****

**REPUBLIQUE DU CAMEROUN**

Paix-Travail-Patrie



**UNIVERSITE DE DOUALA**

**ECOLE NATIONALE SUPERIEURE POLYTECHNIQUE DE DOUALA**

B.P.2701 Douala

Site web : [www.enspd.udo.cm](http://www.enspd.udo.cm)

**REPUBLIC OF CAMEROON**

Peace-Work-Fatherland



**THE UNIVERSITY OF DOUALA**

**NATIONAL HIGHER POLYTECHNIC SCHOOL OF DOUALA**

P.O. Box : 2701 Douala

Phone : (237) 697549240

Email: [www.enspd.udo.cm](http://www.enspd.udo.cm)

**THEME :** **CONCEPTION D’UNE PLATEFORME NUMERIQUE (WEB ET MOBILE) DESTINEE A CONNECTER LES PRESTATAIRES DE SERVICES LOCAUX AVEC DES CLIENTS A LA RECHERCHE DE COMPETENCES DANS LEUR QUARTIER OU LEUR REGION**

**Participant :**

|  |  |
| --- | --- |
| **Nom et prénoms** | **Matricule** |
| **HENEG BAYIHE JOSEPH ADOLPHE** | **22G00160** |
| **DJIKI DJIKI JUNIOR ARTHUR** | **24G01084** |
| **KAMGA FOYANG DARY** | **24G01095** |
| **METAMMO TEUMA DANIELA CLEDYA** | **22G00490** |
| **MPAY JULYANNA CACHARELLE** | **24G01116** |

**Sous l’encadrement professionnel de :**

**Dr IHONOCK**

Année académique 2025-2026

# SOMMAIRE

[SOMMAIRE ii](#_Toc217047925)

[LISTE DES TABLEAUX v](#_Toc217047926)

[LISTE DES FIGURES vi](#_Toc217047927)

[INTRODUCTION GENERALE 1](#_Toc217047928)

[ANALYSE DU PROJET 2](#_Toc217047929)

[Introduction 2](#_Toc217047930)

[I- Étude de l'existant 2](#_Toc217047931)

[1.1. Plateformes généralistes existantes 2](#_Toc217047932)

[1.2. Moyens informels utilisés localement 3](#_Toc217047933)

[II- Critique de l'existant 3](#_Toc217047934)

[III- Problématique 3](#_Toc217047935)

[IV- Proposition de solution 4](#_Toc217047936)

[V- Démarche d’analyse 4](#_Toc217047937)

[5.1. Etude comparative UML/MERISE 5](#_Toc217047938)

[5.2. Choix et présentation de la démarche d’analyse 5](#_Toc217047939)

[5.3. Etude comparative des processus unifiés 8](#_Toc217047940)

[5.4. Choix et présentation du processus unifiés 9](#_Toc217047941)

[Conclusion 11](#_Toc217047942)

[CAHIER DE CHARGE 12](#_Toc217047943)

[Introduction 12](#_Toc217047944)

[I- Contexte et justification 12](#_Toc217047945)

[1.1. Contexte 12](#_Toc217047946)

[1.2. Justification 12](#_Toc217047947)

[II- Cibles du projet 12](#_Toc217047948)

[III- Objectifs 12](#_Toc217047949)

[3.1. Objectif général 12](#_Toc217047950)

[3.2. Objectifs spécifiques 13](#_Toc217047951)

[IV- Expression des besoins 13](#_Toc217047952)

[4.1. Besoins fonctionnels 13](#_Toc217047953)

[4.2. Besoins non fonctionnels 13](#_Toc217047954)

[V- CHARTE GRAPHIQUE 14](#_Toc217047955)

[5.1. Logo 14](#_Toc217047956)

[5.2. Typographie 15](#_Toc217047957)

[5.3. Couleurs 15](#_Toc217047958)

[VI- Planification 15](#_Toc217047959)

[6.1. Déroulement 15](#_Toc217047960)

[6.2. Intervenants 15](#_Toc217047961)

[6.3. Planification des taches 16](#_Toc217047962)

[6.4. Diagramme de gant 17](#_Toc217047963)

[VII- Estimation du cout 18](#_Toc217047964)

[VIII- Contraintes 18](#_Toc217047965)

[IX- Les livrables attendus 19](#_Toc217047966)

[Conclusion 19](#_Toc217047967)

[CONCEPTION 20](#_Toc217047968)

[Introduction 20](#_Toc217047969)

[I- Objectifs de la conception 20](#_Toc217047970)

[II- Présentation des diagrammes 20](#_Toc217047971)

[2.1. Diagramme de cas d’utilisation 20](#_Toc217047972)

[2.2. Diagramme de classes 24](#_Toc217047973)

[2.3. Diagramme de séquence 27](#_Toc217047974)

[2.4. Diagramme d’activités 30](#_Toc217047975)

[2.5. Diagramme d’objet 33](#_Toc217047976)

[2.6. Diagramme de paquetage 34](#_Toc217047977)

[2.7. Diagramme d’état transition 35](#_Toc217047978)

[2.8. Diagramme de communication 36](#_Toc217047979)

[Conclusion 37](#_Toc217047980)

[REALISATION 38](#_Toc217047981)

[I- Diagramme de déploiement 38](#_Toc217047982)

[1.1. Définition 38](#_Toc217047983)

[1.2. Formalisme 38](#_Toc217047984)

[1.3. Représentation graphique du système du diagramme de déploiement 39](#_Toc217047985)

[II- Diagramme de composant 39](#_Toc217047986)

[2.1. Définition 39](#_Toc217047987)

[2.2. Formalismes 39](#_Toc217047988)

[2.3. Présentation du diagramme de composant 40](#_Toc217047989)

[III- Présentation de l’architecture du système 40](#_Toc217047990)

[3.1. Architecture physique 40](#_Toc217047991)

[3.2. Architecture logique 41](#_Toc217047992)

[IV- Méthodologie de développement 42](#_Toc217047993)

[V- Présentation de l’environnement 43](#_Toc217047994)

[5.1. Environnement logiciel 43](#_Toc217047995)

[5.2. Technologies utilisées 44](#_Toc217047996)

[Conclusion 44](#_Toc217047997)

[CONCLUSION GENERALE 55](#_Toc217047998)

# LISTE DES TABLEAUX

[Tableau 1 : Etude comparative UML/MERISE 5](#_Toc216950341)

[Tableau 2 : Description des processus 8](#_Toc216950342)

[Tableau 3 : Couleurs du logo 15](#_Toc216950343)

[Tableau 4 : Liste des intervenants 15](#_Toc216950344)

[Tableau 5 : Planification des taches 16](#_Toc216950345)

[Tableau 6 : Formalisme diagramme de cas d’utilisation 20](#_Toc216950346)

[Tableau 7 :Formalisme du diagramme de classe 25](#_Toc216950347)

[Tableau 8 :Règles metiers 26](#_Toc216950348)

[Tableau 9 :Formalisme du diagramme de sequence 28](#_Toc216950349)

[Tableau 10 :Formalisme du diagramme d’activité 31](#_Toc216950350)

[Tableau 11 :Formalisme diagramme d’objet 33](#_Toc216950351)

[Tableau 12 :Formalisme du diagramme d’état-transition 36](#_Toc216950352)

[Tableau 13 :Formalisme du diagramme de deploiement 38](#_Toc216950353)

[Tableau 14 :Formalisme du diagramme de composant 40](#_Toc216950354)

[Tableau 15 :Technologies utisées 44](#_Toc216950355)

[Tableau 16 :Fonctionnalité de l’application 46](#_Toc216950356)

# LISTE DES FIGURES

[Figure 1 : Hiérachie des diagrammes UML 7](#_Toc217033660)

[**Figure 2** : Présentation des processus unifiés 10](#_Toc217033661)

[Figure 3 : Logo de l’application 14](#_Toc217033662)

[Figure 4 : Diagramme de Gant (backend) 17](#_Toc217033663)

[Figure 5 : Diagramme de Gant (frontend) 18](#_Toc217033664)

[Figure 6 : Diagramme de cas d’utilisation pour client 22](#_Toc217033665)

[Figure 7 : Diagramme de cas d’utilisation pour prestataire 23](#_Toc217033666)

[Figure 8 : Diagramme de cas d’utilisation pour Administrateur 24](#_Toc217033667)

[Figure 9 : Diagramme de classe 26](#_Toc217033668)

[Figure 10 : Diagramme de séquence pour authentification 29](#_Toc217033669)

[Figure 11 : Diagramme de séquence pour recherche d’un prestataire 30](#_Toc217033670)

[Figure 12 31](file:///C:\Users\hp\Desktop\mise_à_jour_rapportBonBon.docx#_Toc217033671)

[Figure 13 : Diagramme d’activité pour authentification 32](#_Toc217033672)

[Figure 14 : Diagramme d’activité pour recherche d’un prestataire 32](#_Toc217033673)

[Figure 15 : Diagramme d’objet 34](#_Toc217033674)

[Figure 16 : Diagramme de paquetage 35](#_Toc217033675)

[Figure 17 : Diagramme d’état transition 36](#_Toc217033676)

[Figure 18 : Diagramme de communication 37](#_Toc217033677)

[Figure 19 : Diagramme de deploiement 39](#_Toc217033678)

[Figure 20 : Diagramme de commposant 40](#_Toc217033679)

[Figure 21 : Architecture physique 41](#_Toc217033680)

[Figure 22 : Architecture logique 42](#_Toc217033681)

[Figure 23 : Test interface utilisateur 48](#_Toc217033682)

[Figure 24 : Test interface utilisateur 49](#_Toc217033683)

[Figure 25 : Résultat 49](#_Toc217033684)

[Figure 26 : Page connexion utilisateur 51](#_Toc217033685)

[Figure 27 : Page d’accueil Dashboard administrateur 52](#_Toc217033686)

[Figure 28 : Page de recherche d’un prestataire 52](#_Toc217033687)

[Figure 29 : Page de gestion des prestataires 53](#_Toc217033688)

[Figure 30 : Page de profil d’un prestataire 53](#_Toc217033689)

[Figure 31 : Page de notation d’un prestataire 54](#_Toc217033690)

[Figure 32 : Aperçu de la base de données 54](#_Toc217033691)

# INTRODUCTION GENERALE

Dans un contexte de digitalisation croissante, la mise en relation entre prestataires de services et clients devient un enjeu stratégique pour favoriser l’accès rapide et efficace aux compétences disponibles localement. Le projet que nous proposons vise à concevoir une plateforme numérique (web et mobile) permettant de connecter les prestataires de services locaux avec des clients à la recherche de compétences spécifiques dans leur zone géographique. Cette solution contribuera à dynamiser l’économie locale, à améliorer la visibilité des prestataires, et à simplifier la recherche de services de proximité.

