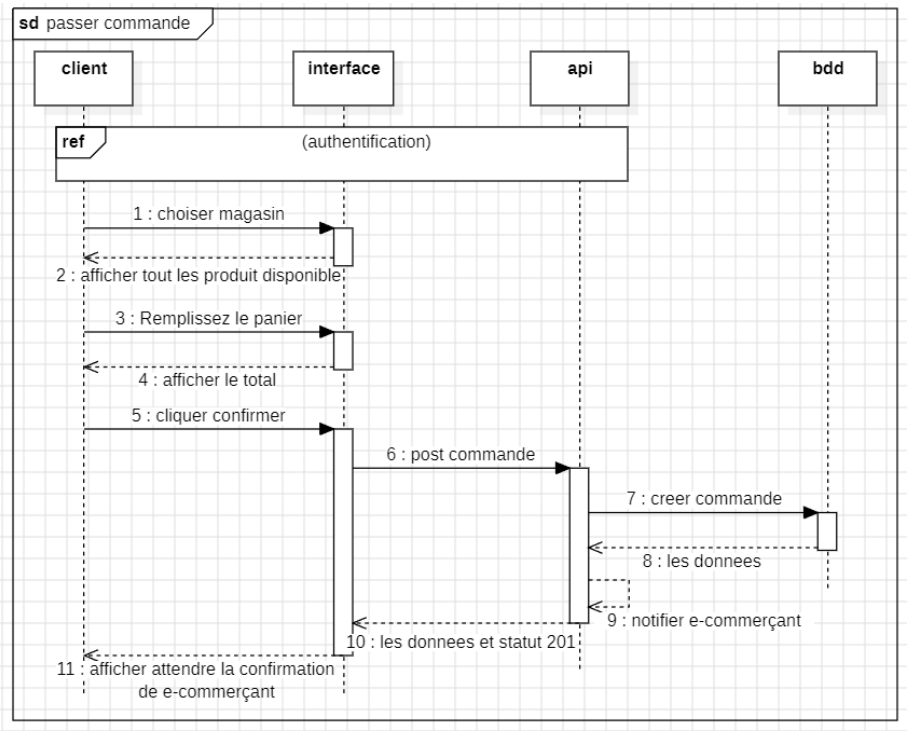


FIGURE 3.33 : Diagramme de séquence - Passer une commande

1. Le client choisit un magasin depuis l'interface.
2. L'interface affiche tous les produits disponibles du magasin sélectionné.
3. Le client remplit son panier avec les produits souhaités.
4. L'interface affiche le total de la commande.
5. Le client clique sur le bouton « Confirmer ».
6. L'interface envoie une requête `POST commande` à l'API.
7. L'API crée une nouvelle commande dans la base de données.
8. La base de données retourne les informations de la commande.

9. L'API notifie le e-commerçant qu'une nouvelle commande est en attente de confirmation.
10. L'API renvoie un statut HTTP 201 et les données à l'interface.
11. L'interface affiche un message au client indiquant que la commande est en attente de confirmation du e-commerçant.

— **2. Suivre une commande**

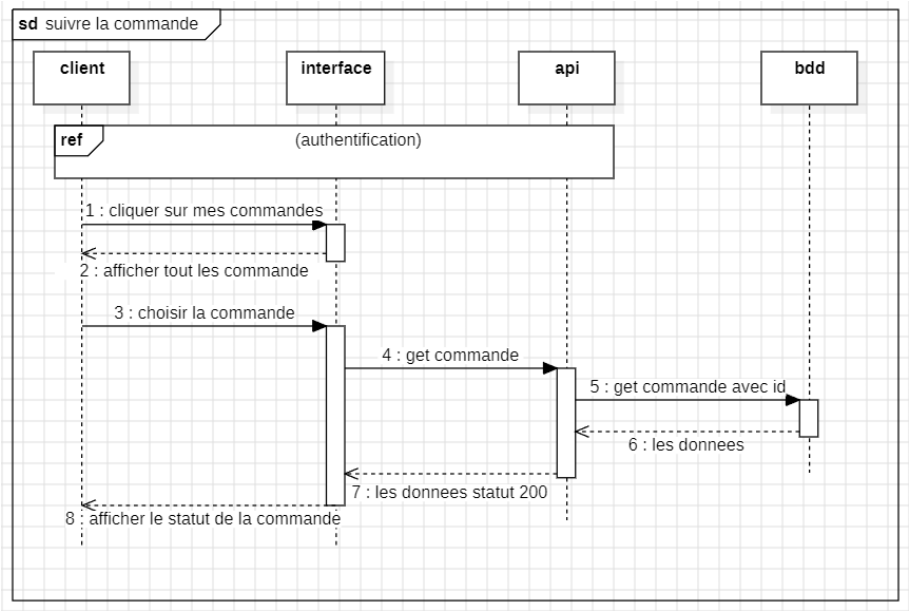


FIGURE 3.34 : Diagramme de séquence - Suivre une commande

1. Le client clique sur l'option « Mes commandes » dans l'interface.
2. L'interface affiche la liste de toutes les commandes passées par le client.
3. Le client sélectionne une commande spécifique dans la liste.
4. L'interface envoie une requête `GET commande` à l'API pour récupérer les détails de la commande.
5. L'API interroge la base de données avec l'identifiant de la commande (`GET commande avec id`).
6. La base de données retourne les données de la commande demandée.
7. L'API renvoie un statut HTTP 200 et les données de la commande à l'interface.
8. L'interface affiche le statut actuel de la commande (par exemple, "En attente", "Confirmée", "Expédiée", etc.) au client.

— **3. Annuler une commande**

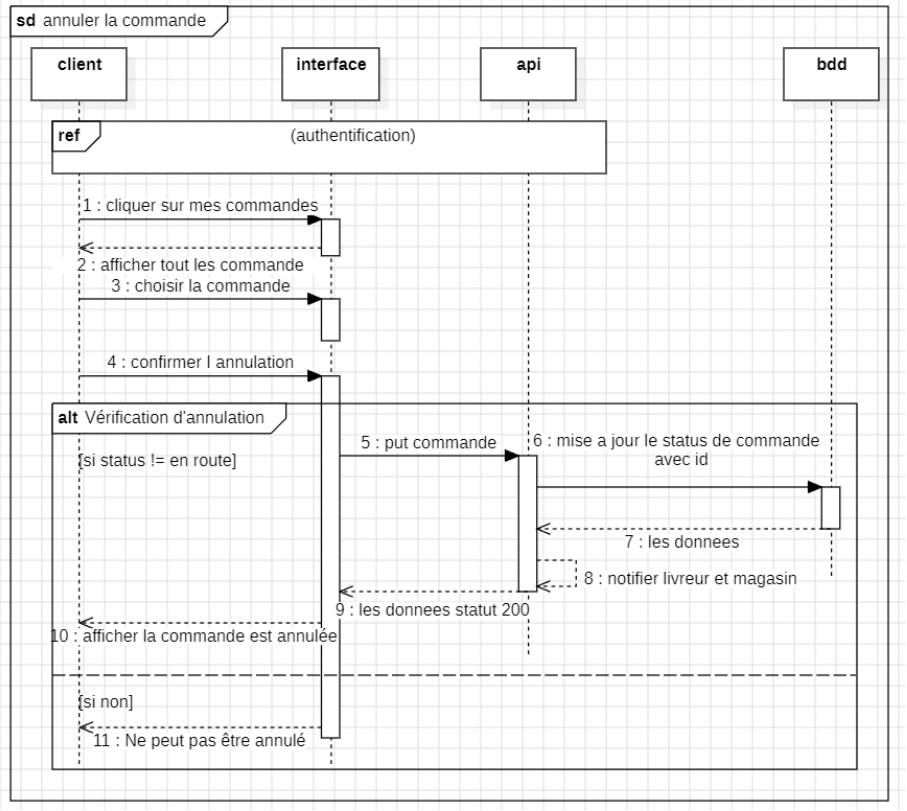


FIGURE 3.35 : Diagramme de séquence - Annuler une commande

1. Le client clique sur « Mes commandes » dans l'interface.
2. L'interface affiche la liste des commandes passées.
3. Le client sélectionne une commande spécifique à annuler.
4. Le client clique sur le bouton « Annuler » pour la commande sélectionnée.
5. L'interface envoie une requête `PUT commande` à l'API.
6. L'API met à jour le statut de la commande à `annulée` dans la base de données.
7. L'API récupère les données mises à jour depuis la base de données.
8. L'API notifie le livreur et le magasin de l'annulation.
9. L'API renvoie un statut HTTP 200 à l'interface.
10. L'interface affiche la confirmation d'annulation au client.

— **Si l'annulation est possible :**

11. Le système vérifie que le statut initial était « En attente ».
12. La base de données confirme la mise à jour du statut.
13. Le client reçoit une notification de confirmation.
14. L'interface affiche « Commande annulée avec succès ».

— **Sinon :**

15. Le système détecte un statut non annulable (ex : « Expédiée »).
16. L'API renvoie un statut HTTP 403 à l'interface.
17. L'interface affiche « Cette commande ne peut plus être annulée ».

18. Le client est redirigé vers la liste des commandes.

— **4. Noter le e-commerçant**

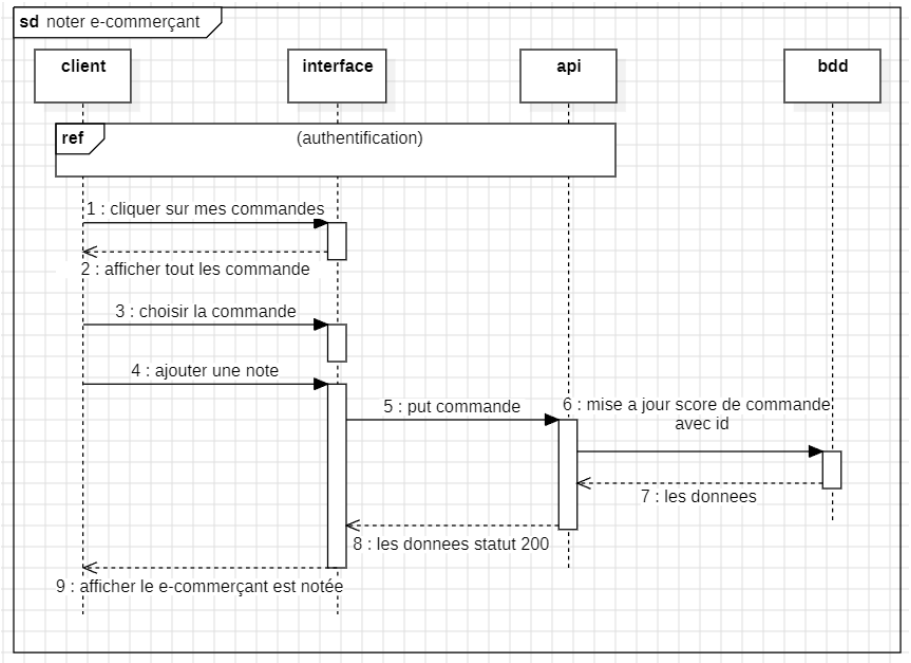


FIGURE 3.36 : Diagramme de séquence - Noter le e-commerçant

1. Le client clique sur « Mes commandes » dans l'interface de l'application.
2. L'interface affiche la liste de toutes les commandes du client.
3. Le client choisit une commande qu'il souhaite évaluer.
4. Le client ajoute une note pour cette commande.
5. L'interface envoie une requête `PUT commande` à l'API pour transmettre la note.
6. L'API met à jour le score de la commande dans la base de données en se basant sur l'identifiant de la commande.
7. La base de données retourne les données mises à jour à l'API.
8. L'API répond à l'interface avec un statut HTTP 200 pour confirmer la mise à jour.
9. L'interface affiche un message confirmant que le e-commerçant a bien été noté.

— **5. Noter un produit**

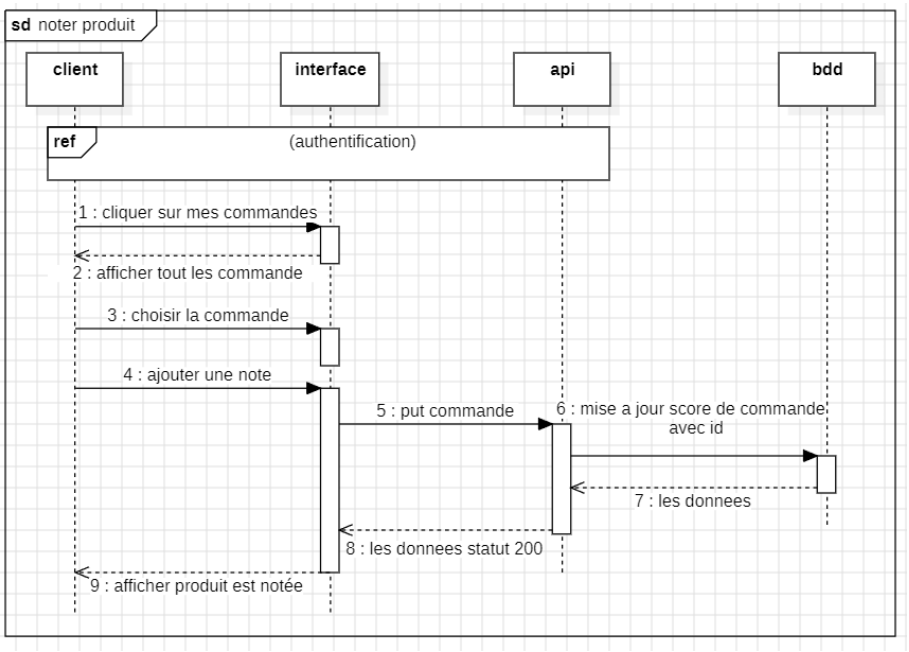


FIGURE 3.37 : Diagramme de séquence - Notation d'un produit

1. Le client clique sur « Mes commandes » dans l'interface
2. L'interface affiche la liste des commandes passées
3. Le client sélectionne une commande spécifique
4. Le client ajoute une note pour le produit
5. L'interface envoie une requête `PUT commande` à l'API
6. L'API met à jour le score du produit dans la base de données en utilisant l'ID de commande
7. La base de données retourne les données mises à jour
8. L'API renvoie un statut HTTP 200 à l'interface
9. L'interface affiche une confirmation que le produit a été noté