# ANALYSE DU PROJET

## Introduction

Pour mener à bien un projet il faut l’analyser correctement, pour se faire nous devons étudier le système existant dans plusieurs aspects et collecter les informations qui seront utiliser tous au long de notre projet. Pour cela nous nous appuierons sur une démarche d’analyse. Nous présenterons ensuite nos critiques sur le system existant, la solution que nous proposons et enfin nous présenterons notre éventuelle solution et le résultat de notre modélisation.

## Étude de l'existant

Plusieurs solutions numériques ont été développées pour faciliter la mise en relation entre prestataires de services et clients. Ces plateformes se répartissent en deux grandes catégories : les plateformes généralistes à large portée géographique, et les solutions locales informelles utilisées à travers les réseaux sociaux ou les contacts personnels.

### Plateformes généralistes existantes

#### Jumia Services, Glovo, Bolt Services

Ces plateformes permettent aux clients de commander des services comme la livraison, le transport, ou l’assistance technique. Bien qu’elles soient très populaires, elles présentent des limites dans leur couverture :

* Elles ne référencent qu’un nombre limité de prestataires agréés.
* Elles sont généralement orientées vers des services spécifiques (livraison, transport) et non vers tous les types de prestations locales (plombier, maçon, couturière, réparateur d’ordinateurs…).
* Elles ciblent surtout les grandes villes, délaissant les zones rurales ou semi-urbaines.

#### Sites de freelances : Upwork, Fiverr

Ces plateformes mettent en relation freelances et clients pour des missions à distance (graphisme, rédaction, développement web, etc.). Toutefois :

- Elles sont inadaptées aux services de proximité qui nécessitent une présence physique.

- Les frais de service sont souvent élevés.

- Elles ne sont pas accessibles à tous, car la barrière linguistique et technologique (profil, compte bancaire, facturation…) empêche de nombreux prestataires locaux de s’y inscrire

### Moyens informels utilisés localement

#### Groupes WhatsApp, Facebook, Telegram

Dans plusieurs villes ou quartiers, les habitants utilisent les réseaux sociaux pour demander ou recommander des prestataires (ex. : « Connaissez-vous un bon électricien ? »). Avantages :

* Rapide et accessible à tous.
* Permet des recommandations basées sur la confiance (avis entre membres).
* Mais ces moyens sont :
* Désorganisés, car les messages se perdent dans les discussions.
* Non sécurisés (pas de vérification d’identité, ni d’évaluations fiables).
* Pas structurés pour la gestion de profils, de zones, ou de types de services.

#### Bouche-à-oreille / Carnets d’adresses

Encore très utilisé, ce système est basé sur la recommandation directe. Il repose sur :

* La confiance personnelle, mais reste limité à un petit réseau.
* L’absence de moyens pour consulter des avis publics ou évaluations objectives.

## Critique de l'existant

Avant de proposer une solution, il est essentiel d’analyser les limites du système existant afin d’identifier les points à améliorer. Les limites observées sont :

* Manque de ciblage local : les plateformes ne permettent pas une recherche fine par quartier ou région.
* Accessibilité limitée : certaines applications nécessitent des smartphones puissants ou une connexion internet stable.
* Peu de valorisation des petits prestataires : artisans ou techniciens peu présents ou mal référencés.
* Absence de confiance : manque de système d’évaluation fiable, ce qui rend difficile le choix d’un prestataire.

## Problématique

Les différents critiques apportés nous amènent à nous questionner sur comment concevoir une plateforme numérique simple, accessible et efficace, permettant de connecter facilement des prestataires de services locaux avec des clients, tout en garantissant la fiabilité, la rapidité et la sécurité des échanges ?

## Proposition de solution

Dans l’optique d’apporter une solution aux différentes limites des systèmes existants, nous avons pour but de mettre sur pied une plateforme web et mobile qui permet de :

* L’enregistrement et la création de profils de prestataires locaux (artisan, réparateur, professeur, etc.)
* La recherche géolocalisée par quartier ou ville
* Un système de notation et d’avis clients
* La messagerie directe entre client et prestataire
* Une interface simple et légère, même utilisable avec peu de connexion internet.

# Démarche d’analyse

L’analyse est une étape fondamentale dans la conception d’un logiciel. C’est la base de tous travaux de réalisation de système d’information. Un système d’information est un système organisé de ressources, de personnes, et de structures qui évoluent dans une organisation et dont le comportement coordonné vise à atteindre un but commun. Plusieurs méthodes et langages ont été développés pour faciliter et normaliser l’analyse et la conception des systèmes d’information parmi lesquels nous avons principalement UML et MERISE.

**MERISE** (Méthode d’Étude et de Réalisation Informatique pour les Systèmes d’Entreprise) est une méthode d’analyse, de conception structurelle et de réalisation des systèmes d’informations très utiles notamment dans les entreprises françaises. Elle est basée sur la séparation des données et des traitements à effectuer en plusieurs modèles conceptuels et physiques. Son but principal est d’arriver à concevoir un système d’information (SI). C’est une méthode systémique d’analyse et de conception de SI qui propose de considérer le système réel selon deux points de vue : une vue statique (données) et une vue dynamique (traitement).

**UML** (Unified Modeling Language) quant à lui, est un langage de modélisation des systèmes standards, qui utilise des diagrammes pour présenter chaque aspect d’un système en s’appuyant sur la notion d’orienté objet qui est un véritable atout pour ce langage. UML propose donc une approche différente de celle de MERISE en ce sens qu’il associe les données aux traitements. En effet, avec UML centraliser les données d’un type et le traitement associé permet de limiter les points de maintenance dans le code et faciliter l’accès à l’information en cas d’évolution de logiciel. De plus, UML décrit la dynamique du système d’information comme un ensemble d’opérations attachée aux objets du système.

## Etude comparative UML/MERISE

**Tableau 1** : Etude comparative UML/MERISE

|  |  |
| --- | --- |
| **MERISE** | **UML** |
| Méthode d’étude et de Réalisation Informatique pour les Systèmes d’Entreprises. | Unified Modeling Language (Langage de modélisation Unifié). |
| MERISE est une méthode systémique d’analyse et de conception de systèmes d’information. C’est-à-dire qu’elle utilise une approche systémique. | UML est un langage de modélisation objet à qui il faut associer un processus (2TUP, RUP, XUP) pour en faire une méthode. |
| MERISE propose de considérer le système réel selon deux points de vue :  - Une vue statique (données)  - Une vue dynamique (traitements)  Avec la méthode MERISE, nous avons une étude séparée des données et des traitements. | UML propose une approche différente en ce sens qu’il associe les données et les traitements.  Avec UML, centraliser les données d’un type et les traitements associés permet de limiter les points de maintenance dans le code et faciliter l’accès à l’information en cas d’évolution. |
| Approche systémique | Approche objet. |
| Démarche « Bottom Up » : De la base de données vers le code. | Démarche « Top Down » : Du modèle vers la base de données. |

## Choix et présentation de la démarche d’analyse

Commençons d’abord par définir ce que c’est qu’un modèle. Un modèle est donc une représentation simplifiée d’une réalité, il permet de capturer des aspects pertinents pour un objectif défini à priori.

**UML** (Unified Modeling Language) est un langage de modélisation orienté objet graphique basé sur des pictogrammes. Il est apparu dans le monde du génie logiciel pour résoudre les problèmes de contraintes qui survenaient à différentes phases de conception d’un logiciel. Il en résulte que de nombreuses méthodes de développement ou d’analyse de logiciel ont vu le jour, chacune plus ou moins adaptée ou spécialisée à une démarche particulière. Celles-ci ayant été développées indépendamment les unes des autres, elles sont souvent partiellement redondantes ou incompatibles lorsqu’elles font appel à des notations ou des terminologies différentes. UML est donc l’accomplissement de précédents langages de modélisation que nous allons lister ici :

* **OMT (Object Modeling Technic)** de James Rumbaugh, c’est une méthode d’analyse et de conception orienté objet qui insistait sur l’analyse des systèmes logiciels.
* **OOD (Object Oriented Design)** de Grady Booch, c’était une méthode de conception orienté objet qui mettait un accent sur le design et la construction des systèmes logiciels.
* **OOSE (Object Oriented Software Engineering)** d’Ivar Jacobson, elle se concentrait au business engineering et à l’analyse des exigences.

UML est donc un langage qui permet de définir des modèles sans toutefois définir leurs processus d’élaboration.

UML à partir de sa version 1.3 propose neuf (09) diagrammes tandis qu’il en existe quatorze (14) depuis UML 2.3. Les 14 diagrammes UML sont dépendants hiérarchiquement et se complètent de façon à permettre la modélisation d’un projet tout au long de son cycle de vie.

Ils sont regroupés en trois grandes vues :

**\* Diagrammes structurels ou statiques**

- Diagramme de classes : il représente les classes intervenant dans le système ;

- Diagramme d’objets : Représente les instances de classes (objets) utilisées dans le système ;

- Diagramme de composants : Montre les composants du système d’un point de vue physique ;

- Diagramme de déploiement : Il sert à représenter les éléments matériels (ordinateurs, périphériques, réseaux, système de stockage…) et la manière dont les composants du système sont répartis sur ces éléments matériels et interagissent entre eux.

- Diagramme de paquetages : ce diagramme sert à représenter les dépendances entre les différents packages utilisés dans le système ;

- Diagramme de structure composite : Depuis UML 2.x permet de décrire les relations entre composants d’une classe ;

- Diagramme de profils : Depuis UML 2.2 permet de spécialiser, de personnaliser pour un domaine particulier un méta modèle de référence d’UML.

**\* Diagrammes comportementaux**

- Diagramme des cas d’utilisation : Il permet d’identifier toutes les fonctionnalités que doit fournir le système ;

- Diagramme état-transition : Permet de décrire sous forme de machine à états finis le comportement du système ou de ses composants ;

- Diagramme d’activité : Il décrit sous forme de flux ou d’enchaînement d’activités le comportement du système ou de ses composants.

**\* Diagrammes d’interaction ou dynamiques**

- Diagramme de séquence : Il représente séquentiellement le déroulement des traitements et des interactions entre les éléments du système et/ou de ses acteurs ;

- Diagramme de communication : Disponible depuis UML 2.2, c’est une représentation simplifiée d’un diagramme de séquence qui se concentre sur les échanges de messages entre objets ;

- Diagramme global d’interaction : Depuis UML 2.x, ce diagramme permet de décrire les enchaînements possibles entre les scénarios préalablement identifiés sous forme de diagrammes de séquences ;

- Diagramme de temps : Depuis UML 2.3, permet de décrire les variations d’une donnée à court du temps.

Nous allons illustrer la hiérarchie des diagrammes UML comme suite :

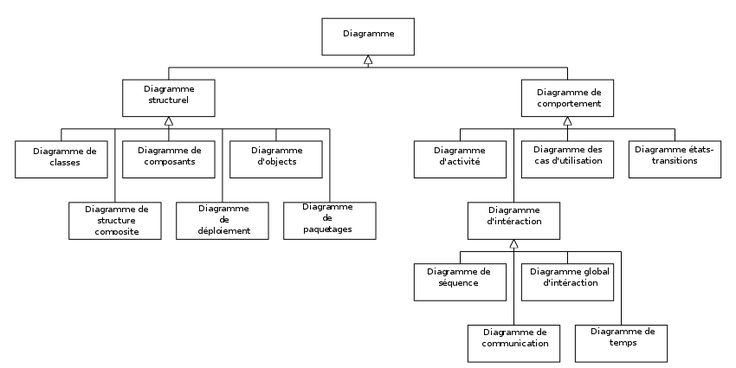


Figure 1 : Hiérachie des diagrammes UML

Dans le cadre de ce projet, nous travaillerons avec les spécifications de la version 1.4 sortie en 2003 qui compte 9 diagrammes répartis sur 2 vues.

**\* Les vues statiques :**

- Diagrammes de cas d’utilisation ;

- Diagrammes d’objets ;

- Diagrammes de classes ;

- Diagrammes de composants ;

- Diagrammes de déploiement.

**\* Les vues dynamiques :**

- Diagrammes de collaboration ;

- Diagrammes de séquence ;

- Diagrammes d’état-transitions ;

- Diagrammes d’activité.

Malgré tous ses atouts incontestés, UML ne propose néanmoins pas de façon claire une démarche à suivre dans l’élaboration d’un projet, de la compréhension des besoins des utilisateurs à la production du logiciel. Pourtant, le choix d’une méthode est d’une importance capitale dans la démarche de mise sur pied d’un logiciel. C’est pourquoi, il faut lui associer un processus de traitement.

## Etude comparative des processus unifiés

Plusieurs processus ont été définis pour accompagner le langage UML. Le tableau ci-dessous présente quelques processus unifiés utilisant UML.

**Tableau 2** : Description des processus

|  |  |
| --- | --- |
| **Processus** | **Description** |
| **AUP** | Agile Unified Process (AUP) est une variante simplifiée d’UP qui intègre des pratiques agiles comme le développement piloté par les tests (TDD). Elle reprend ainsi les phases séquentielles, et les jalons de fin de phase du processus unifié, mais renforce l’approche agile en préconisant une mise à jour du plan en fin de chaque phase, la mise en œuvre d’une version potentiellement publiable à la fin de chaque itération, et l’application des principes du manifeste agile. |
| **EssUP** | Essential Unified Process (EssUP) cherche à alléger le processus unifié qui bien qu’itératif et incrémental est souvent perçu comme trop lourd et trop formalisé. EssUP adopte pour cela le principe de la séparation des préoccupations. Plutôt qu’un processus monolithique, EssUP définit un ensemble de pratiques indépendantes qui peuvent être librement combinées pour former un processus adapté au contexte. |
| **2TUP** | 2TUP est un processus unifié qui a pour but d’apporter une réponse aux contraintes de changement fonctionnelles et techniques qui s’imposent aux systèmes d’information, il propose un cycle de développement qui dissocie les aspects techniques des aspects fonctionnels |
| **RUP** | Dérivé d’UP et commercialisé par IBM en 1998, RUP est l’une des plus célèbres implémentations de la méthode UP permettant de donner un cadre au développement logiciel.  RUP est un exemple de formalisation d’un processus fondé sur le cycle de vie en spirale. |

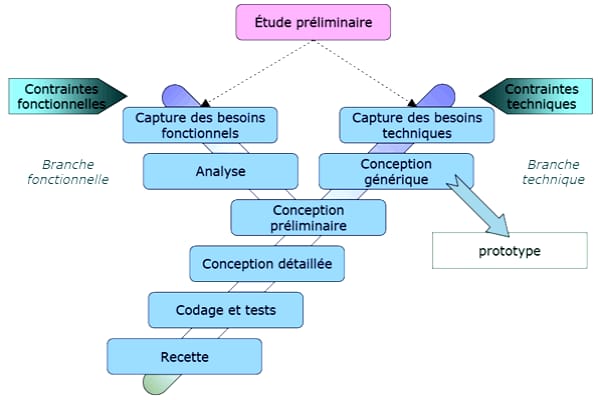
## Choix et présentation du processus unifiés

En ce qui concerne la méthode que nous allons utiliser, il s’agit de celle directement rattachée au langage de modélisation UML à savoir la méthode 2TUP qui signifie 2 Track Unified Process.

C’est un cycle de développement en Y qui dissocie les aspects techniques des aspects fonctionnels qui correspondent aux deux chemins utilisés pour le développement des systèmes informatiques.

Tout processus UP (Unified Process) unifié répond aux caractéristiques suivantes : il est itératif et incrémental. Ainsi donc les avantages que nous présente la méthode 2TUP sont :

* Elle permet de rendre les systèmes extensibles ;
* Elle permet l’ajout des modules ;
* Elle permet la flexibilité dans l’analyse d’un projet ;
* Elle est facile à la conception et la compréhension ;
* Elle permet d’avoir une vue préalable de comment se présentera l’application.



**Figure 2** : Présentation des processus unifiés

* **Décrivons la forme Y de la méthode 2TUP :**
* **La branche gauche** (fonctionnelle) comporte :
* **La capture des besoins fonctionnels** : qui produit un modèle des besoins focalisé sur le métier des utilisateurs. Elle qualifie au plus tôt le risque de produire un système inadapté aux utilisateurs. De son côté, la maîtrise d’œuvre consolide les spécifications et en vérifie la cohérence et l’exhaustivité ;
* **L’analyse** : qui consiste à étudier précisément la spécification fonctionnelle de manière à obtenir une idée de ce que va réaliser le système en termes de métier. Les résultats de l’analyse ne dépendent d’aucune technologie particulière.
* La branche droite (architecture technique) comporte :
* **La capture des besoins techniques** : qui recense toutes les contraintes et les choix dimensionnant la conception du système. Les outils et les matériels sélectionnés ainsi que la prise en compte de contraintes d’intégration avec l’existant conditionnent généralement des prérequis d’architecture technique.
* Les architectures matérielles et logicielles : constituant de la conception générique, qui définit ensuite les composants nécessaires à la construction de l’architecture technique. Cette conception est complètement indépendante des aspects fonctionnels. Elle a pour objectif d’uniformiser et de réutiliser les mêmes mécanismes pour tout un système. L’architecture technique construit le squelette du système informatique et écarte la plupart des risques de niveau technique. L’importance de sa réussite est telle qu’il est conseillé de réaliser un prototype pour assurer sa validité.
* La branche du milieu (conception) comporte :
* **La conception préliminaire** : qui représente une étape délicate, car elle intègre le modèle d’analyse dans l’architecture technique de manière à tracer la cartographie des composants du système à développer ;
* **La conception détaillée** : qui étudie ensuite comment réaliser chaque composant, l’étape de codage, qui produit ces composants et teste au fur et à mesure les unités de code réalisées ; l’étape de recette, qui consiste enfin à valider les fonctions du système développé.

**Codage**, **tests** et **recettage** : qui renvoie à la phase de programmation des fonctionnalités au fur et à mesure ; et aussi, à la validation des fonctions du système développé.

## Conclusion

Ce projet s’inscrit dans une logique de valorisation des compétences locales et de développement d’un écosystème numérique adapté aux réalités de nos communautés. En facilitant la connexion entre offre et demande de services à l’échelle locale, la plateforme envisagée contribuera à la création d’opportunités économiques et à la transformation digitale inclusive des prestataires souvent peu visibles dans les circuits classiques.

# CAHIER DE CHARGE

## Introduction

Le cahier des charges est un document préalablement établit entre le maitre d’œuvre et la maitrise d’ouvrages qui étale les besoins du client. D’une manière générale, il étudie et présente avec exactitude les exigences formulées par les utilisateurs en ce qui concerne un projet, son déroulement et les résultats attendus. Le cahier de charge regroupe donc le contexte et la justification du projet, l’expression des besoins de l’utilisateur, la planification, les contraintes et les livrables.

## Contexte et justification

### Contexte

Dans un contexte marqué par une forte informalité des services et un accès limité à des plateformes adaptées aux réalités locales, les clients rencontrent des difficultés pour trouver des prestataires fiables à proximité. En parallèle, de nombreux prestataires qualifiés peinent à se faire connaître et à valoriser leurs compétences.

### Justification

Ce projet vise à pallier l’absence d’un canal structuré, sécurisé et localisé permettant une mise en relation efficace. Il répond à des besoins concrets : simplifier l’accès aux services, formaliser l’activité des prestataires et développer une économie locale plus dynamique et connectée.

## Cibles du projet

Cette plateforme sera spécialement conçue pour les Prestataires de services, les Clients.