— **6. Modifier une commande**

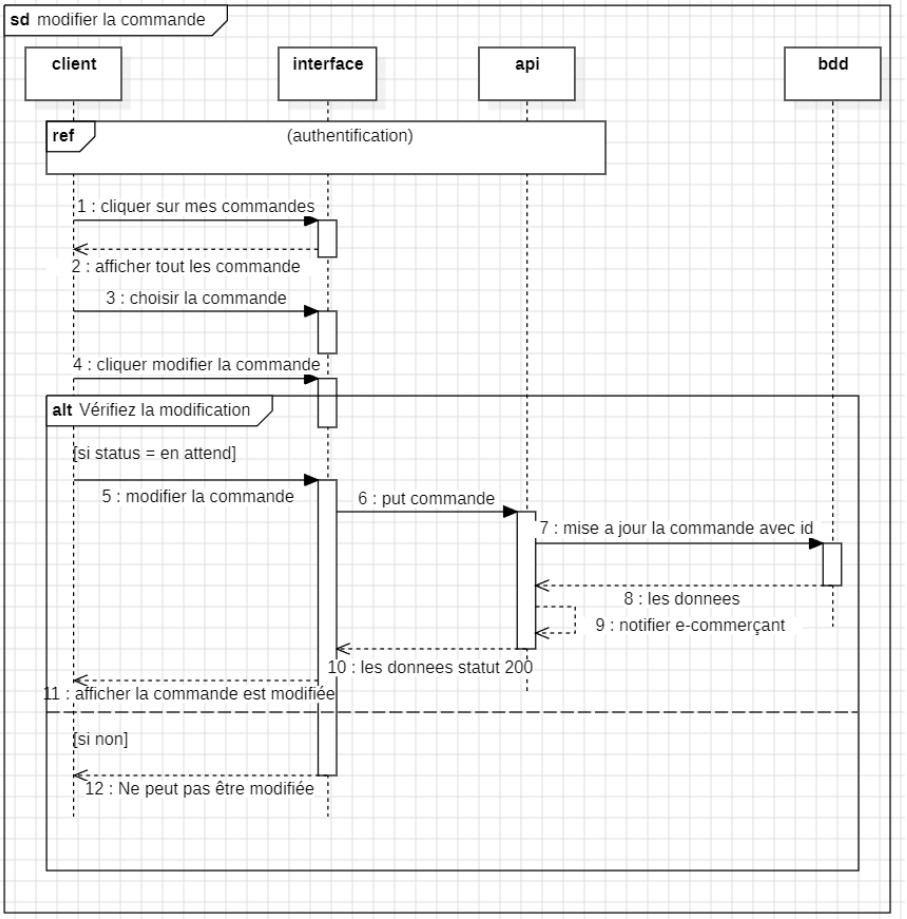


FIGURE 3.38 : Diagramme de séquence - Modification d'une commande

1. Le client clique sur « Mes commandes » dans l'interface
2. L'interface affiche la liste des commandes passées
3. Le client sélectionne une commande spécifique
4. Le client clique sur « Modifier la commande »

   Scénario principal (statut = "En attente")

   5. Le client modifie la commande
   6. L'interface envoie une requête `PUT commande` à l'API
   7. L'API met à jour la commande dans la base de données
   8. La base de données retourne les données mises à jour
   9. L'API notifie l'e-commerçant de la modification
   10. L'API renvoie un statut HTTP 200 à l'interface
   11. L'interface affiche « Commande modifiée avec succès »

   Scénario alternatif (statut $\neq$ "*Enattente*")

   12. L'interface affiche « La commande ne peut plus être modifiée »

7. Modifier les produits

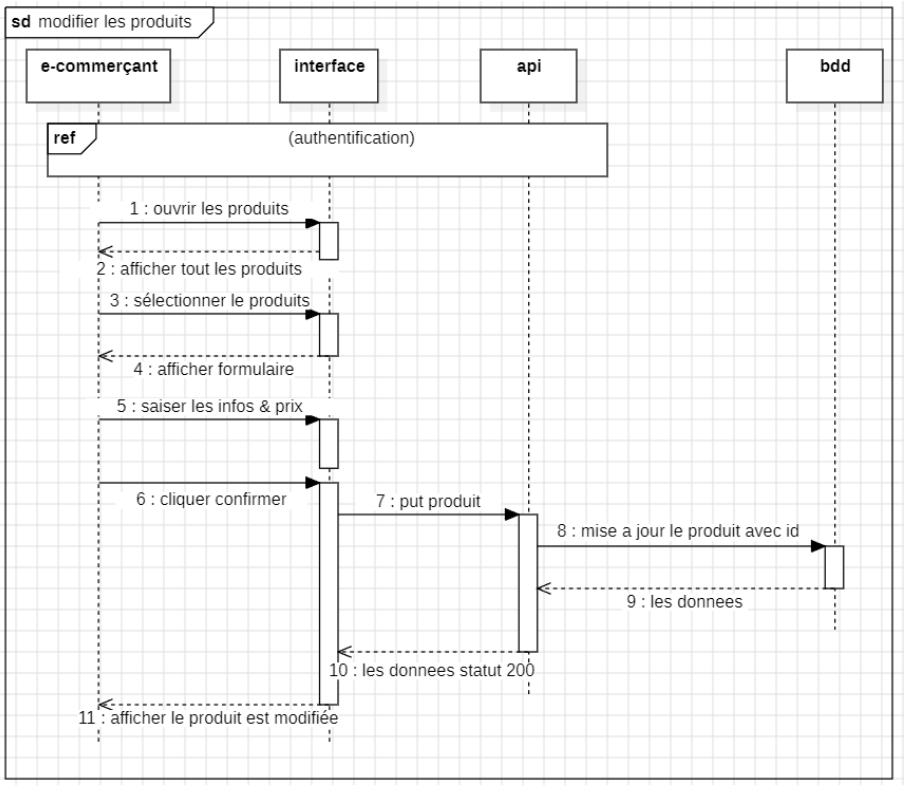


FIGURE 3.39 : Diagramme de séquence - Modification des produits

1. L'e-commerçant ouvre la section « Produits » dans l'interface
2. L'interface affiche la liste complète des produits
3. L'e-commerçant sélectionne un produit spécifique
4. L'interface affiche le formulaire de modification
5. L'e-commerçant saisit les nouvelles informations et prix
6. L'e-commerçant clique sur « Confirmer »
7. L'interface envoie une requête `PUT /produits/{id}` à l'API
8. L'API met à jour le produit dans la base de données
9. La base de données retourne les données mises à jour
10. L'API renvoie un statut HTTP 200 à l'interface
11. L'interface affiche « Produit modifié avec succès »

8. Ajouter un produit

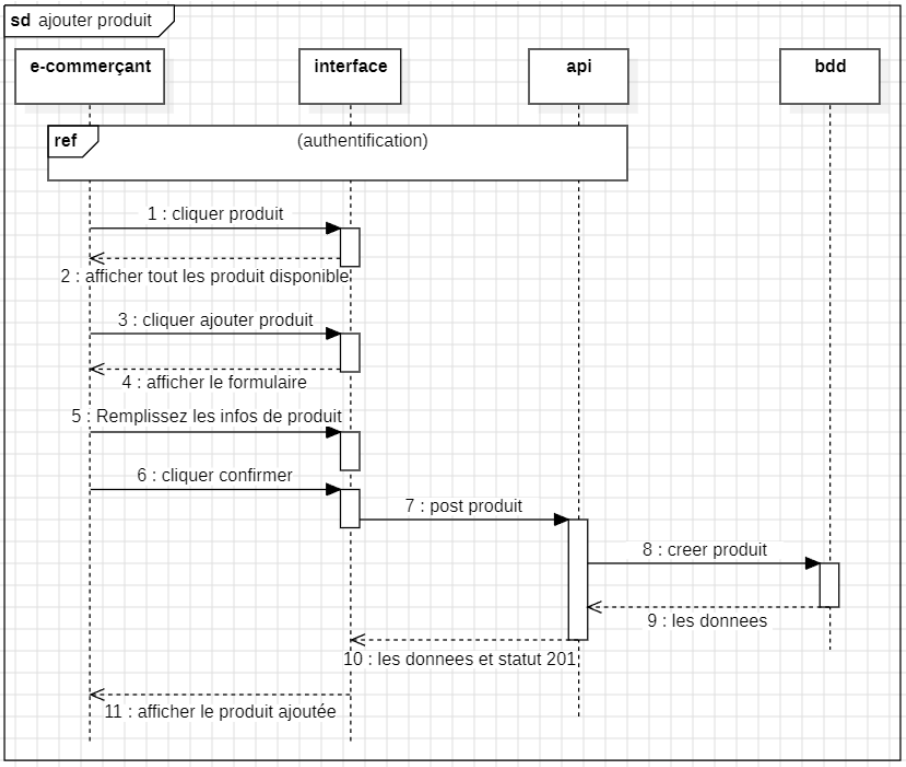


FIGURE 3.40 : Diagramme de séquence - Ajout d'un produit

1. L'e-commerçant clique sur « Produits » dans l'interface
2. L'interface affiche tous les produits disponibles
3. L'e-commerçant clique sur « Ajouter un produit »
4. L'interface affiche le formulaire d'ajout
5. L'e-commerçant remplit les informations du produit (nom, description, prix, etc.)
6. L'e-commerçant clique sur « Confirmer »
7. L'interface envoie une requête `POST /produits` à l'API
8. L'API crée le nouveau produit dans la base de données
9. La base de données retourne les données du produit créé
10. L'API renvoie les données avec un statut HTTP 201 (Created)
11. L'interface affiche « Produit ajouté avec succès »

9. Modifier les informations du magasin

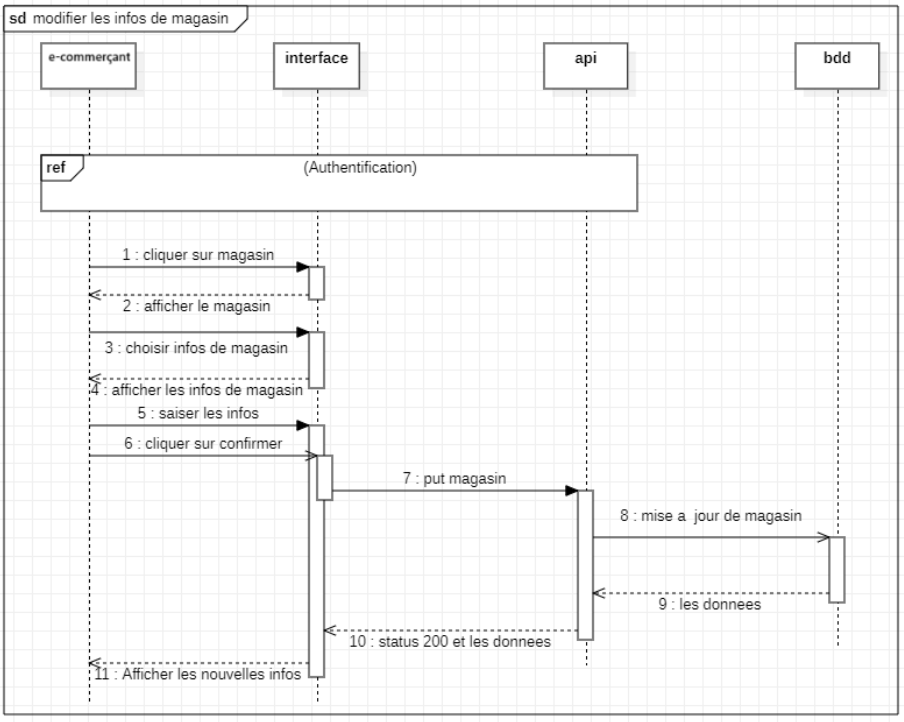


FIGURE 3.41 : Diagramme de séquence - Modification des informations du magasin

1. L'e-commerçant clique sur « Magasin » dans l'interface
2. L'interface affiche les informations actuelles du magasin
3. L'e-commerçant sélectionne « Modifier les informations »
4. L'interface affiche le formulaire avec les informations existantes
5. L'e-commerçant saisit les nouvelles informations (nom, adresse, etc.)
6. L'e-commerçant clique sur « Confirmer »
7. L'interface envoie une requête `PUT /magasin` à l'API
8. L'API met à jour les informations du magasin dans la base de données
9. La base de données retourne les données mises à jour
10. L'API renvoie un statut HTTP 200 avec les données actualisées
11. L'interface affiche les nouvelles informations du magasin

10. Confirmer une commande

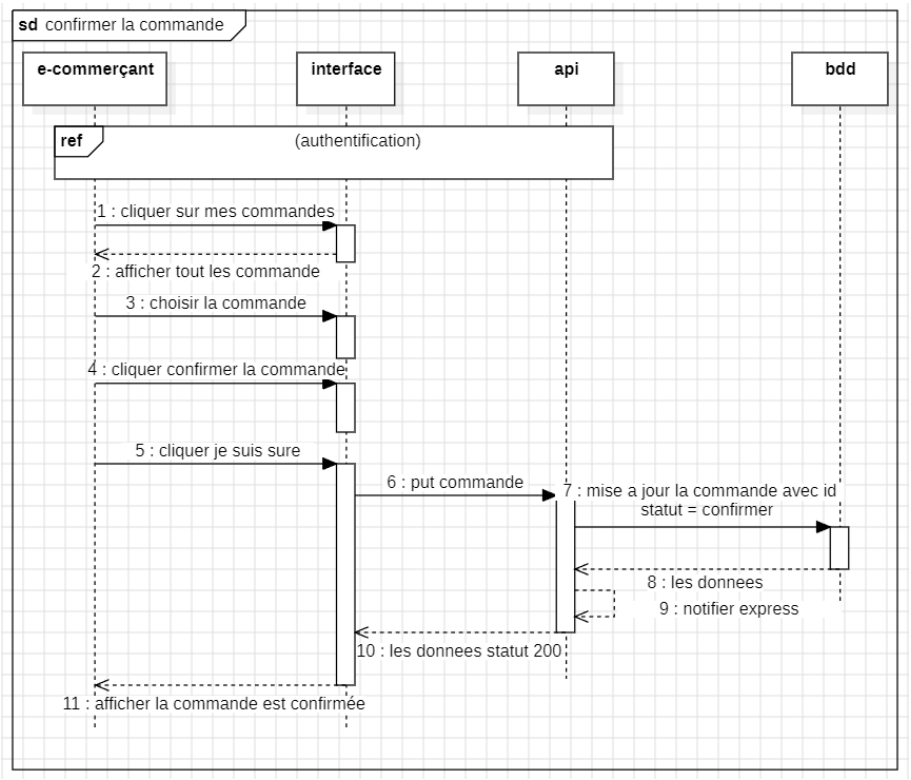


FIGURE 3.42 : Diagramme de séquence - Confirmation d'une commande

1. L'e-commerçant clique sur « Mes commandes » dans l'interface
2. L'interface affiche la liste des commandes
3. L'e-commerçant sélectionne une commande spécifique
4. L'e-commerçant clique sur « Confirmer la commande »
5. L'interface demande confirmation (« Je suis sûr »)
6. L'e-commerçant confirme son choix
7. L'interface envoie une requête `PUT /commandes/{id}` à l'API
8. L'API met à jour le statut de la commande à « confirmée » dans la base de données
9. La base de données retourne les données mises à jour
10. L'API notifie le service de livraison (Express)
11. L'API renvoie un statut HTTP 200 avec les données actualisées
12. L'interface affiche « Commande confirmée avec succès »