* **Prestataires de services** : artisans, réparateurs, aides à domicile, enseignants, etc.
* **Clients** : particuliers, entreprises ou organisations cherchant des services fiables à proximité.
* **Administrateurs** : équipe en charge de la gestion et de la supervision de la plateforme.

## Objectifs

### Objectif général

Développer une application web/mobile permettant de connecter des clients avec des prestataires locaux de manière fluide, géolocalisée et sécurisée.

### Objectifs spécifiques

* Permettre aux prestataires de créer un profil et proposer leurs services.
* Offrir aux clients un moteur de recherche intelligent.
* Intégrer un système de notation et de feedback.
* Gérer les réservations, les notifications et le suivi des prestations.

## Expression des besoins

### Besoins fonctionnels

Les besoins fonctionnels définissent les services que les utilisateurs s’attendent à voir, fournis par la plateforme. Ils sont en relations avec un ensemble de besoins utilisateur et décrivent les fonctionnalités que la plateforme doit offrir. De ce fait, dans l’optique de satisfaire la totalité des exigences des utilisateurs, les besoins fonctionnels sont classés comme suite :

* Inscription / Connexion (clients et prestataires)
* Création de profil avec photo, spécialité, zone d'intervention
* Recherche de prestataires par catégorie et localisation
* Réservation et messagerie interne
* Paiement en ligne (facultatif ou via mobile money)
* Notation et avis clients
* Tableau de bord admin pour gestion et modération

### Besoins non fonctionnels

Ce sont les besoins spécifiant les propriétés du système telle que performance, fiabilité, sécurité, accessibilité etc. Dans notre système, on distingue les besoins non fonctionnels suivant :

* Application responsive (PC, mobile)
* Sécurité des données utilisateurs
* Disponibilité 24h/24 - 7j/7
* Interface simple et intuitive
* Temps de réponse rapide

## CHARTE GRAPHIQUE

## Logo

Le logo de **Helpix** représente une localisation connectée à un réseau intelligent, symbolisant l’assistance rapide, locale et digitale.

Il est composé :

- D’une forme de repère de localisation stylisée,

- D’arcs fléchés pour représenter l’échange,

- D’un réseau de points connectés au centre.

**Usage :**

Utiliser sur fond blanc de préférence pour un meilleur contraste.



Figure 3 : Logo de l’application

### Typographie

**Nom de la marque :**

- "help" en bleu foncé : moderne, professionnel

- "pix" en orange et bleu clair : dynamisme, technologie

**Police suggérée** :

- Sans-serif moderne comme Poppins, Montserrat ou Lato

### Couleurs

Tableau 3 : Couleurs du logo

|  |  |  |
| --- | --- | --- |
| Couleur | Code HEX | Utilisation |
| Orange vif | #F76C1F | Accent, dynamisme |
| Bleu turquoise | #00B8C4 | Élément graphique, modernité |
| Bleu foncé | #142850 | Texte principal, professionnalisme |

## Planification

### Déroulement

Cette étape nous permet de représenter la répartition des taches sur les intervalles de temps bien définis. Pour cette planification, nous ferons recours au logiciel Gant Project. Nous obtiendrons donc le chronogramme de Gant qui représentera visuellement l’état d’avancement de nos différentes tâches.

### Intervenants

**Tableau 4** : Liste des intervenants

|  |  |
| --- | --- |
| **Noms et prénoms** | **Rôles** |
| M. HENEG Joseph | Chef de projet, Développeur frontend, Développeur Backend |
| Mme METAMMO Daniela | Analyste, Développeur frontend |
| M. DJIKI Arthur | Développeur backend, Développeur frontend |
| Mme MPAY Cacharelle | Développeur frontend |
| M. KAMGA Daryl | Développeur frontend |

### Planification des taches

La planification de projet est une activité qui consiste à déterminer et à ranger les taches du projet en les présentant sur des intervalles de temps bien délimitées. Pour ainsi atteindre nos objectifs suscités, nous avons élaboré un plan de management du projet en plusieurs taches et chronogramme d’activité dans un diagramme de Gant.

**Tableau 5** : Planification des taches

|  |  |  |  |
| --- | --- | --- | --- |
| **Programme frontend** | | **Programme backend** | |
| Taches | Durée (en Jours) | Taches | Durée (en Jours) |
| Setup React/Vue + routing + thèmes. | 1 | Setup projet, DB, entités User/ServiceProvider/Service/Reservation. | 1 |
| Login/Register + validations. | 2 | API Auth (register/login/reset) + JWT/Sanctum. | 2 |
| Profil client et prestataire. | 3 | API Profil (CRUD) + profil prestataire. | 3 |
| Gestion services côté prestataire. | 4 | CRUD Services + relation prestataire. | 4 |
| Liste prestataires + carte. | 5 | API recherche + géolocalisation. | 5 |
| Réservation + dashboard prestataire. | 6 | API Réservations (créer/ accepter/ annuler/ terminer). | 6 |
| Chat web (temps réel/poll). | 7 | API Messagerie (conversations/ messages). | 7 |
| Geston des admin (CRUD) et gestion des user(bloquer). | 8 | API Avis + calcul note moyenne. | 8 |
| Tests UX + corrections. | 9 | Sécurité API + tests unitaires. | 9 |
| Build + déploiement Web. | 10 | Déploiement API (Docker/VPS). | 10 |

### Diagramme de gant

Afin de mener à bien le projet et de le livrer dans le temps impartis qui est de 01 mois, ce dernier a été jalonné en plusieurs phases qui représentent des périodes. Au bout de chaque jalon défini par une période bien précise et d’un objectif distinct à atteindre, nous aboutissons à des résultats qui marquent l’avancée progressive du projet. Il est représenté dans le tableau qui sui

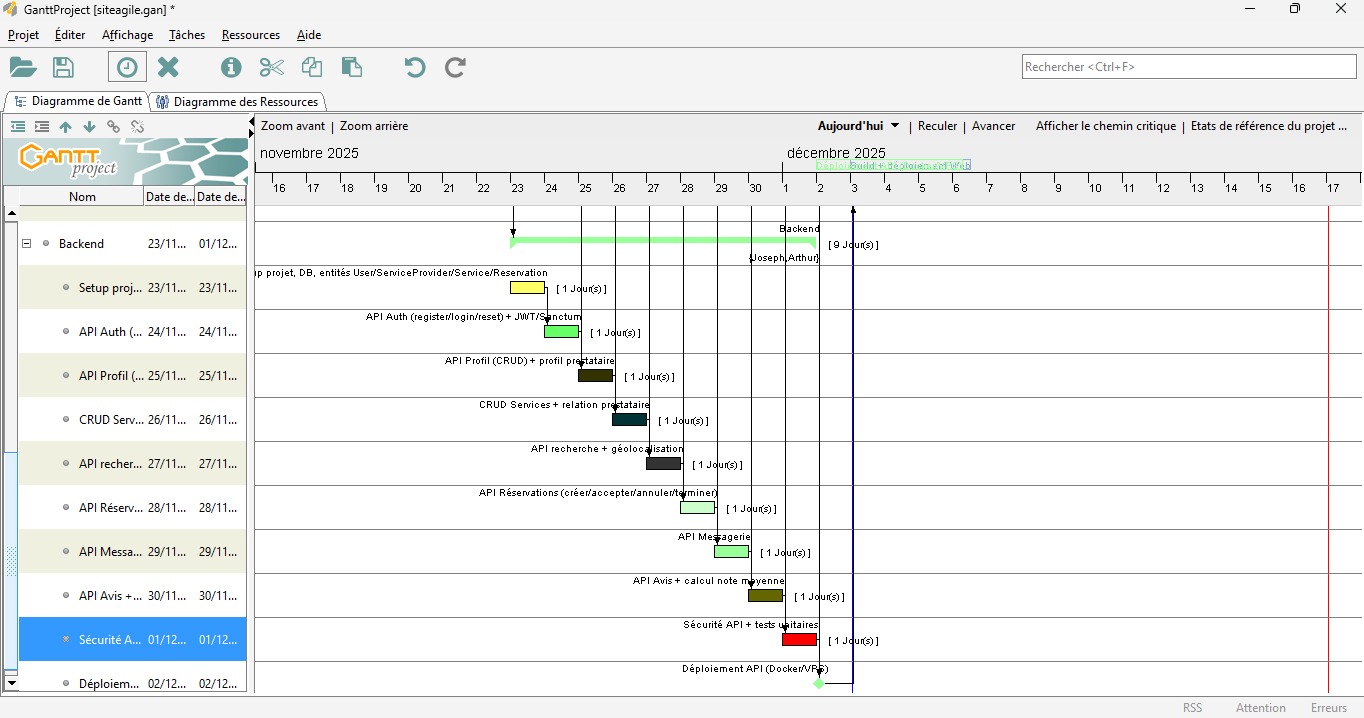


Figure 4 : Diagramme de Gant (backend)