11. Vérifier une commande

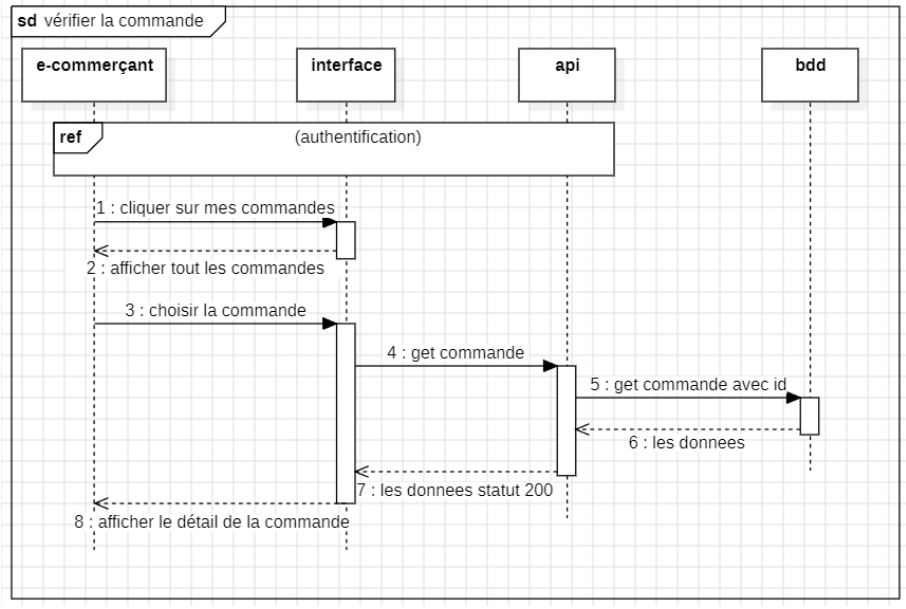


FIGURE 3.43 : Diagramme de séquence - Vérification d'une commande

1. L'e-commerçant clique sur « Mes commandes » dans l'interface
2. L'interface affiche la liste complète des commandes
3. L'e-commerçant sélectionne une commande spécifique à vérifier
4. L'interface envoie une requête `GET /commandes` à l'API
5. L'API récupère la commande spécifique avec son ID dans la base de données
6. La base de données retourne les données complètes de la commande
7. L'API renvoie un statut HTTP 200 avec les données de la commande
8. L'interface affiche le détail complet de la commande sélectionnée

3.2.5 Diagramme de cas d'utilisation du service des Artisans

Le diagramme ci-dessous présente les différents cas d'utilisation du service des artisans dans le système Mefria. Il met en évidence les interactions entre les différents acteurs (clients, artisans, administrateurs) et le système, en précisant les fonctionnalités accessibles à chacun. Ce schéma permet de visualiser les rôles et responsabilités liés à la gestion des missions, des services artisanaux ainsi qu'au suivi des activités :

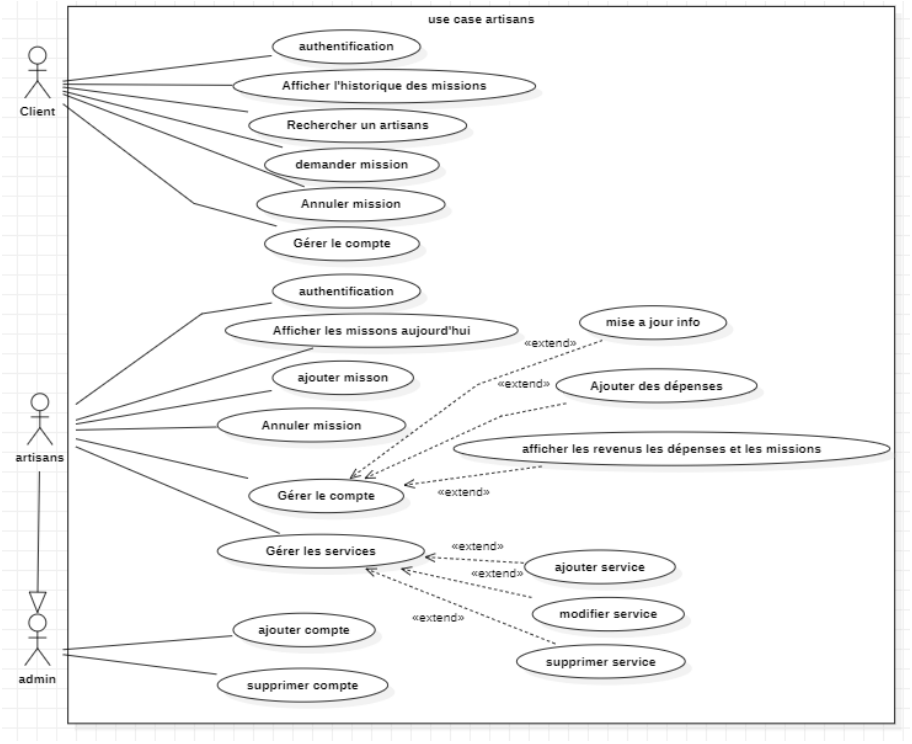


FIGURE 3.44 : Diagramme de cas d'utilisation du service Artisan

Remarque : L'authentification est requise pour toutes les opérations du système, comme indiqué par la référence `(Authentification)` dans chaque diagramme de séquence.

Acteurs identifiés :

— **Client** : utilisateur qui demande et suit les missions artisanales.

— **Artisan** : professionnel qui propose et réalise des services artisanaux.

— **Admin** : administrateur ayant accès à l'ensemble des fonctions de gestion.

Principaux cas d'utilisation :

— Authentification (tous les rôles)

— Afficher l'historique des missions (artisan/client)

— Rechercher un artisan par compétence (client)

— Demander/annuler une mission (client)

— Afficher les missions du jour (artisan)

— Ajouter/annuler une mission (artisan)

— Mise à jour des informations professionnelles (artisan)

— Gérer les services proposés (artisan)

— Ajouter/supprimer un compte (admin)

— Suivi des revenus et dépenses (artisan)

Ce diagramme offre une vue d'ensemble des fonctionnalités du service Artisan de Mefria, en illustrant les interactions entre les utilisateurs et les fonctions de gestion des missions artisanales. Il sert de base aux spécifications techniques détaillées à venir.

Descriptions textuelles et diagrammes de séquence des cas d'utilisation

1. Afficher l'historique des missions

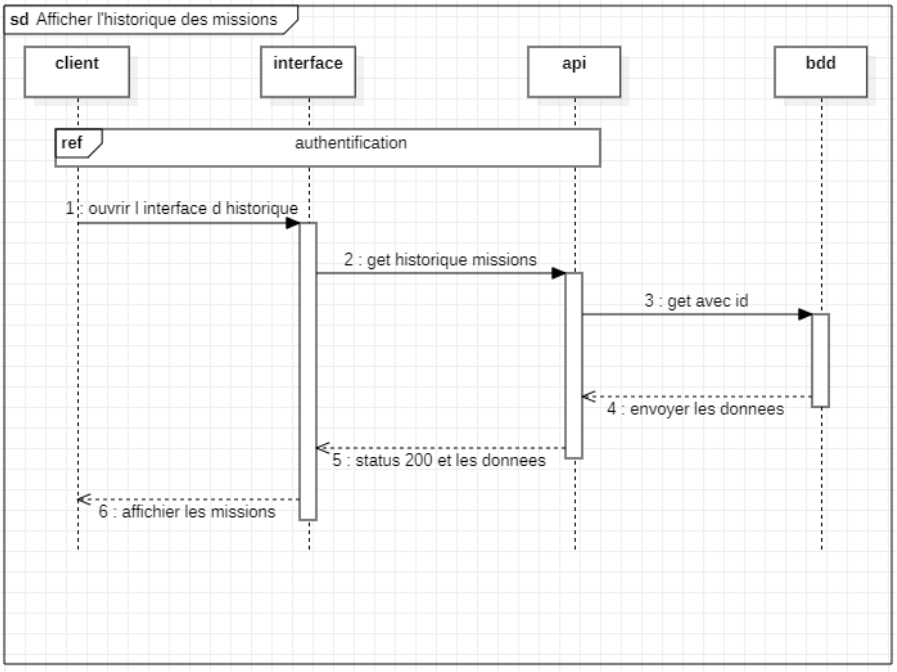


FIGURE 3.45 : Diagramme de séquence - Affichage de l'historique des missions

1. Le client ouvre l'interface d'historique dans l'application
2. L'interface envoie une requête `GET /historique/missions` à l'API
3. L'API récupère l'historique avec l'ID client dans la base de données
4. La base de données envoie les données historiques à l'API
5. L'API retourne un statut HTTP 200 avec les données de l'historique
6. L'interface affiche la liste des missions passées avec leurs détails

2. Rechercher un artisan

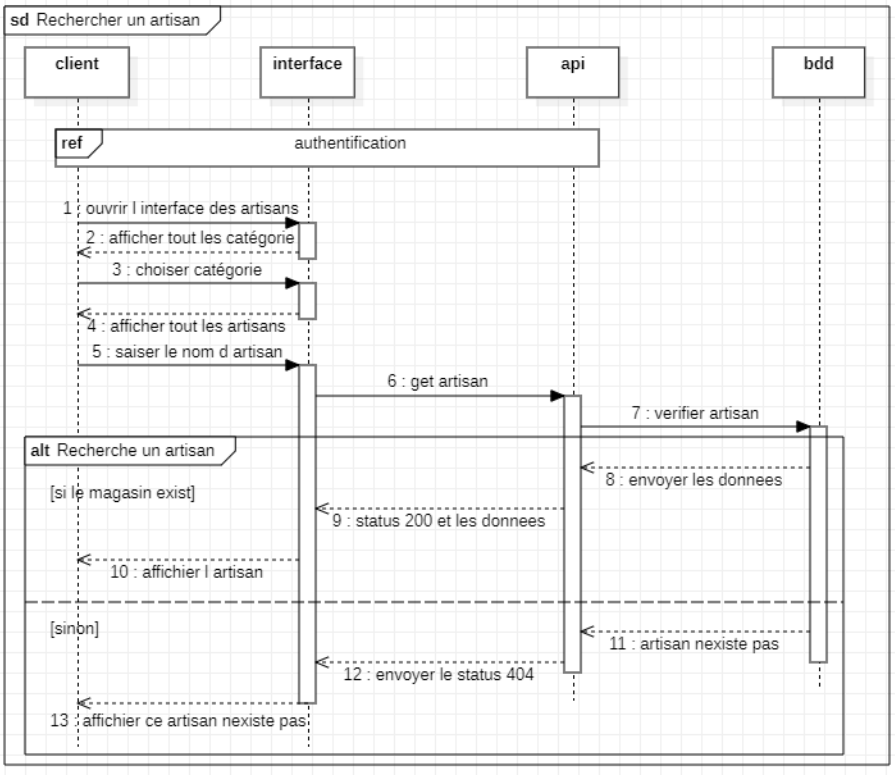


FIGURE 3.46 : Diagramme de séquence - Recherche d'un artisan

1. Le client ouvre l'interface des artisans
2. L'interface affiche toutes les catégories disponibles
3. Le client choisit une catégorie spécifique
4. L'interface affiche tous les artisans de cette catégorie
5. Le client saisit le nom d'un artisan
6. L'interface envoie une requête `GET /artisans` à l'API
7. L'API vérifie l'existence de l'artisan dans la base de données

   Scénario principal (artisan existe) :

   8. La base de données envoie les données de l'artisan
   9. L'API retourne un statut HTTP 200 avec les données
   10. L'interface affiche le profil de l'artisan trouvé

   Scénario alternatif (artisan n'existe pas) :

   12. L'API retourne un statut HTTP 404 (Not Found)
   13. L'interface affiche "Cet artisan n'existe pas"

3. Demander une mission

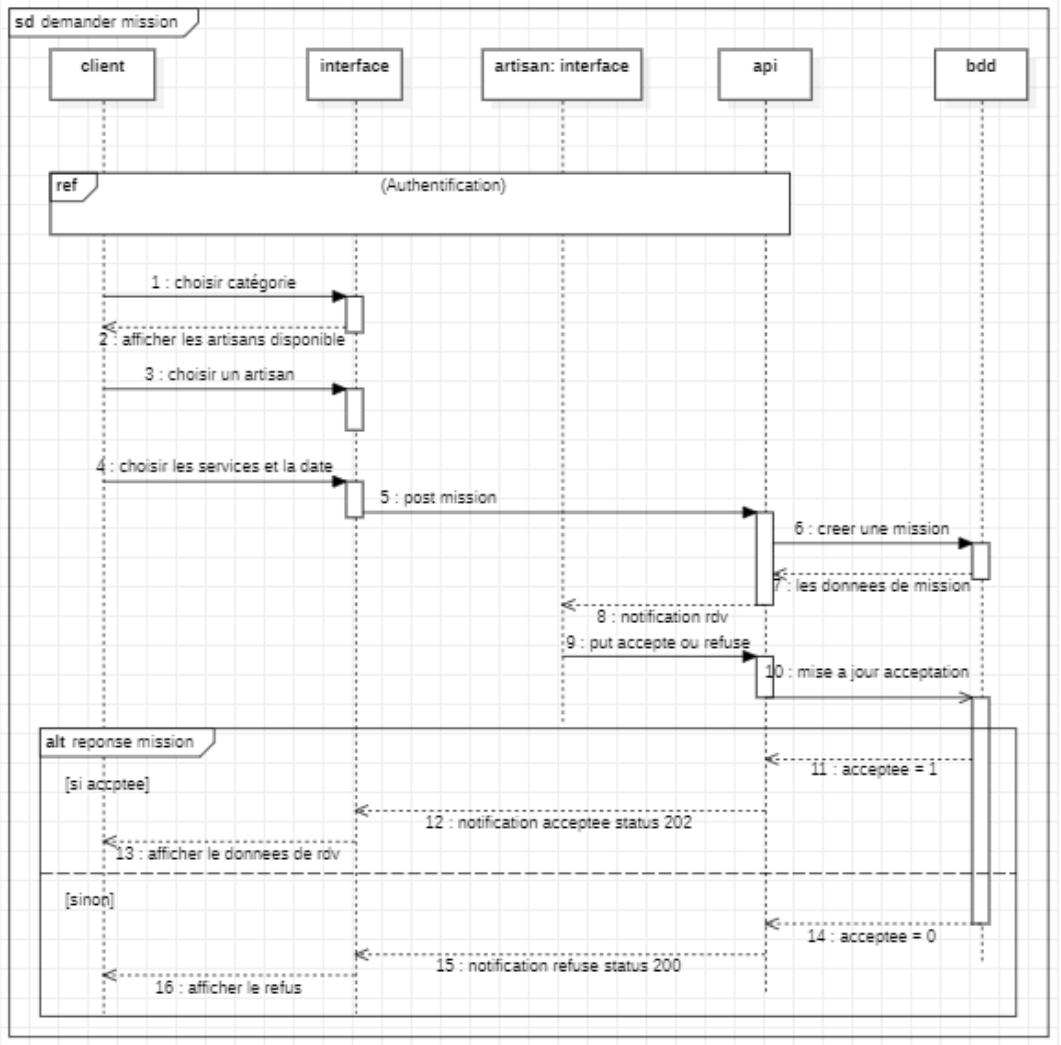


FIGURE 3.47 : Diagramme de séquence - Demande de mission

1. Le client choisit une catégorie de service
2. L'interface affiche les artisans disponibles
3. Le client sélectionne un artisan spécifique
4. Le client choisit les services et la date souhaitée
5. L'interface envoie une requête `POST /missions` à l'API
6. L'API crée une nouvelle mission dans la base de données
7. La base de données retourne les données de la mission créée
8. L'API envoie une notification de rendez-vous à l'interface artisan
9. L'artisan accepte ou refuse via son interface (`PUT /missions/{id}/statut`)
10. L'API met à jour le statut d'acceptation dans la base de données

    Scénario principal (mission acceptée) :

    11. Notification envoyée avec statut HTTP 202 (Accepted)
    12. L'interface client affiche les détails du rendez-vous confirmé

Scénario alternatif (mission refusée) :

13. Notification envoyée avec statut HTTP 200 (OK)
14. L'interface client affiche le message de refus
15. Statut d'acceptation mis à 0 (refusé)

4. Annuler une mission (client

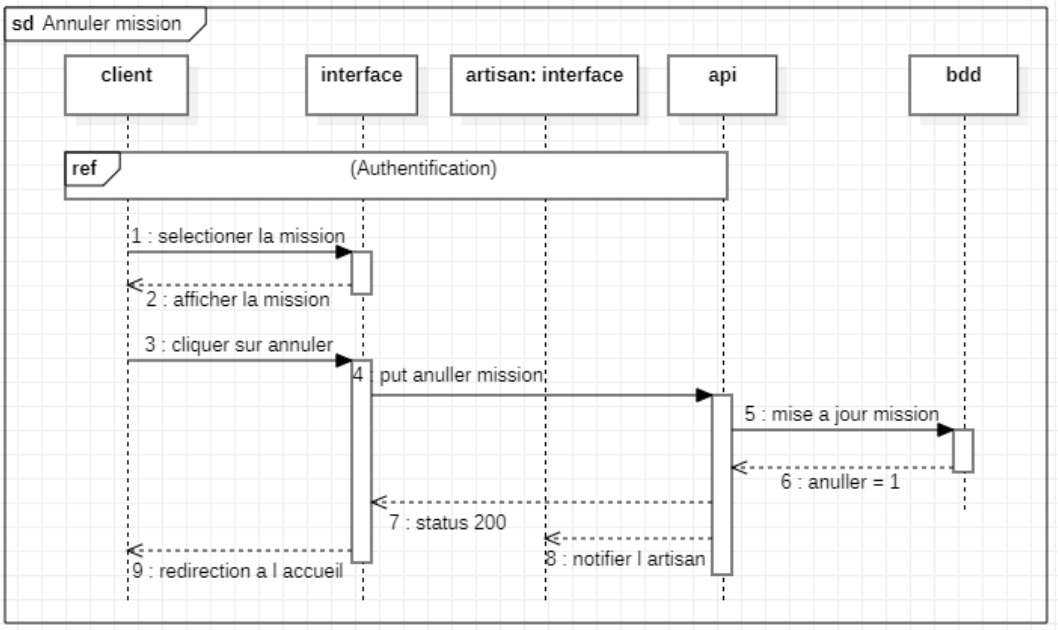


FIGURE 3.48 : Diagramme de séquence - Annulation d'une mission par le client

1. Le client sélectionne une mission dans son historique
2. L'interface affiche les détails de la mission sélectionnée
3. Le client clique sur le bouton "Annuler"
4. L'interface envoie une requête `PUT /missions/{id}/annuler` à l'API
5. L'API met à jour le statut de la mission dans la base de données
6. La base de données marque la mission comme annulée (annuler = 1)
7. L'API retourne un statut HTTP 200 (OK)
8. L'API envoie une notification à l'interface de l'artisan concerné
9. L'interface client redirige vers la page d'accueil

5. Afficher les missions aujourd'hui

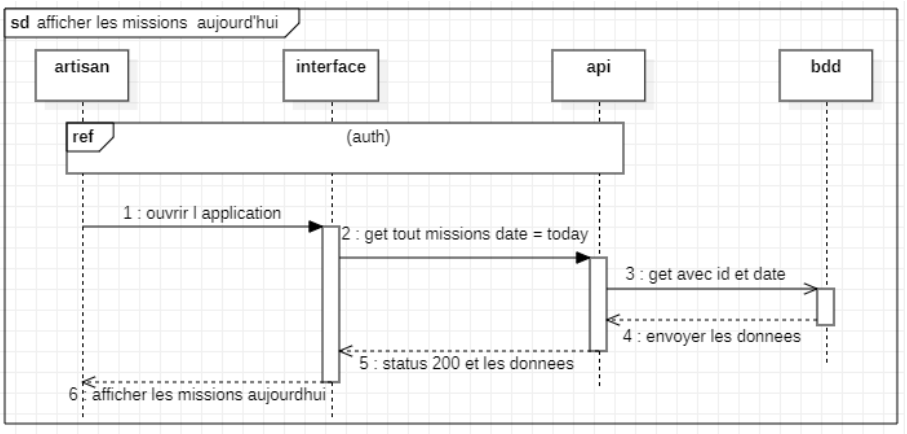


FIGURE 3.49 : Diagramme de séquence - Affichage des missions du jour

1. L'artisan ouvre l'application
2. L'interface envoie une requête `GET /missions?date=today` à l'API
3. L'API récupère les missions avec l'ID artisan et la date du jour dans la base de données
4. La base de données envoie les données des missions à l'API
5. L'API retourne un statut HTTP 200 avec les données des missions
6. L'interface affiche la liste des missions prévues pour aujourd'hui

6. Ajouter une mission (artisan)

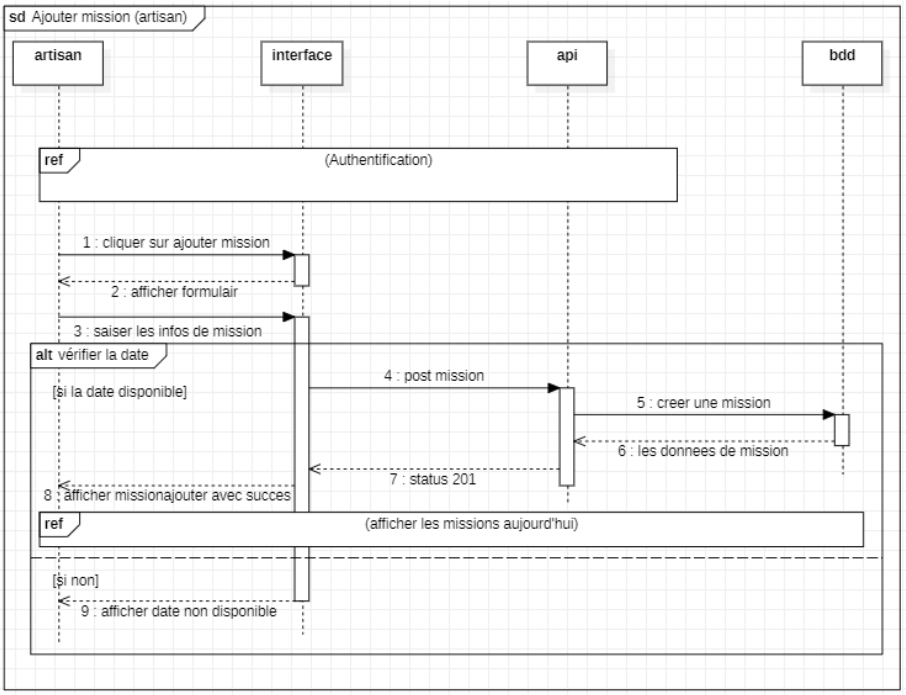


FIGURE 3.50 : Diagramme de séquence - Ajout d'une mission par l'artisan

1. L'artisan clique sur "Ajouter mission" dans l'interface
2. L'interface affiche le formulaire d'ajout
3. L'artisan saisit les informations de la mission (date, service, etc.)

   Scénario principal (date disponible) :

   4. L'interface envoie une requête `POST /missions` à l'API
   5. L'API crée la nouvelle mission dans la base de données
   6. La base de données retourne les données de la mission créée
   7. L'API renvoie un statut HTTP 201 (Created)
   8. L'interface affiche "Mission ajoutée avec succès"

   Scénario alternatif (date non disponible) :

   9. L'interface affiche "Date non disponible"

7. Annuler une mission (par artisan)

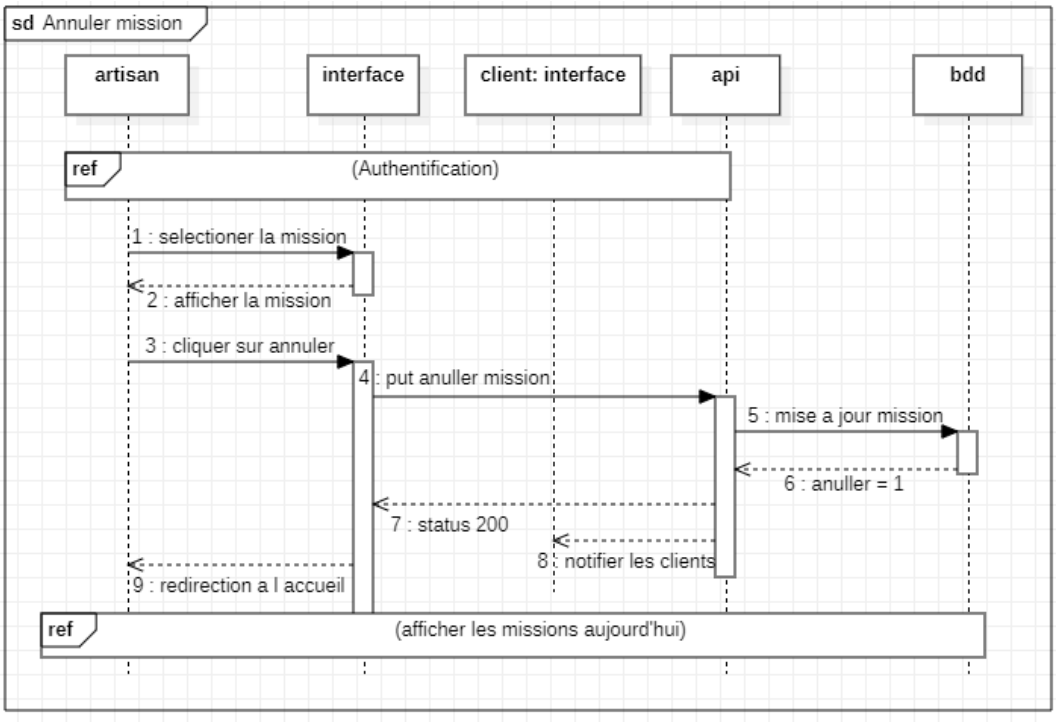


FIGURE 3.51 : Diagramme de séquence - Annulation d'une mission par l'artisan

1. L'artisan sélectionne une mission dans son interface
2. L'interface affiche les détails de la mission sélectionnée
3. L'artisan clique sur le bouton "Annuler"
4. L'interface envoie une requête `PUT /missions/{id}/annuler` à l'API
5. L'API met à jour le statut de la mission dans la base de données
6. La base de données marque la mission comme annulée (annuler = 1)
7. L'API retourne un statut HTTP 200 (OK)
8. L'API envoie une notification à l'interface du client concerné
9. L'interface artisan redirige vers la page d'accueil