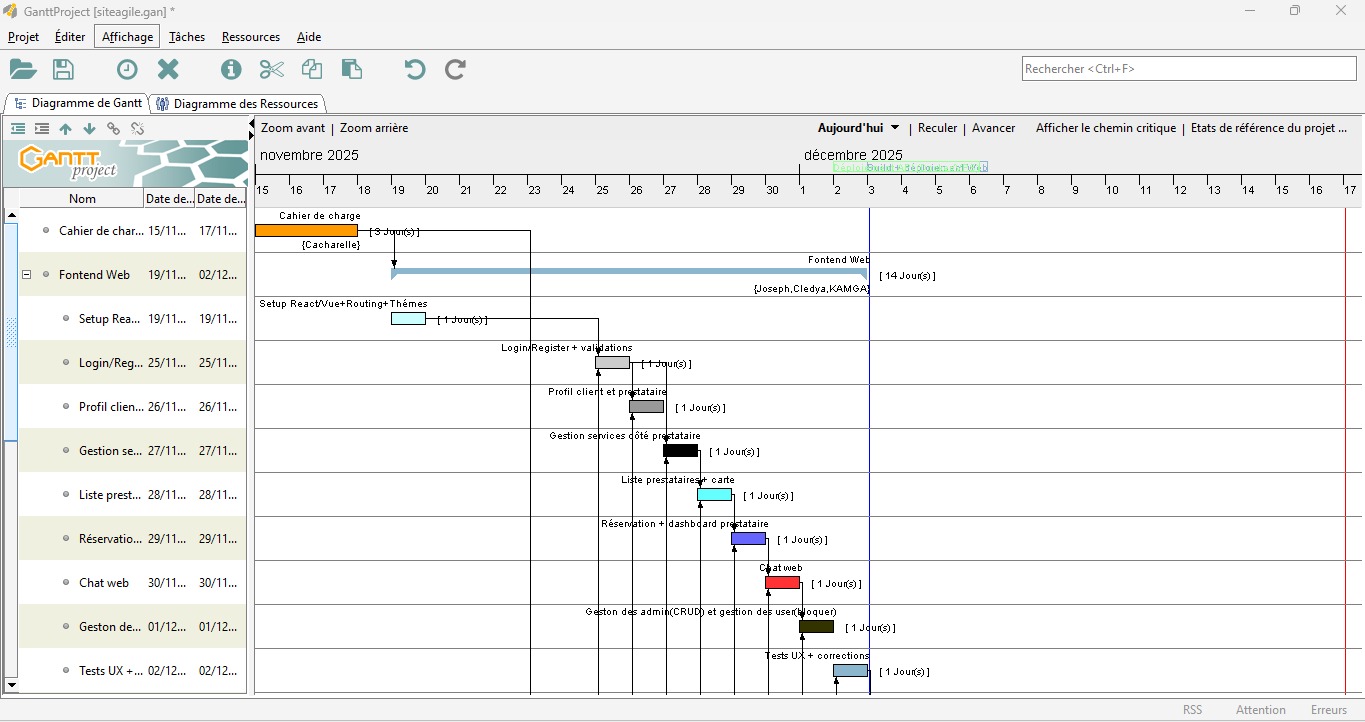


Figure 5 : Diagramme de Gant (frontend)

## Estimation du cout

* + Couts d’infrastructure et licences
* Hébergement cloud (AWS/GCP) : 420 000 XAF
* Licences logicielles et outils : 250 000 XAF
* Certificats SSL et sécurité : 80 000 XAF
* Couts marketing et commerciaux
* Etude de marché approfondie : 300 000 XAF
* Documentation : 150 000 XAF
* Cout total : **1 200 000 XAF**

## Contraintes

Les contraintes font référence aux exigences qui nous ont été données pour le travail que nous devons fournir. Pour ce faire nous pouvons les classer de la manière suivante :

❖ **Le délai** : notre projet est valide uniquement si les livrables suivant sont terminer avant la limite.

❖ **La qualité du projet :** la contrainte de qualité constitue un aspect très important. Etant donné qu’une plateforme ne peut être totalement parfaite, elle devrait se rapprocher du zéro défaut. Pour plus de sécurité, l’application devra être :

* **Exacte** : l’aptitude d’une plateforme à fournir des résultats attendus dans les conditions normales d’utilisation ;
* **Robuste** : l’aptitude à bien réagir lorsqu’on s’éloigne des conditions normales d’utilisation ;
* **Efficiente** : temps d’exécution, taille mémoire…

## Les livrables attendus

* Code source complet
* Base de données
* Documentation technique
* Manuel utilisateur (PDF ou en ligne)
* Vidéo de démonstration

## Conclusion

Au terme de cette partie consacrer au cahier de charge dans lequel nous avons précisé les objectifs à atteindre afin d’apporter satisfaction à l’utilisateur, un accent a été porter sur les ressource et contraintes nécessaires à la réalisation de ce projet et enfin un planning provisionnel a été proposer.

# CONCEPTION

## Introduction

La **phase de conception** intervient après l’analyse des besoins. Elle permet de structurer techniquement le projet afin de faciliter sa réalisation. Elle sert à modéliser les composants du système et leurs interactions en s'appuyant sur des diagrammes UML.

## Objectifs de la conception

Pour faciliter et normaliser l’analyse et la conception des systèmes d’information, plusieurs méthodes et langages ont été développés. En ce qui concerne la méthode que nous allons utiliser il s’agit de celle directement rattachée au langage de modélisation UML ( Unified Modeling Language) .

La conception UML permet entre autres de :

* Définir l’architecture du système.
* Modéliser les entités, les acteurs, et leurs interactions.
* Assurer la cohérence entre les besoins exprimés et l’implémentation.
* Etablir les différentes tables de données et ressortir la base de données.
* Faciliter le travail des développeurs par une vision claire du système.

## Présentation des diagrammes

### Diagramme de cas d’utilisation

#### Définition

Ce diagramme permet de représenter les interactions entre les acteurs (utilisateurs ou systèmes externes) et le système. Il met en évidence les fonctionnalités principales que le système doit offrir, sans entrer dans les détails techniques. Il sert à définir les besoins fonctionnels du projet et constitue une base de communication entre les développeurs et les utilisateurs. Dans le cadre de notre projet, ce diagramme a permis de visualiser les différents cas d’utilisation tels que :

#### Formalisme du diagramme de cas d’utilisation

Tableau 6 : Formalisme diagramme de cas d’utilisation

|  |  |  |
| --- | --- | --- |
| **COMPOSANTS** | **DESCRIPTIONS** | **FORMALISMES** |
| **Acteur** | C’est un type stéréotypé représentant une abstraction qui réside juste en dehors du système à modéliser. Un acteur représente un rôle joué par une personne ou une chose qui interagit avec le système.  Pour identifier les acteurs, il faut donc se concentrer sur les rôles joués par les entités extérieures au périmètre. Il existe deux (2) catégories d’acteurs à savoir :   * Les acteurs principaux : les personnes qui utilisent les fonctions principales du système ; * Les acteurs secondaires : qui sont des acteurs qui n’effectuent pas principalement une fonctionnalité, mais participent à la réalisation de cette dernière. |  |
| **Cas d’utilisation** | Un cas d’utilisation représente une fonctionnalité fournie par le système. Il (use case) correspond à un objectif du système, motivé par un besoin d’un ou plusieurs acteurs. Les cas d’utilisation sont représentés par une ellipse contenant leur nom. |  |
| **Association** | Les associations sont utilisées pour lier des acteurs avec des cas d’utilisation. Elles indiquent qu’un acteur participe au cas d’utilisation sous une forme quelconque. |  |
| **Extension** | La relation d’extension (« extend ») permet d’étendre les interactions. Elle désigne qu’un cas d’utilisation peut fonctionner tout seul mais peut aussi être complété par d’autres sous certaines conditions. |  |
| **Inclusion** | La relation d’inclusion (« include ») désigne qu’un cas d’utilisation dépend d’un autre : c’est-à-dire que pour pouvoir exécuter un cas donné, il faudrait obligatoirement exécuter un autre cas. |  |
| **Héritage** | Il permet de montrer qu’un acteur ou cas d’utilisation est une généralisation d’autres acteurs ou cas d’utilisation qui eux sont des spécialisations. |  |
| **Relation entre acteur et cas d’utilisation** | Un acteur interagit avec le système grâce au cas d’utilisation. C’est une relation qui va donc de ce dernier vers son cas. |  |

#### Diagrammes de cas d’utilisation de notre système

* Cas utilisation Client

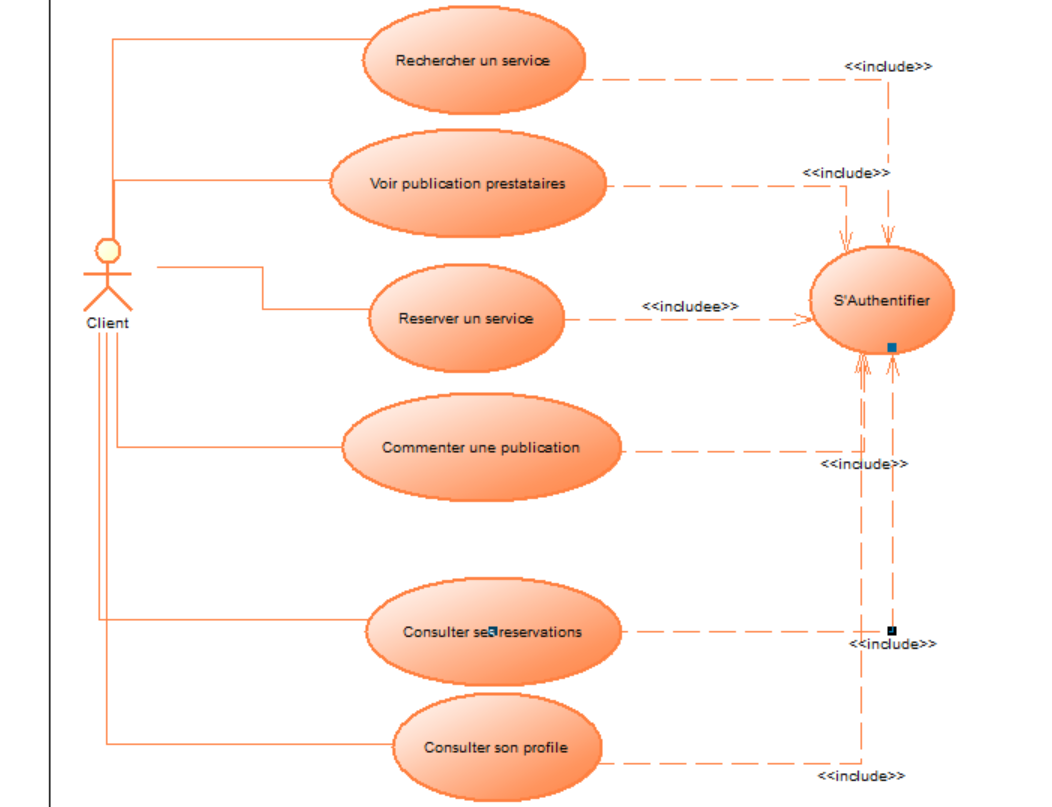


Figure 6 : Diagramme de cas d’utilisation pour client

* Cas utilisation Prestataire

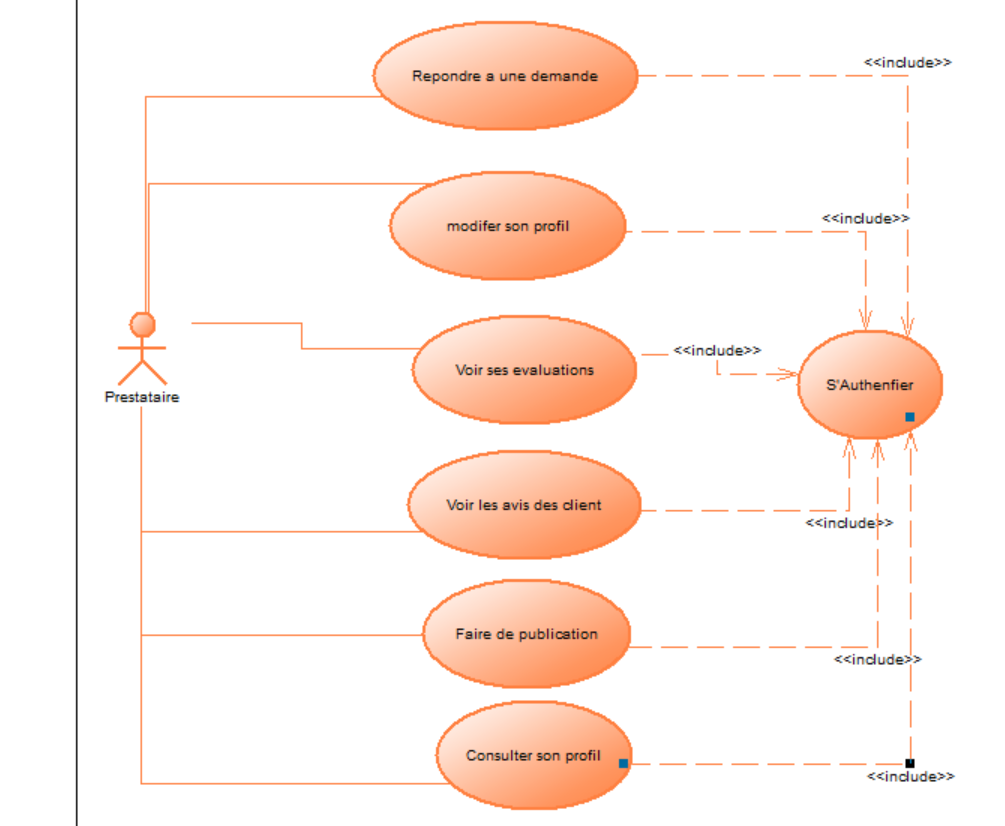


Figure 7 : Diagramme de cas d’utilisation pour prestataire

* Pour l’Administrateur

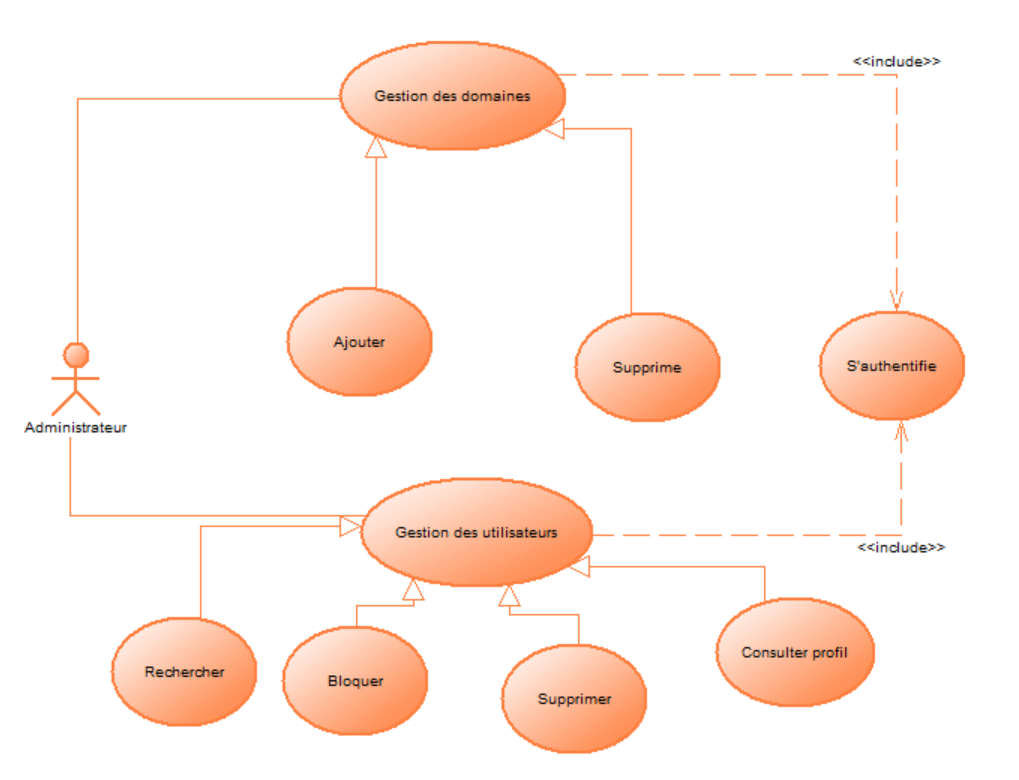


Figure 8 : Diagramme de cas d’utilisation pour Administrateur

### Diagramme de classes

#### Définition

Un diagramme de classe permet de présenter les différents objets d’un système rassemblé sous forme de classe. Il met en exergue les différentes collaborations, et relations entre les différentes classes. C’est un diagramme statique qui fournit une vue globale du système, illustre les échanges entre les objets lors d’un scénario clé (ex : réservation d’un prestataire).

#### Formalisme du diagramme de classe

|  |  |  |
| --- | --- | --- |
| **COMPOSANTS** | **DESCRIPTIONS** | **FORMALISMES** |
| **La classe** | Une classe est un type abstrait caractérisé par des propriétés (attributs et méthodes) communes à un ensemble d’objets et permettant de créer des objets ayant ces propriétés. |  |
| **Les attributs** | Un attribut représente la modélisation d’une information élémentaire représentée par son nom et son format. |  |
| **L’identification de la classe** | C’est un attribut particulier permettant d’identifier de façon unique une classe. |  |
| **Méthode** | Une opération est une fonctionnalité d’une classe. Elle peut inclure des paramètres d’entrée, de sortie, et décrire les actions à exécuter. |  |
| **TYPES DE RELATION** | |  |
| **Association** | Représente une relation structurelle entre deux ou plusieurs classes. |  |
| **Agrégation** | Représente une relation où une classe est constituée d’objets d’une autre classe. |  |
| **Composition** | C’est une agrégation forte : la vie du composant dépend de celle de l’agrégat. |  |
| **Généralisation/ spécification** | Relations dans laquelle des classes appelées classes filles héritant des attributs de méthodes d’une autre classe appelé classe mère. |  |

Tableau 7 :Formalisme du diagramme de classe

* **Identifications des classes de notre système**
* Utilisateur
* Prestataire
* Client
* Service
* Demande
* Avis
* Paiement
* **Règles métiers**

|  |  |
| --- | --- |
| **Règles** | **Relations** |
| 1 | Un utilisateur peut être client ou prestataire. |
| 2 | Un prestataire peut offrir plusieurs services. |
| 3 | Un client peut faire plusieurs demandes. |
| 4 | Une demande est liée à un service. |
| 5 | Un client peut laisser plusieurs avis. |
| 6 | Un paiement est lié à une demande. |

Tableau 8 :Règles metiers

#### Diagramme de classe de la solution

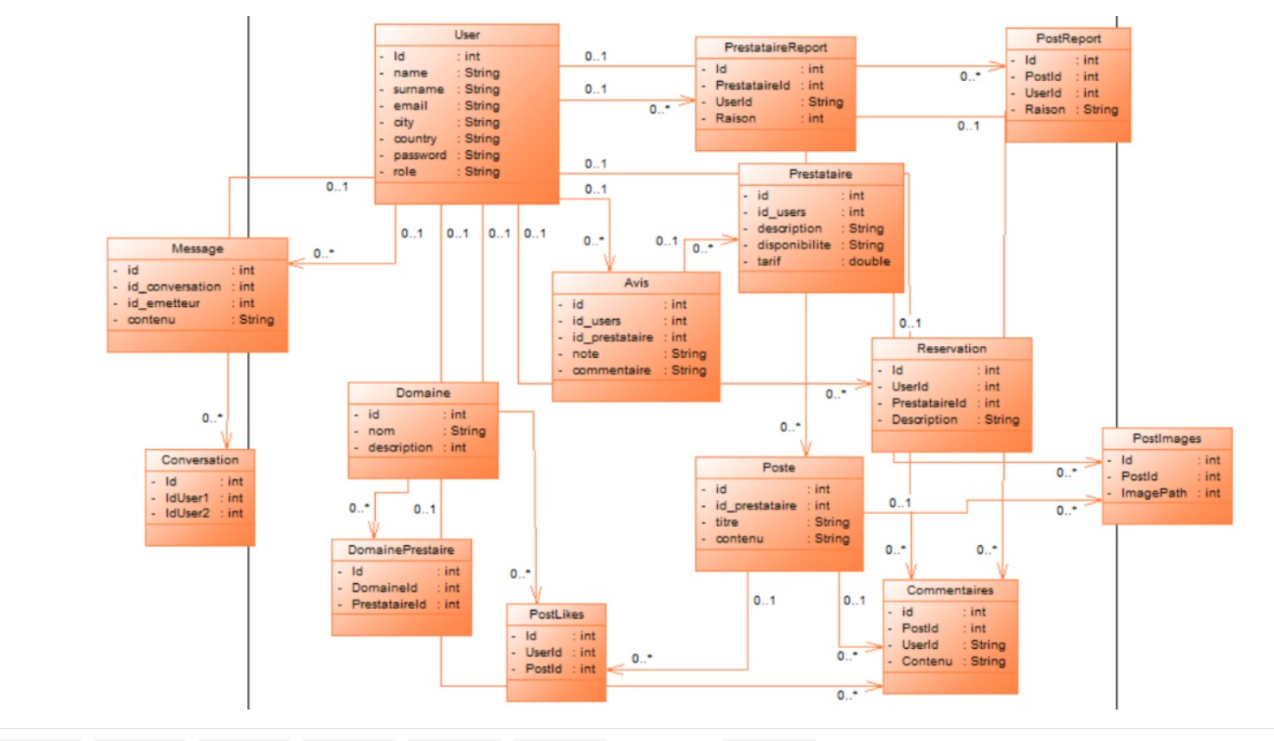


Figure 9 : Diagramme de classe

#### Implémentation de la base de données MySQL

**MySQL** est un système de gestion de base de données relationnelle (SGBDR) très utilisé, notamment dans les applications web. Il repose sur le langage SQL (Structured Query Language) pour la gestion, la manipulation et l'interrogation des données. MySQL se distingue par sa robustesse, sa stabilité et sa capacité à gérer efficacement de grands volumes de données relationnelles.