8. Modifier les informations du compte

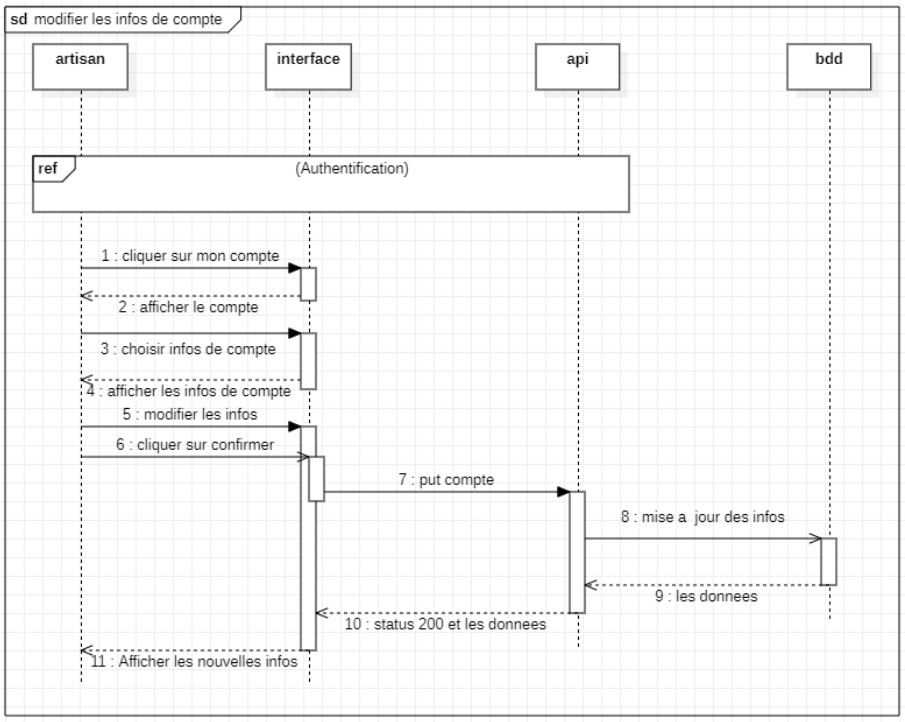


FIGURE 3.52 : Diagramme de séquence - Modification des informations du compte

1. L'artisan clique sur "Mon compte" dans l'interface
2. L'interface affiche les informations actuelles du compte
3. L'artisan sélectionne "Modifier les informations"
4. L'interface affiche le formulaire avec les informations existantes
5. L'artisan modifie les informations souhaitées
6. L'artisan clique sur "Confirmer"
7. L'interface envoie une requête `PUT /compte` à l'API
8. L'API met à jour les informations dans la base de données
9. La base de données retourne les données mises à jour
10. L'API renvoie un statut HTTP 200 avec les données actualisées
11. L'interface affiche les nouvelles informations du compte

9. Ajouter les dépenses

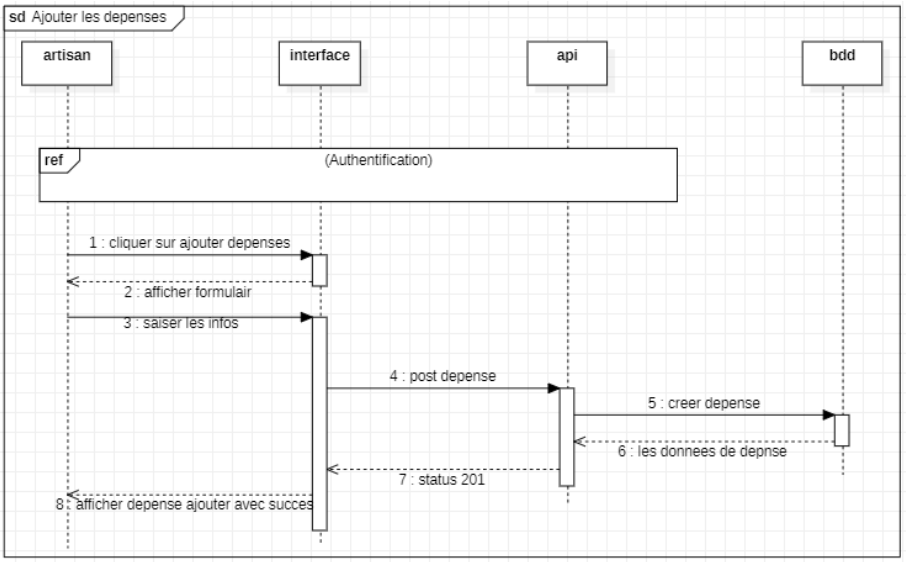


FIGURE 3.53 : Diagramme de séquence - Ajout de dépenses

1. L'artisan clique sur "Ajouter dépenses" dans l'interface
2. L'interface affiche le formulaire d'ajout de dépenses
3. L'artisan saisit les informations de la dépense (montant, catégorie, date, etc.)
4. L'interface envoie une requête `POST /depenses` à l'API
5. L'API crée une nouvelle dépense dans la base de données
6. La base de données retourne les données de la dépense créée
7. L'API renvoie un statut HTTP 201 (Created)
8. L'interface affiche "Dépense ajoutée avec succès"

10. Afficher les revenus, dépenses et missions

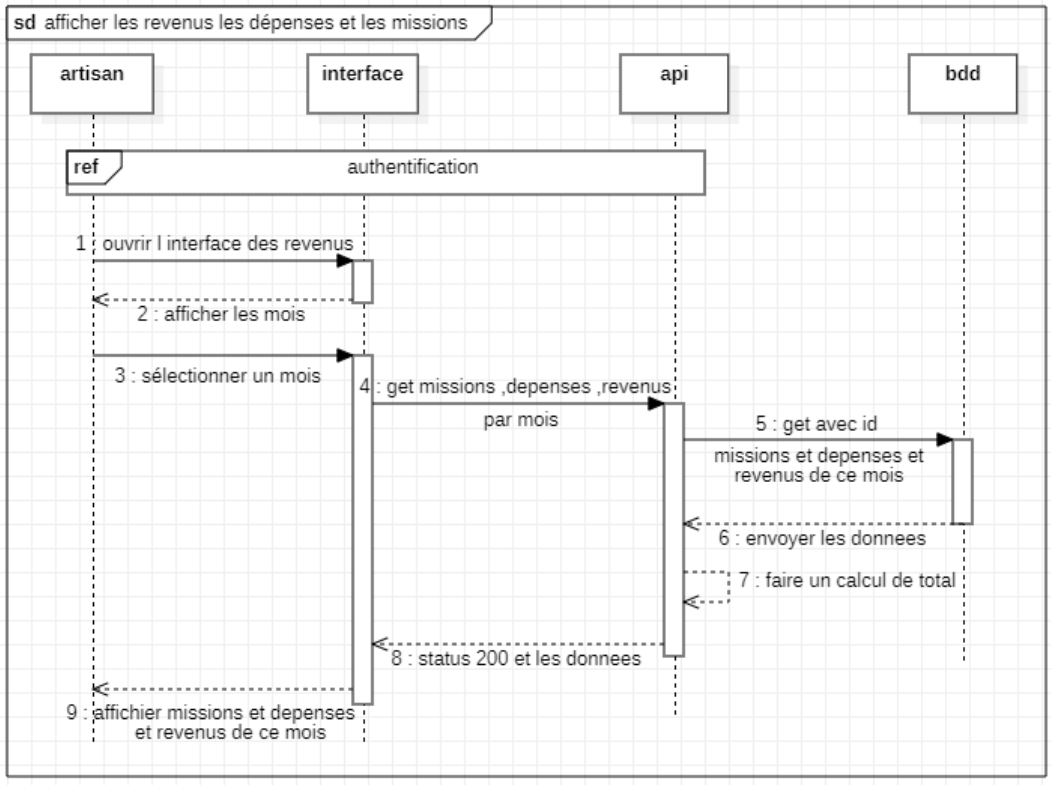


FIGURE 3.54 : Diagramme de séquence - Consultation des statistiques financières

1. L'artisan ouvre l'interface des revenus dans l'application
2. L'interface affiche la liste des mois disponibles
3. L'artisan sélectionne un mois spécifique
4. L'interface envoie une requête combinée `GET /stats?mois=X&type=missions,dépenses,revenus` à l'API
5. L'API récupère les données avec l'ID artisan et le mois sélectionné dans la base de données
6. La base de données envoie les données complètes (missions, dépenses, revenus) pour le mois demandé
7. L'API effectue les calculs des totaux et bilans
8. L'API retourne un statut HTTP 200 avec les données consolidées
9. L'interface affiche le rapport complet pour le mois sélectionné avec :
   - Liste des missions réalisées
   - Détail des dépenses
   - Calcul des revenus nets
   - Bilans et statistiques

11. Modifier les services

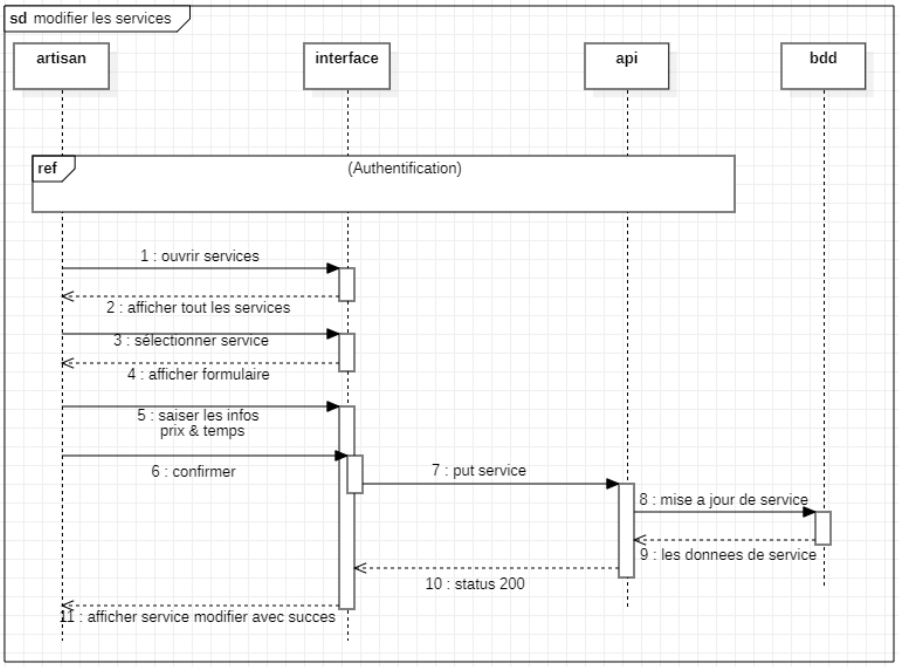


FIGURE 3.55 : Diagramme de séquence - Modification d'un service

1. L'artisan ouvre la section "Services" dans l'interface
2. L'interface affiche la liste complète des services existants
3. L'artisan sélectionne un service spécifique à modifier
4. L'interface affiche le formulaire de modification pré-rempli
5. L'artisan modifie les informations (prix, durée, description, etc.)
6. L'artisan clique sur "Confirmer" pour valider les modifications
7. L'interface envoie une requête `PUT /services/{id}` à l'API
8. L'API met à jour le service dans la base de données
9. La base de données retourne les données mises à jour du service
10. L'API renvoie un statut HTTP 200 (OK)
11. L'interface affiche "Service modifié avec succès"

12. Ajouter un service

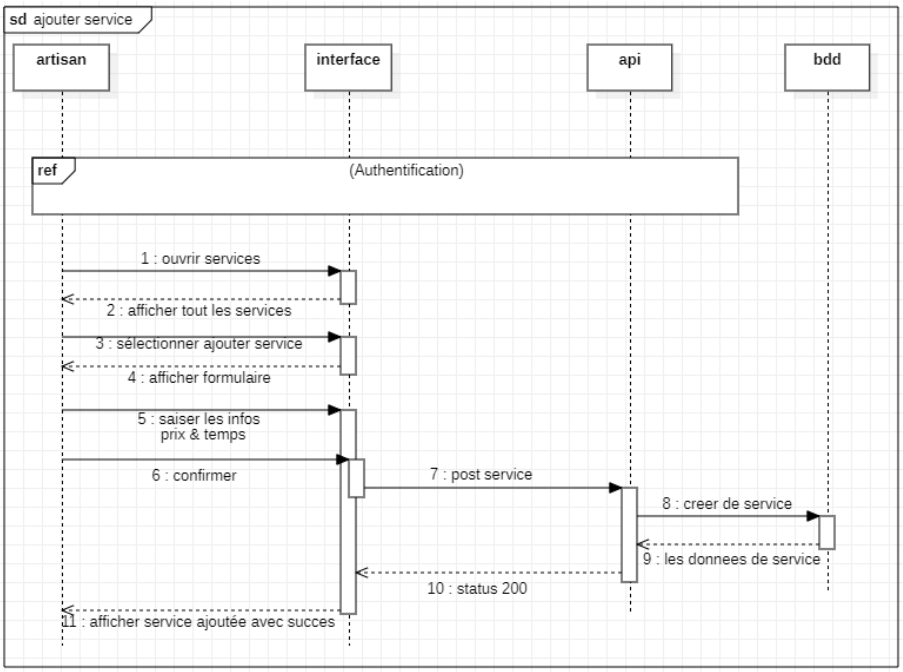


FIGURE 3.56 : Diagramme de séquence - Ajout d'un nouveau service

1. L'artisan ouvre la section "Services" dans l'interface
2. L'interface affiche la liste des services existants
3. L'artisan sélectionne "Ajouter un service"
4. L'interface affiche un formulaire vierge pour le nouveau service
5. L'artisan saisit les informations du service (nom, prix, durée, etc.)
6. L'artisan clique sur "Confirmer" pour valider la création
7. L'interface envoie une requête `POST /services` à l'API
8. L'API crée le nouveau service dans la base de données
9. La base de données retourne les données du service créé
10. L'API renvoie un statut HTTP 200 (OK)
11. L'interface affiche "Service ajouté avec succès"

13. Supprimer un service

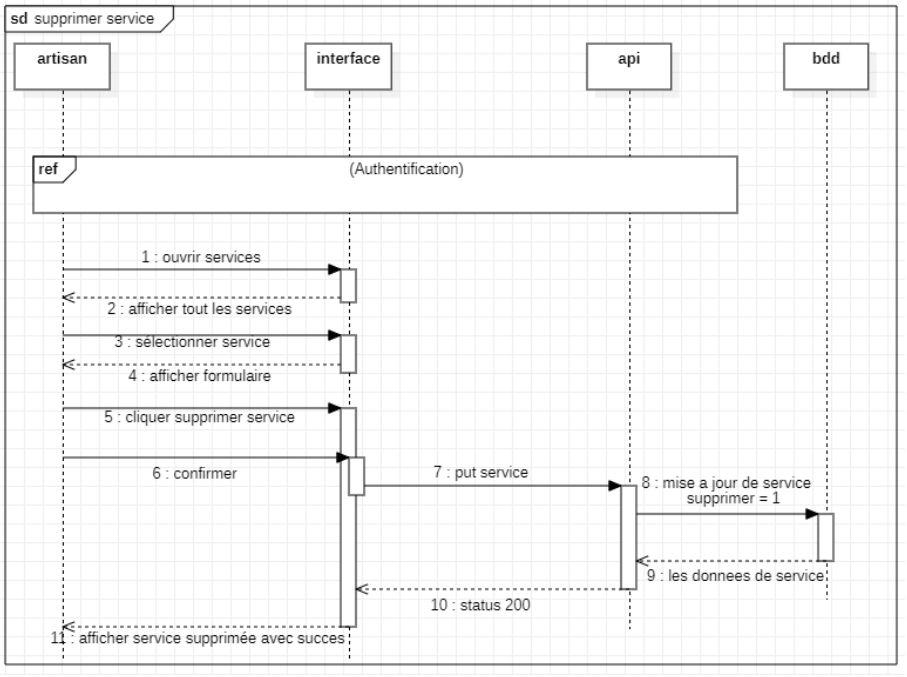


FIGURE 3.57 : Diagramme de séquence - Suppression d'un service

1. L'artisan ouvre la section "Services" dans l'interface
2. L'interface affiche la liste complète des services existants
3. L'artisan sélectionne un service spécifique
4. L'interface affiche les détails du service sélectionné
5. L'artisan clique sur "Supprimer le service"
6. L'interface demande et reçoit la confirmation
7. L'interface envoie une requête `PUT /services/{id}` à l'API avec `supprimer=1`
8. L'API met à jour le statut du service dans la base de données (supprimer=1)
9. La base de données retourne les données mises à jour du service
10. L'API renvoie un statut HTTP 200 (OK)
11. L'interface affiche "Service supprimé avec succès"

14. Ajouter un artisan

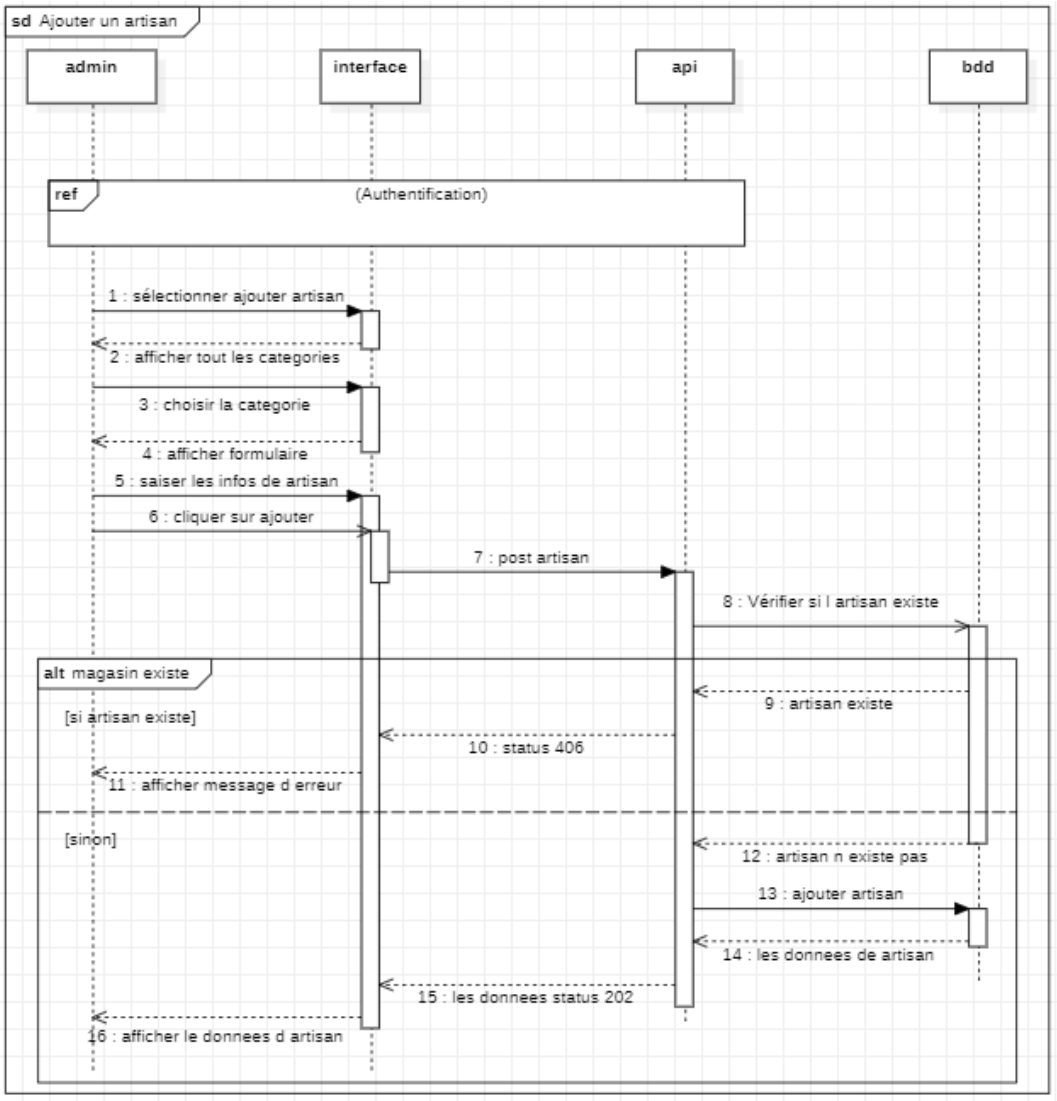


FIGURE 3.58 : Diagramme de séquence - Ajout d'un nouvel artisan

1. L'administrateur sélectionne "Ajouter un artisan" dans l'interface
2. L'interface affiche toutes les catégories disponibles
3. L'admin choisit une catégorie spécifique
4. L'interface affiche le formulaire d'ajout
5. L'admin saisit les informations de l'artisan
6. L'admin clique sur "Ajouter"
7. L'interface envoie une requête `POST /artisans` à l'API
8. L'API vérifie si l'artisan existe déjà

   Scénario alternatif (artisan existe déjà) :

   9. La base de données confirme l'existence de l'artisan
   10. L'API retourne un statut HTTP 406 (Not Acceptable)
   11. L'interface affiche "Cet artisan existe déjà"

Scénario principal (artisan n'existe pas) :

12. La base de données confirme que l'artisan n'existe pas
13. L'API crée le nouvel artisan dans la base de données
14. La base de données retourne les données de l'artisan créé
15. L'API renvoie un statut HTTP 202 (Accepted)
16. L'interface affiche les détails du nouvel artisan

15. Supprimer un artisan

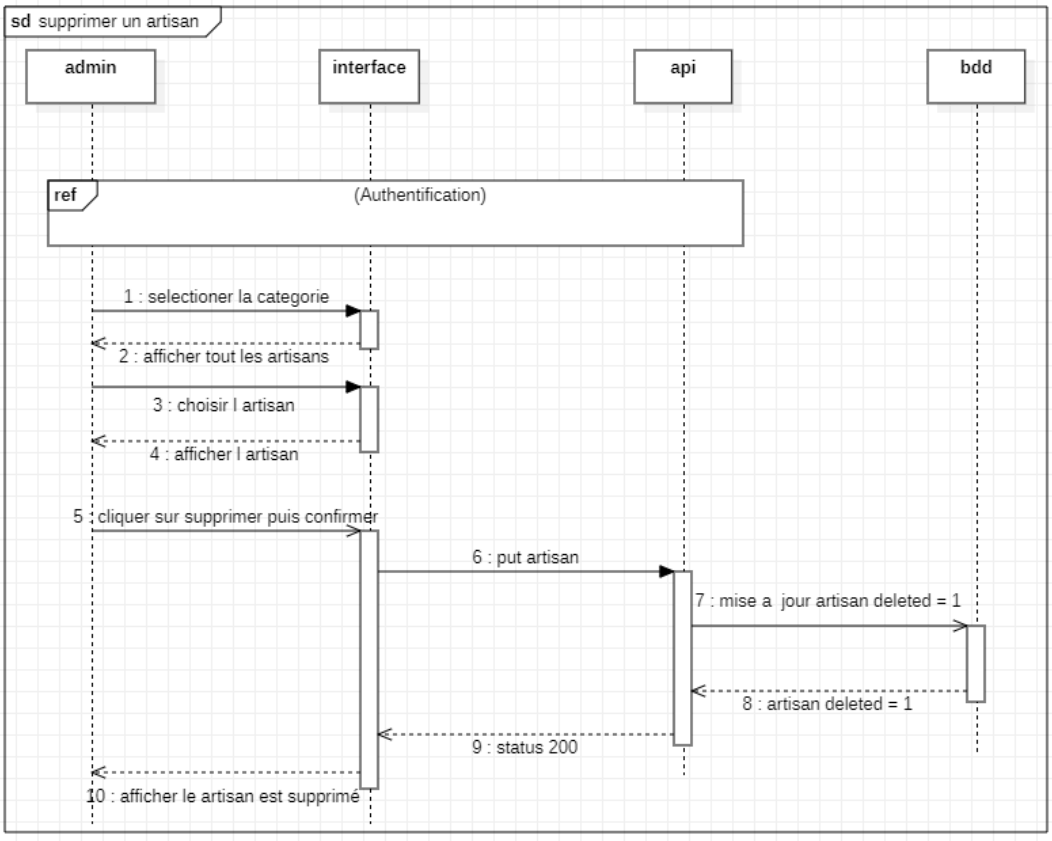


FIGURE 3.59 : Diagramme de séquence - Suppression d'un artisan

1. L'administrateur sélectionne une catégorie d'artisans
2. L'interface affiche tous les artisans de cette catégorie
3. L'admin choisit un artisan spécifique
4. L'interface affiche les détails de l'artisan sélectionné
5. L'admin clique sur "Supprimer" et confirme l'action
6. L'interface envoie une requête `PUT /artisans/{id}` à l'API
7. L'API met à jour le statut de l'artisan (deleted=1) dans la base de données
8. La base de données confirme la mise à jour (deleted=1)
9. L'API retourne un statut HTTP 200 (OK)
10. L'interface affiche "Artisan supprimé avec succès !"

3.2.6 Diagrammes de classes des services du système

Les diagrammes de classes suivants illustrent la structure des entités principales manipulées par les différents services de l'application MeFria. Ils montrent les attributs, les relations entre les entités, et permettent de mieux comprendre la conception orientée objet du système.

1. Module Rendez-vous

Ce diagramme illustre la structure du module Rendez-vous de la plateforme Mefria. Il permet d'organiser la prise de rendez-vous entre les clients et les fournisseurs de services, tout en intégrant les aspects liés au paiement, au suivi et à la planification.

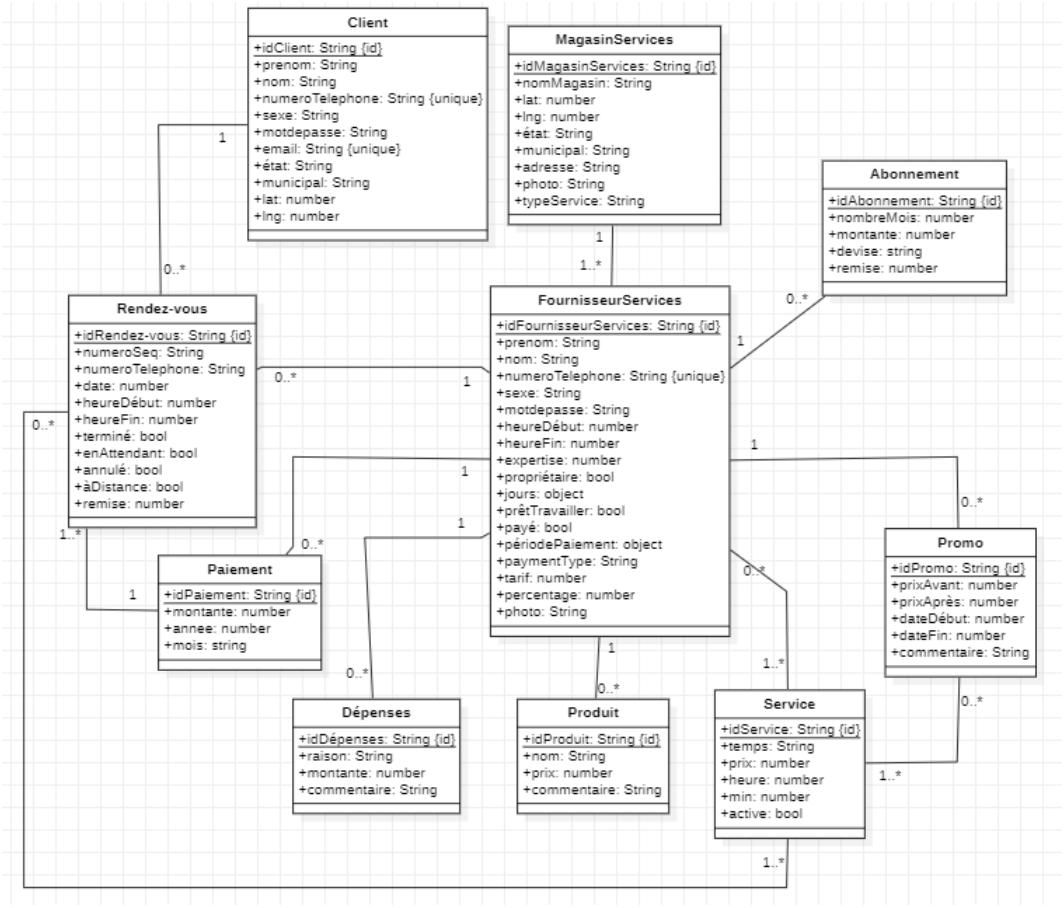


FIGURE 3.60 : Diagramme de classes - Module Rendez-vous

Chaque **Client** peut réserver plusieurs **Rendez-vous** auprès de différents **Fournisseurs-Services**. Ces derniers sont associés à un **MagasinServices** qui définit le cadre d'intervention (adresse, type de service, etc.).