Le stockage des données dans MySQL se fait sous forme de bases de données, qui contiennent des tables. Chaque table est composée de lignes (enregistrements) et de colonnes (attributs), représentant respectivement les entités et leurs propriétés. Les relations entre les tables sont définies à l'aide de clés primaires et clés étrangères, permettant de structurer les données de manière cohérente et normalisée.

Ce modèle relationnel facilite la cohérence, l’intégrité et la sécurité des données, tout en permettant des opérations complexes comme les jointures, les transactions et les contraintes d'intégrité.

### Diagramme de séquence

#### Définition

Ce diagramme permet de modéliser le déroulement chronologique des interactions entre les objets ou acteurs dans un scénario donné. Il montre l’échange de messages dans le temps, mettant en lumière le comportement dynamique du système. Pour notre plateforme, le diagramme de séquence a servi à représenter des scénarios précis comme : le processus d’inscription, la validation d’une commande ou encore le suivi de livraison.

#### Formalisme du diagramme de séquence

|  |  |  |
| --- | --- | --- |
| **COMPOSANTS** | **DESCRIPTIONS** | **FORMALISMES** |
| **Objet** | Les objets sont des instances des classes et sont rangés horizontalement.  La représentation graphique pour un objet est similaire à une classe (un rectangle) précédée du nom d'objet (Facultatif).  Formalisme : rectangle avec « : Objet » |  |
| **Acteur** | Les acteurs peuvent également communiquer avec des objets, ainsi ils peuvent eux aussi être énumérés en colonne. Un acteur est modélisé en utilisant le symbole habituel. |  |
| **Ligne de vie** | Les lignes de vie, Life Line, identifient l'existence de l'objet par rapport au temps. La notation utilisée pour une ligne de vie est une ligne pointillée verticale partant de l'objet. |  |
| **Activation** | Les activations sont modélisées par des boîtes rectangulaires sur la ligne de vie. Elles indiquent quand l'objet effectue une action. |  |
| **Messages synchrone** | Ce type de message est représenté par une flèche pleine avec une tête fermée. Lorsqu’un objet A envoie un message synchrone à un objet B, A reste bloqué tant que B n’a pas terminé. |  |
| **Messages asynchrone** | Ce type de message est représenté par une flèche pleine avec une tête ouverte. Un message asynchrone n’est pas bloquant pour l’expéditeur. |  |
| **Message de retour** | Il est généralement associé aux messages d’appel de méthode et est représenté par une ligne en pointillés. Il marque la reprise du contrôle par l’objet émetteur du message synchrone. |  |
| **Messages réflexif** | Un message réflexif dans un diagramme de séquence UML est un type de message qui est envoyé par un objet à lui-même. Cela peut représenter une méthode ou une opération que l’objet appelle sur lui-même. |  |

Tableau 9 :Formalisme du diagramme de sequence

#### Diagrammes de séquence de notre système

* Pour l’authentification

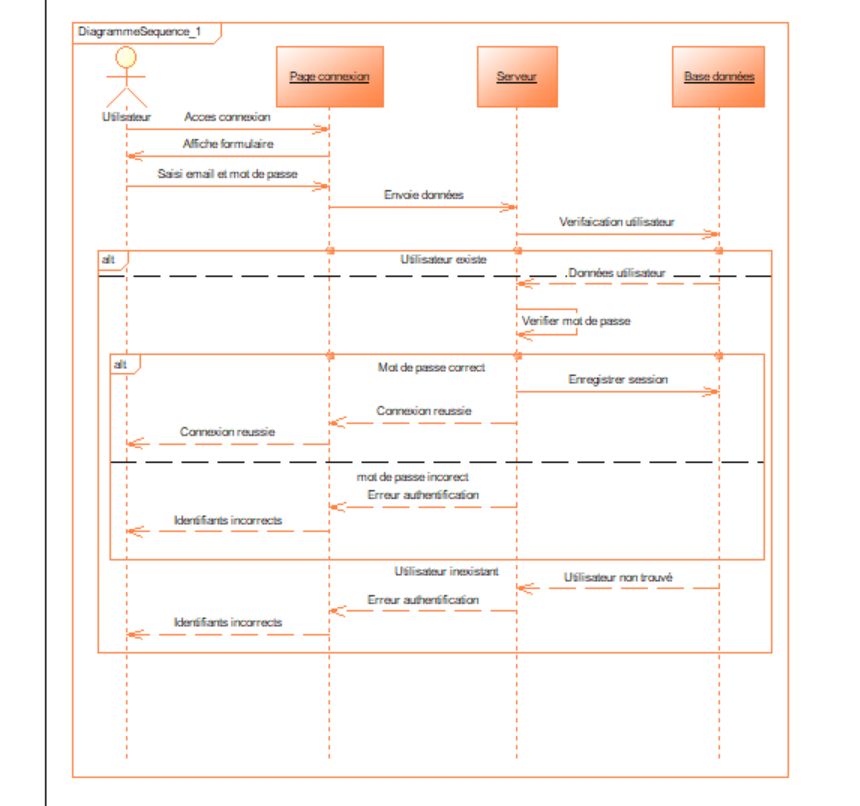


Figure 10 : Diagramme de séquence pour authentification

* Pour la recherche d’un prestataire

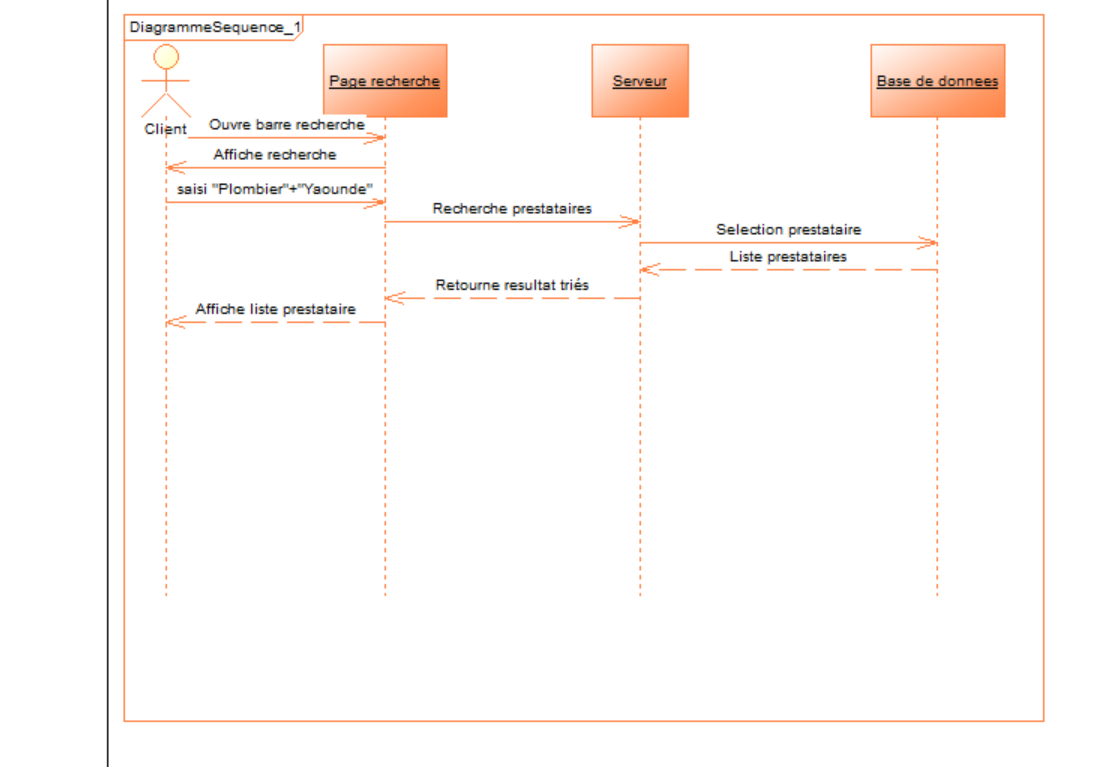


Figure 11 : Diagramme de séquence pour recherche d’un prestataire

### Diagramme d’activités

#### Définition

Le diagramme d’activité est un diagramme UML qui décrit le déroulement d’un processus ou d’un scénario métier. Il montre les différentes étapes (activités), les décisions possibles, les conditions de transition et les flux entre les actions. Il sert à visualiser la logique d’un traitement, de façon claire et séquentielle ; il modélise le déroulement logique d’un processus (ex : création de compte, réservation).