Le **Rendez-vous** contient plusieurs attributs de planification comme la date, l'heure de début, l'heure de fin, ainsi que des indicateurs tels que l'état de fin, l'annulation, la présence d'un accompagnant, et la distance à parcourir. Une remise peut également être appliquée.

Chaque rendez-vous est lié à un **Paiement**, comprenant le montant, l'année et le mois de facturation. Ce module permet donc de gérer l'aspect financier directement depuis le processus de réservation.

Ce module s'intègre également avec d'autres éléments comme les **Abonnements** (pour

les clients réguliers) et les **Promotions** (appliquées aux services), tout en permettant un suivi structuré et professionnel des prestations entre clients et prestataires.

2. Module Livraison

Ce diagramme représente la structure du module Livraison de la plateforme Mefria. Il permet de gérer l'acheminement des commandes depuis les magasins jusqu'aux clients, en assurant le suivi des livreurs et des livraisons.

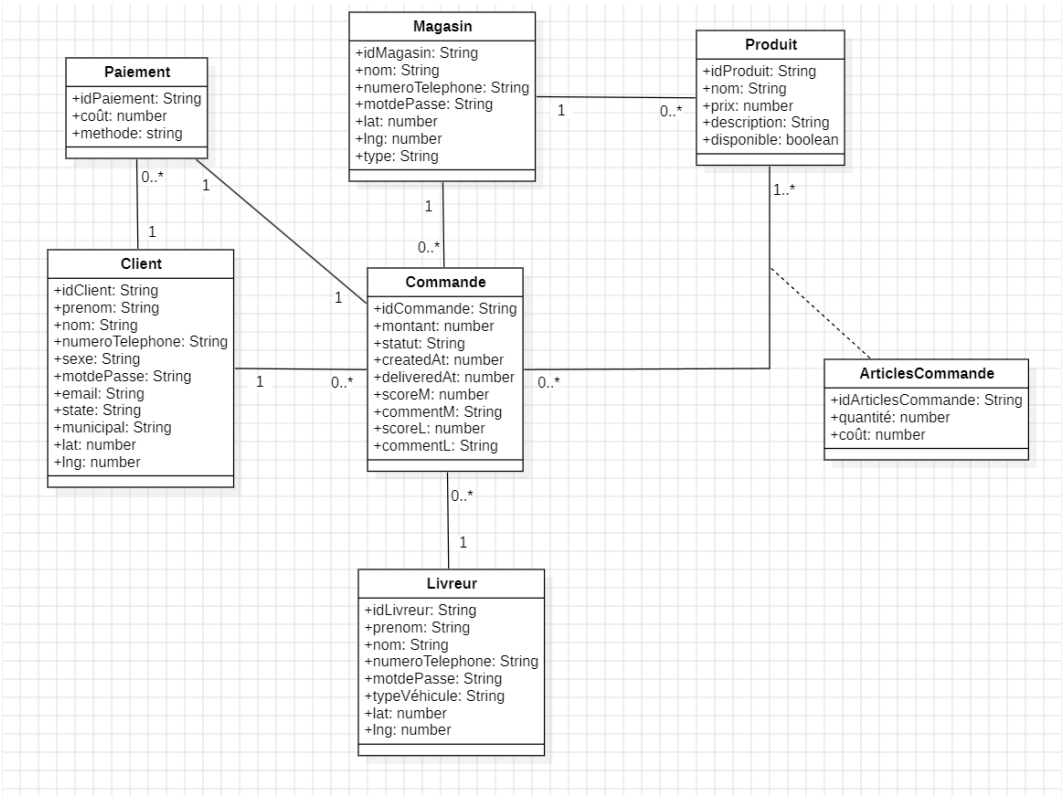


FIGURE 3.61 : Diagramme de classes - Module Livraison

Chaque commande peut être assignée à un livreur unique, tandis qu'un livreur peut prendre en charge plusieurs commandes. Le diagramme permet de suivre les informations du livreur (nom, prénom, type de véhicule, localisation, etc.) ainsi que l'état de livraison des commandes (date de livraison effective, notes et commentaires).

Ce module joue un rôle clé dans la logistique de la plateforme en garantissant une traçabilité complète et une évaluation de la qualité des livraisons.

3. Module E-commerce

Ce diagramme représente la structure du module E-commerce de la plateforme Mefria. Il permet de gérer les interactions entre les clients, les e-commerçants, les produits, les commandes et les paiements.

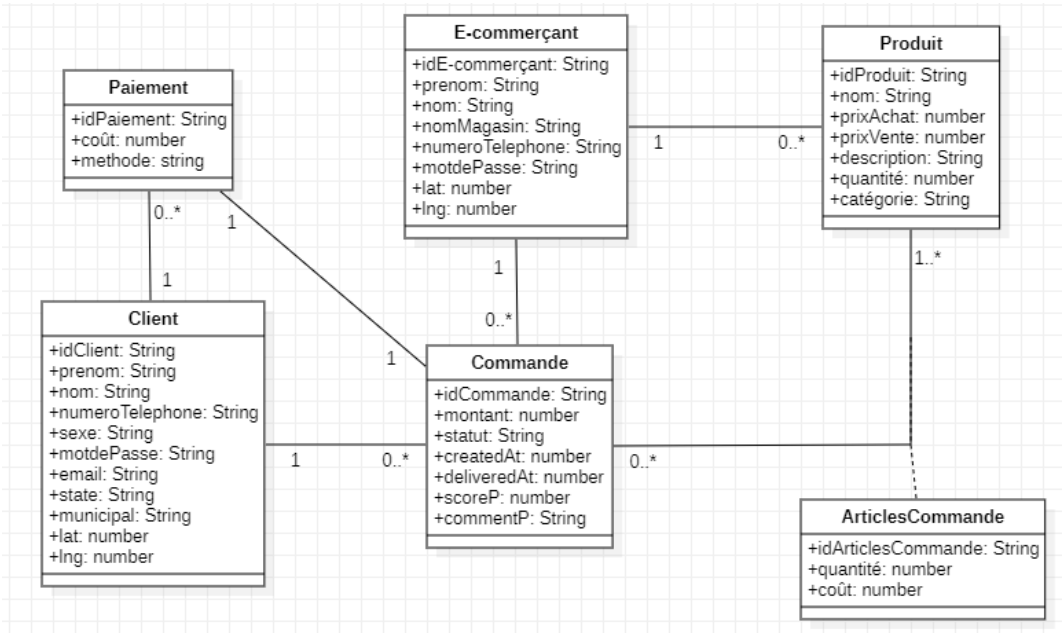


FIGURE 3.62 : Diagramme de classes - Module E-commerce

Chaque client peut effectuer plusieurs commandes, chacune étant associée à un ou plusieurs produits via la classe ArticlesCommande. Les commandes sont passées auprès d'un e-commerçant et peuvent être réglées à l'aide de divers moyens de paiement. Le diagramme intègre également les informations nécessaires à la gestion des produits (prix, stock, description, etc.) ainsi que les statuts et délais des commandes.

4. Module Artisans

Le diagramme suivant modélise la gestion des artisans dans le système Mefria. Il inclut les entités essentielles comme les artisans eux-mêmes, les missions qu'ils exécutent, les services proposés, ainsi que les dépenses associées à leur activité.

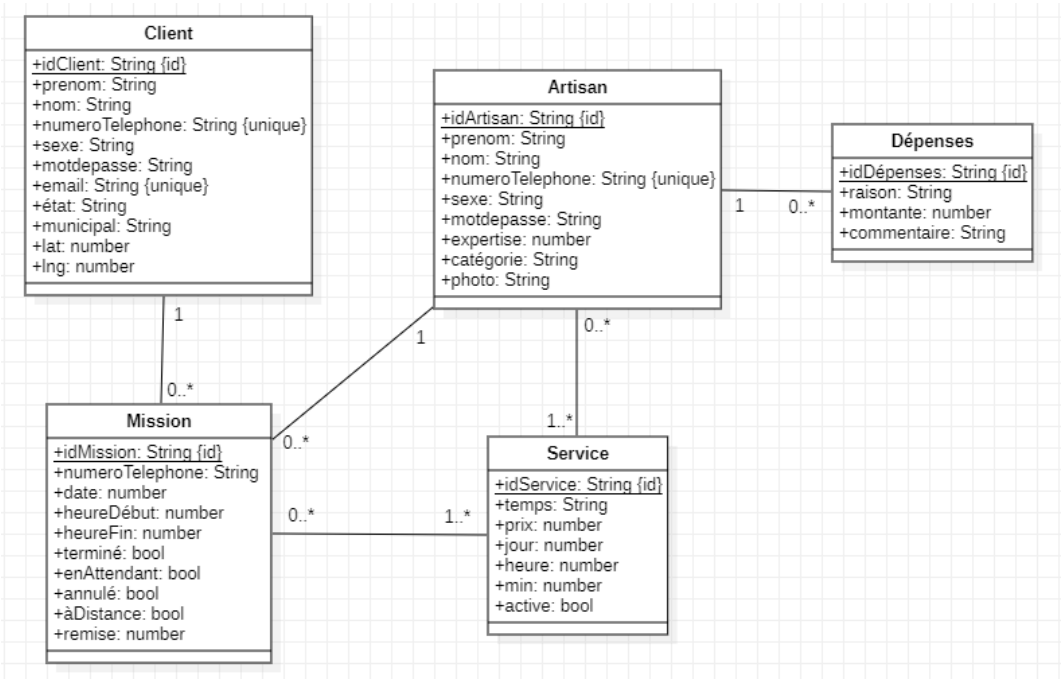


FIGURE 3.63 : Diagramme de classes - Module Artisans

Chaque artisan est associé à plusieurs missions, chacune étant liée à un client. Les missions sont caractérisées par des horaires, des statuts (terminé, annulé, en attente), et des remises. Un artisan peut aussi proposer plusieurs services avec des détails comme le tarif, la durée, et l'activité. Le diagramme prend également en compte les dépenses déclarées par les artisans, liées à leurs missions ou à leur activité générale.

3.2.7 Conclusion

Ce chapitre a permis d'établir les fondations de la plateforme Mefria à travers la spécification détaillée des besoins fonctionnels et non fonctionnels du système. Une fois les attentes clairement identifiées, nous avons procédé à la modélisation du système en utilisant le formalisme UML.

Les diagrammes de cas d'utilisation ont offert une vision globale des interactions entre les utilisateurs (clients, fournisseurs, livreurs, administrateurs) et les différents modules du système : rendez-vous, livraison et e-commerce. Chaque cas d'utilisation a été conçu pour refléter fidèlement les scénarios métiers et les services proposés par la plateforme.

Enfin, les diagrammes de classes ont permis de détailler la structure interne de chaque module, en identifiant les entités principales, leurs attributs, ainsi que les relations entre elles. Cette étape de conception fournit une base solide pour le développement technique, en assurant la cohérence et l'évolutivité de l'architecture logicielle.

Ainsi, ce chapitre constitue une étape clé dans la transition entre l'analyse des besoins et la phase de réalisation du projet.

Chapitre 4

Réalisation et implémentation de l'application Mefria

Introduction

Ce chapitre présente les aspects techniques de l'implémentation de l'application Mefria. Il détaille l'environnement de développement, les choix technologiques, l'architecture logicielle adoptée ainsi que le processus de mise en œuvre côté client et côté serveur. L'objectif est de montrer comment les composants logiciels interagissent pour répondre aux besoins spécifiés dans la phase de conception.

4.1 Environnement de développement

4.1.1 Outils et technologies utilisés

— **Node.js** : environnement d'exécution JavaScript côté serveur [4].

— **Express.js** : framework web minimaliste pour Node.js, utilisé pour développer des API REST [5].

— **CouchDB** : base de données NoSQL orientée documents, supportant la synchronisation et la réplication [6].

— **Socket.IO** : bibliothèque JavaScript permettant des communications bidirectionnelles en temps réel via WebSocket [7].

— **HTTPS** : protocole sécurisé utilisé pour chiffrer les échanges entre client et serveur [8].

— **SolidJS** : bibliothèque JavaScript déclarative et performante pour le frontend [9].

— **Capacitor** : plateforme pour la création d'applications mobiles natives à partir de technologies web [10].

— **BeerCSS** : framework CSS inspiré de Material Design 3 basé sur Web Components [11].

— **Material Design 3** : système de design de Google, utilisé pour assurer une interface cohérente et accessible [12].

— **StarUML** : outil de modélisation UML utilisé pour la conception des diagrammes [13].

— **Thunder Client** : extension pour Visual Studio Code permettant de tester facilement des API REST [14].

— **Git** : système de gestion de versions distribué [15] .

— **GitHub Desktop** : interface graphique facilitant la gestion des dépôts GitHub [16].

4.1.2 Architecture logicielle adoptée

L'architecture de Mefria suit le modèle client-serveur :

— **Frontend** : développé avec SolidJS, stylisé avec BeerCSS et structuré selon les principes de Material Design 3. Capacitor est utilisé pour générer les applications mobiles natives.

— **Backend** : basé sur Node.js avec Express.js, Socket.IO pour la communication temps réel, et CouchDB pour le stockage.

— **Sécurité** : les communications client-serveur sont chiffrées à l'aide du protocole HTTPS.

4.2 Développement du frontend mobile

4.2.1 Interface utilisateur

L'interface utilisateur est conçue avec les objectifs suivants : simplicité, accessibilité, adaptabilité et performance.

— **BeerCSS** offre des composants visuellement modernes et réactifs (boutons, barres de navigation, cartes, listes).

— **Material Design 3** fournit un cadre visuel structurant et des composants UX accessibles, uniformisant l'expérience utilisateur.

— **Capacitor** permet de déployer l'application comme application native Android/iOS.

Les principales interfaces développées sont :

— Page d'accueil

— Formulaire d'inscription/connexion

— Liste des services disponibles

— Page de réservation

— Portefeuille numérique

— Profil utilisateur

— Messagerie instantanée (via Socket.IO)

4.2.2 Communication entre client et serveur

API REST (HTTPS) Toutes les requêtes HTTP (GET, POST, PUT, DELETE) vers le serveur passent par HTTPS, ce qui garantit la confidentialité et l'intégrité des données. Les appels sont sécurisés avec un token JWT transmis dans l'en-tête Authorization.

Socket.IO (WebSocket) Pour la messagerie instantanée, la réception des notifications ou le suivi des commandes en temps réel, Mefria utilise Socket.IO, qui repose sur le protocole WebSocket.

— Les messages utilisateur sont transmis via un canal `/chat`.

— Les notifications (réservations, paiements, confirmations) utilisent `/notifications`.

— Le suivi en temps réel (géolocalisation du prestataire) utilise `/tracking`.

4.3 Interfaces graphiques de l'application

Cette section présente les interfaces principales de l'application Mefria. Chaque interface est conçue pour offrir une navigation fluide, intuitive et conforme aux principes de Material Design 3. Les captures d'écran suivantes illustrent les écrans clés de l'application mobile.

4.3.1 Interface d'accueil

Cette interface est la première page affichée à l'utilisateur lors du lancement de l'application Mefria. Elle présente le slogan de l'application « Avec Mefria, vous oublierez l'attente ».

Deux actions sont proposées à l'utilisateur :

— **Commencer** : permet d'initier le parcours utilisateur.

— **Se connecter** : redirige vers l'écran d'authentification pour les utilisateurs possédant déjà un compte.



FIGURE 4.1 : Interface d'accueil de l'application Mefria

4.3.2 Interface d'inscription

Cette interface permet à l'utilisateur de créer un nouveau compte sur l'application Mefria. Elle comporte plusieurs champs à remplir pour l'enregistrement, incluant :

— Prénom

— Nom

— Numéro de téléphone

— Mot de passe

— Confirmation du mot de passe

— Genre (à sélectionner dans un menu déroulant)

Un bouton situé en bas permet de finaliser l'inscription. Un lien est également proposé pour revenir à la page précédente.

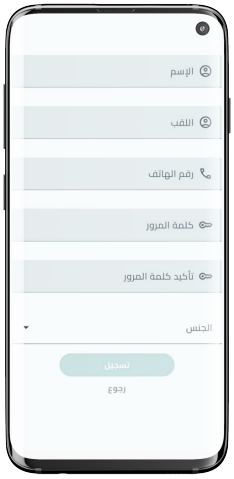


FIGURE 4.2 : Interface d'inscription de l'application Mefria

4.3.3 Interface de connexion

Cette interface permet à l'utilisateur de se connecter à son compte existant sur l'application Mefria. Elle contient deux champs principaux :

— Numéro de téléphone

— Mot de passe

Un bouton permet de valider l'authentification. Un lien est également présent pour revenir à la page précédente.



FIGURE 4.3 : Interface de connexion de l'application Mefria

4.3.4 Interface principale (page d'accueil)

Une fois connecté, l'utilisateur accède à cette interface principale qui présente les différentes catégories de services disponibles sur l'application Mefria. Chaque catégorie est représentée par une carte illustrée, facilitant la navigation visuelle.

Les services actuellement proposés incluent :

— Coiffure

— Barbier

— Restaurants

— Transport

La barre de navigation inférieure permet d'accéder rapidement aux sections suivantes : accueil, calendrier, et compte utilisateur.

À venir : de nouveaux services seront progressivement intégrés pour enrichir l'expérience utilisateur et répondre à un plus large éventail de besoins.

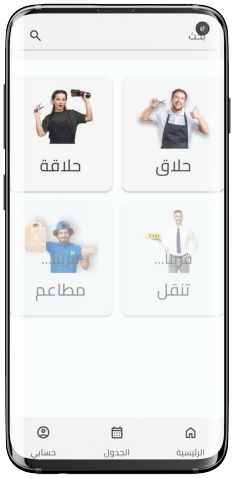


FIGURE 4.4 : Interface principale avec les catégories de services

4.3.5 Menu utilisateur et sélection de la langue

Cette interface représente le menu utilisateur accessible depuis l'application Mefria. En haut de l'écran, l'utilisateur peut voir son nom et accéder à différentes options relatives à son compte.

Les fonctionnalités accessibles depuis ce menu incluent :

— **Compte** — Informations personnelles et paramètres du profil.

— **Historique des activités** — Affiche les actions récentes effectuées par l'utilisateur.

— **Langue** — Permet de changer la langue de l'interface utilisateur.

— **Déconnexion** — Permet de se déconnecter de l'application en toute sécurité.

Lorsqu'on sélectionne l'option « langue », une liste déroulante s'affiche, proposant les langues disponibles : **arabe**, **français** et **anglais**. Cela permet à l'utilisateur d'adapter l'interface à sa langue préférée.

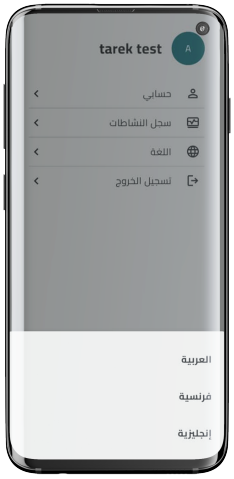


FIGURE 4.5 : Menu utilisateur avec options de gestion du compte et de langue

4.3.6 Interface des informations personnelles

Cette page permet à l'utilisateur de consulter ses informations personnelles enregistrées dans l'application Mefria. Elle offre une vue claire et simple des données essentielles liées à son profil.

Les éléments affichés incluent :

— **Nom de l'utilisateur** — Affiché en haut au centre, ici « tarek test ».

— **Numéro de téléphone** — Champ non modifiable affichant le numéro associé au compte.

— **Wilaya** — Région de résidence de l'utilisateur, ici « Tébessa ».

— **Bouton de déconnexion** — Permet de quitter l'application.

Une illustration personnalisée de l'utilisateur est également affichée, renforçant l'aspect humain et convivial de l'interface.



FIGURE 4.6 : Interface des informations personnelles de l'utilisateur

4.3.7 Historique des activités

L'interface de l'historique des activités permet à l'utilisateur de consulter la liste de ses réservations passées ou en cours. Cette fonctionnalité est essentielle pour assurer le suivi des services utilisés via l'application Mefria.

Deux onglets sont disponibles pour filtrer les données :

— **Terminé** — Affiche les services déjà effectués.

— **Annulé** — Affiche les réservations annulées.

Chaque carte d'activité comprend les informations suivantes :

— **Montant payé** (en dinars algériens).

— **Date de la réservation**.

— **Heure de début et de fin du service**.

— **Indication de complétion** (icône de coche).

Cette interface est conçue pour offrir une lecture rapide et organisée des transactions de l'utilisateur.



FIGURE 4.7 : Interface de l'historique des activités

4.3.8 Interface des boutiques de coiffure disponibles

L'interface présente les boutiques de coiffure situées dans la zone du client, permettant une sélection rapide et intuitive. Voici les établissements affichés :

— **The One** But House ! BounceBenne, Telesca, Algérie
text

— **Glam pain !**
Route Sans Hom, Telesca, Algérie



FIGURE 4.8 : Interface des boutiques de coiffure à proximité

4.3.9 Coiffeurs disponibles dans la boutique sélectionnée

L'interface affiche les coiffeurs actuellement disponibles dans la boutique choisie par le client, avec leurs horaires de travail.

— **Kalla Abdou** Disponibilité : 10 :00 - 23 :00

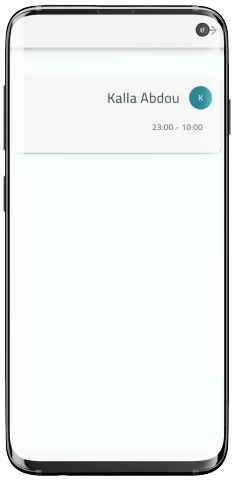


FIGURE 4.9 : Interface des coiffeurs disponibles dans la boutique sélectionnée

Cette interface permet au client de :

— Voir les coiffeurs en service

— Connaître leurs créneaux horaires

— Choisir le professionnel qui convient à ses besoins

4.3.10 Services disponibles chez le coiffeur sélectionné

L'interface présente la liste des services offerts par le coiffeur Kalla Abdou, avec les tarifs correspondants pour chaque prestation.

Liste des services et prix

- **Kreatine** - 1000 DA
- **Coupe** - 300 DA
- **Protein** - 1000 DA
- **Nettoyage de Peau** - 1000 DA
- **Barbe** - 150 DA
- **Broaching** - 150 DA

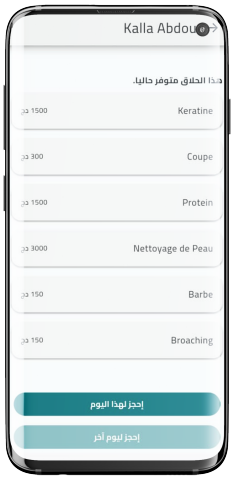


FIGURE 4.10 : Interface des services disponibles chez le coiffeur sélectionné

Fonctionnalités de l'interface

- Affichage clair des différents services de coiffure et soins
- Indication des prix en dinars algériens
- Organisation visuelle des prestations
- Possibilité de sélection pour prise de rendez-vous

4.4 Conclusion

Ce chapitre a permis de détailler l'ensemble du processus de mise en œuvre de l'application Mefria, en couvrant aussi bien l'environnement technique que les choix technologiques adoptés. L'architecture client-serveur, combinée à des technologies modernes comme SolidJS, Node.js, CouchDB et Socket.IO, a permis d'assurer une application réactive, sécurisée et évolutive.

Du côté frontend, l'utilisation de Capacitor et de BeerCSS, alliée aux principes de Material Design 3, a permis de concevoir une interface utilisateur intuitive, fluide et accessible sur différentes plateformes mobiles. Côté backend, la robustesse de Node.js, la légèreté d'Express.js et la puissance de la synchronisation offerte par CouchDB ont permis de construire une infrastructure solide, adaptée aux besoins de Mefria.

L'intégration des fonctionnalités en temps réel via WebSocket, la prise en compte de la sécurité avec HTTPS et JWT, ainsi que la gestion structurée des interfaces utilisateur démontrent une approche professionnelle et centrée sur l'expérience utilisateur.

La section des interfaces graphiques a illustré concrètement les résultats obtenus, confirmant l'adéquation entre les objectifs de conception et leur implémentation effective.

Ce travail d'implémentation constitue une base fonctionnelle stable sur laquelle il sera possible de faire évoluer l'application en ajoutant de nouvelles fonctionnalités ou en adaptant l'interface à d'autres cas d'usage.

Chapitre 5

Conclusions et travaux futurs

Ce travail s'inscrit dans le cadre de la conception et du développement d'une super application mobile, nommée Mefria, qui vise à centraliser plusieurs services essentiels dans une seule plateforme conviviale et accessible.

Dans le premier chapitre, nous avons présenté le concept des super applications, en expliquant leur origine, leur fonctionnement, ainsi que leur impact croissant dans les écosystèmes numériques modernes, notamment en Algerie. Nous avons mis en lumière leur capacité à regrouper une multitude de services (rendez-vous, paiement, logistique, e-commerce, etc.) au sein d'un seul environnement intégré.

Le deuxième chapitre a été consacré à la spécification des besoins et à la conception de notre application. Nous avons identifié les fonctionnalités clés à implémenter, et modélisé le système à travers différents diagrammes UML : diagrammes de cas d'utilisation, de séquence et de classes, pour chaque service proposé (prise de rendez-vous, demande d'artisans, logistique, e-commerce), en vue d'assurer une structure cohérente et modulaire.

Le troisième chapitre en explotant les technologie et les interface . Nous y avons décrit les technologies utilisées pour le développement de l'application, les choix techniques effectués pour assurer robustesse et évolutivité, ainsi que la conception des interfaces graphiques côté client. Ces dernières ont été pensées pour offrir une expérience utilisateur fluide, intuitive et adaptée aux besoins identifiés.

Nous travaillons sur ce projet depuis près d'une année. Actuellement, l'application Mefria offre un service de prise de rendez-vous chez le coiffeur et la coiffeuse. Parallèlement, nous sommes en train de développer un service de livraison, qui sera bientôt intégré à l'application. À l'avenir, nous prévoyons d'ajouter progressivement les autres services planifiés, tels que la commande d'artisans, la logistique, et le commerce électronique, afin de faire de Mefria une véritable super application multiservices complète et accessible à tous.

Enfin, nous sommes heureux de partager notre expérience avec vous, et nous vous invitons à utiliser notre application et à nous attendre, si Dieu le veut, sur la scène mondiale.

Bibliographie

[1] "wechat site," https://www.wechat.com/, 2025.

[2] "grab site," https://www.grab.com/sg/, 2025.

[3] "gojek site," https://www.gojek.com/en-id, 2025.

[4] "Node.js documentation," https://nodejs.org, 2025.

[5] "Express.js documentation," https://expressjs.com, 2025.

[6] "Apache couchdb," https://couchdb.apache.org, 2025.

[7] "Socket.io," https://socket.io, 2025.

[8] "Https - hypertext transfer protocol secure," https://developer.mozilla.org/fr/docs/Web/HTTP/Overview, 2025.

[9] "Solidjs documentation," https://www.solidjs.com, 2025.

[10] "Capacitor by ionic," https://capacitorjs.com, 2025.

[11] "Beercss," https://www.beercss.com, 2025.

[12] "Material design 3," https://m3.material.io, 2025.

[13] "Staruml," https://staruml.io, 2025.

[14] "Thunder client," https://www.thunderclient.com, 2025.

[15] "Git documentation," https://git-scm.com/doc, 2025.

[16] "Github desktop," https://desktop.github.com, 2025.