#### Formalisme du diagramme d’activité

|  |  |  |
| --- | --- | --- |
| **COMPOSANTS** | **DESCRIPTIONS** | **FORMALISMES** |
| **Activité** | L’état d’activité marque une action faite par un objet. Il est représenté par un rectangle. |  |
| **Transition** | Quand un état d’activité est accompli, le traitement passe à un autre état d’activité. Les transitions sont utilisées pour marquer ce passage. Les transitions sont modélisées par des flèches. |  |
| **Etat** | L’état initial marque le point d’entrée de la première activité. Il est représenté, comme dans le diagramme d’état, par un cercle plein. Il ne peut y avoir qu’un seul état initial sur un diagramme. |  |
| **Etat final** | L’état final marque la fin du déroulement des opérations modélisées. Il peut y avoir des états finaux multiples sur un diagramme. Il en existe deux types :  - Fin de flux : Représente la fin d’une interaction et marque l’arrêt du système sur un échec de l’opération enclenchée. Son formalisme est un cercle contenant une croix.  - Fin d’activité : Représente généralement la fin du déroulement d’un système qui s’achève sur le succès de l’opération enclenchée. Son formalisme est un cercle plein contenu dans un autre cercle. |  |
| **Barre de synchronisation** | Souvent, certaines activités peuvent être faites en parallèle. Pour dédoubler le traitement, ou le reprendre quand des activités multiples ont été accomplies, des barres de synchronisation sont utilisées.  Celles-ci sont modélisées par des rectangles pleins, avec des transitions multiples entrantes ou sortantes. |  |
| **Décision** | Une décision est le résultat final d’une ou plusieurs activités. | Décisionn  Figure 12 |

Tableau 10 :Formalisme du diagramme d’activité

#### Diagrammes d’activité de notre système

* Pour la connexion

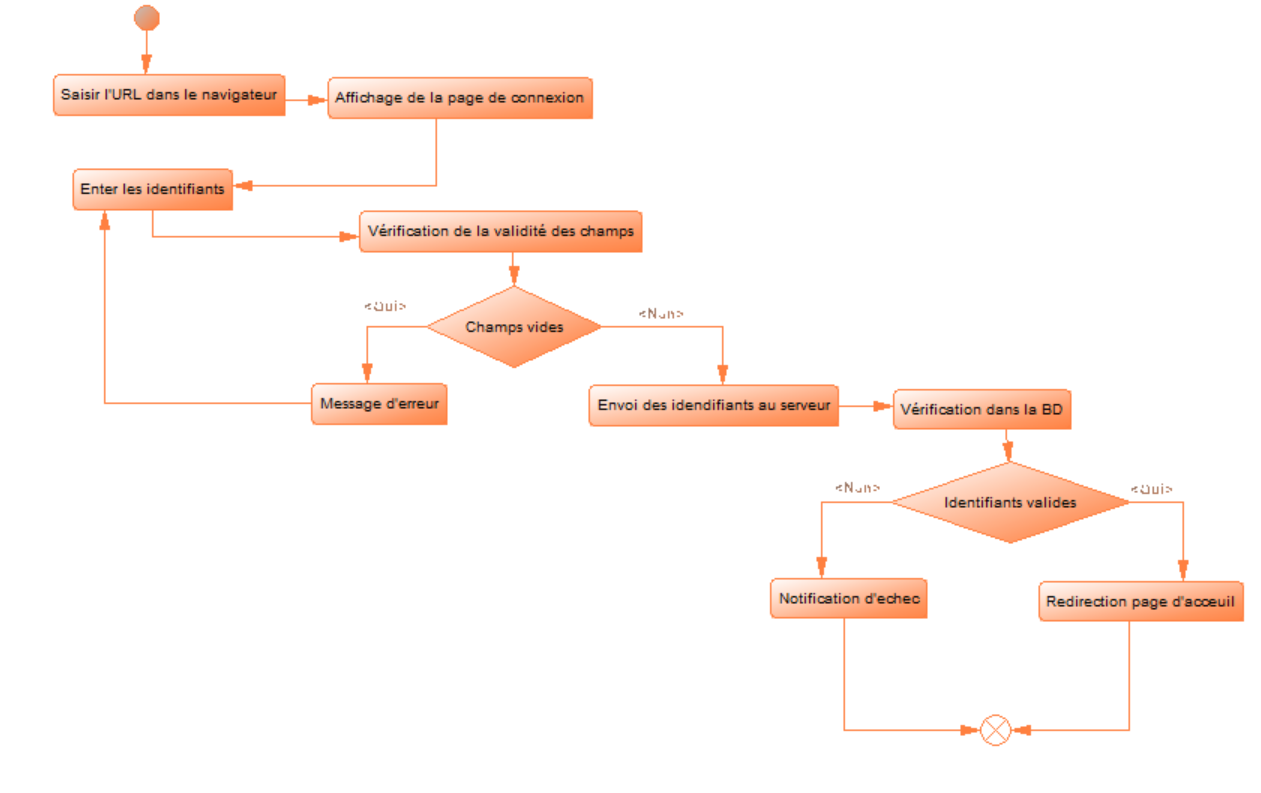


Figure 13 : Diagramme d’activité pour authentification

* Pour le recherche d’un prestataire

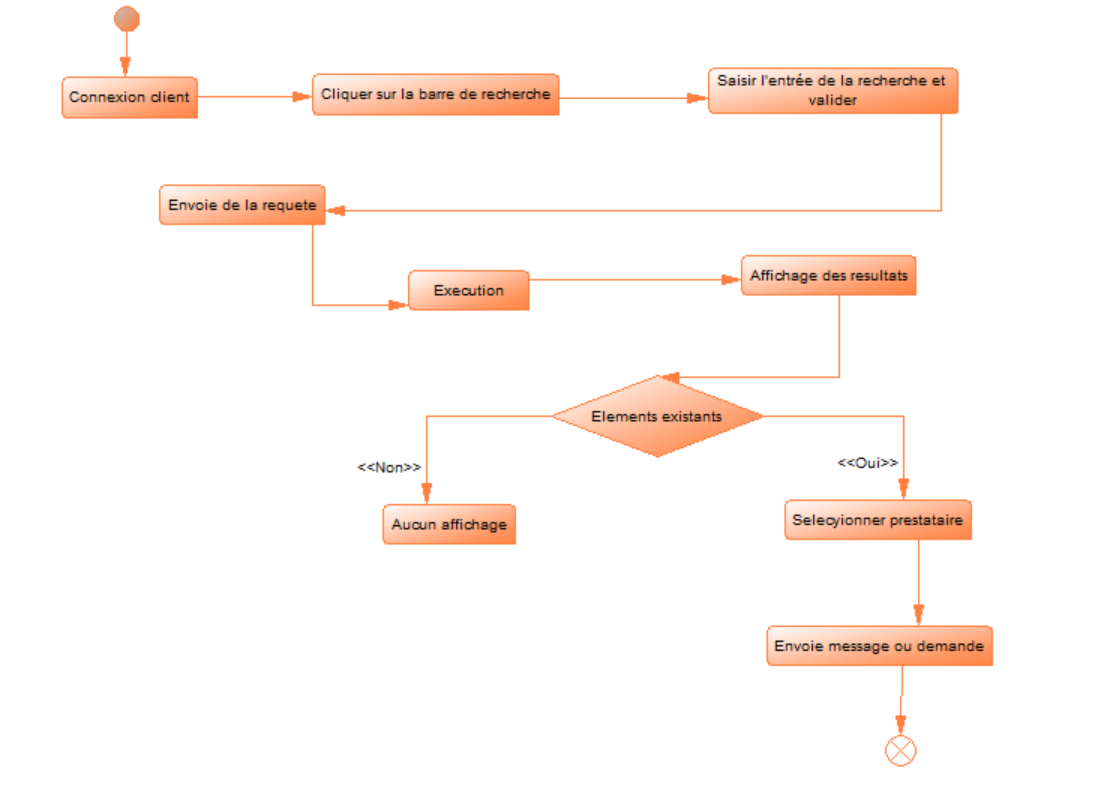


Figure 14 : Diagramme d’activité pour recherche d’un prestataire

### Diagramme d’objet

#### Définition

Un diagramme d’objets UML est un type de diagramme qui représente une instance spécifique d’un diagramme de classes à un moment précis. Il permet d’avoir une vue d’ensemble de votre système et de décrire les objets qui s’y trouvent. Dans sa représentation visuelle, il est très similaire à un diagramme de classes. Les diagrammes d’objets offrent une image instantanée des instances d’un système et des relations entre ces instances. Les diagrammes UML, dont les diagrammes d’objets font partie, servent à visualiser et décrire la structure et le comportement des objets qui se trouvent dans un système.

#### Formalismes du diagramme d’objet

|  |  |  |
| --- | --- | --- |
| **COMPOSANTS** | **DESCRIPTIONS** | **FORMALISMES** |
| **Objet** | Représente l’instance d’une classe (état d’une classe à un instant précis) | |  | | --- | | Objet\_1 | |  | |
| **Lien** | Représente la relation entre les objets à un instant donné |  |
| **Dépendance** | Représente une relation dans laquelle un élément tire parti d’un autre élément du système |  |

Tableau 11 :Formalisme diagramme d’objet

#### Diagramme d’objet du système

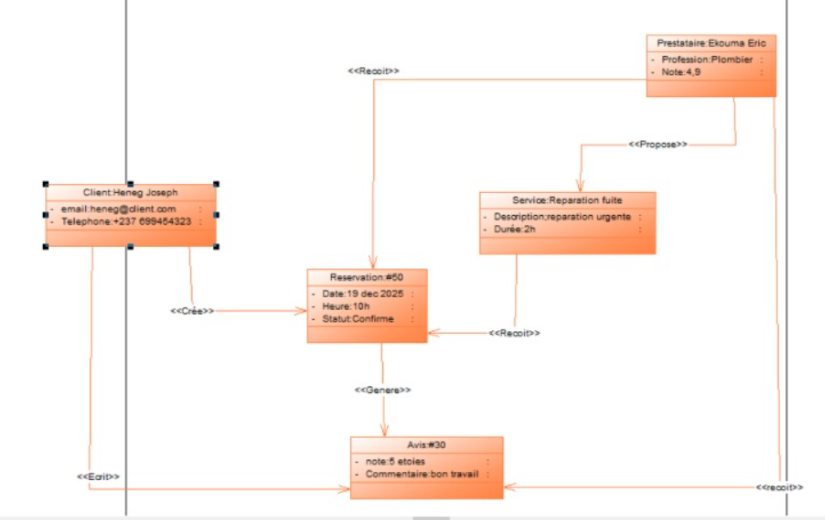


Figure 15 : Diagramme d’objet

### Diagramme de paquetage

#### Définition

Un diagramme de paquetage UML est un type de diagramme de structure qui représente les relations entre les paquetages d’un système logiciel. Les paquetages sont des regroupements d’éléments UML tels que des classes, des documents, des autres paquetages, etc. Les éléments du diagramme sont imbriqués dans des paquetages, qui sont représentés sous forme de dossiers de fichiers et organisés de manière hiérarchique. Les diagrammes de paquetages sont souvent utilisés pour donner un aperçu visuel de l’architecture en couches d’un classifieur UML, tel qu’un système logiciel.

#### Diagramme de paquetage du système

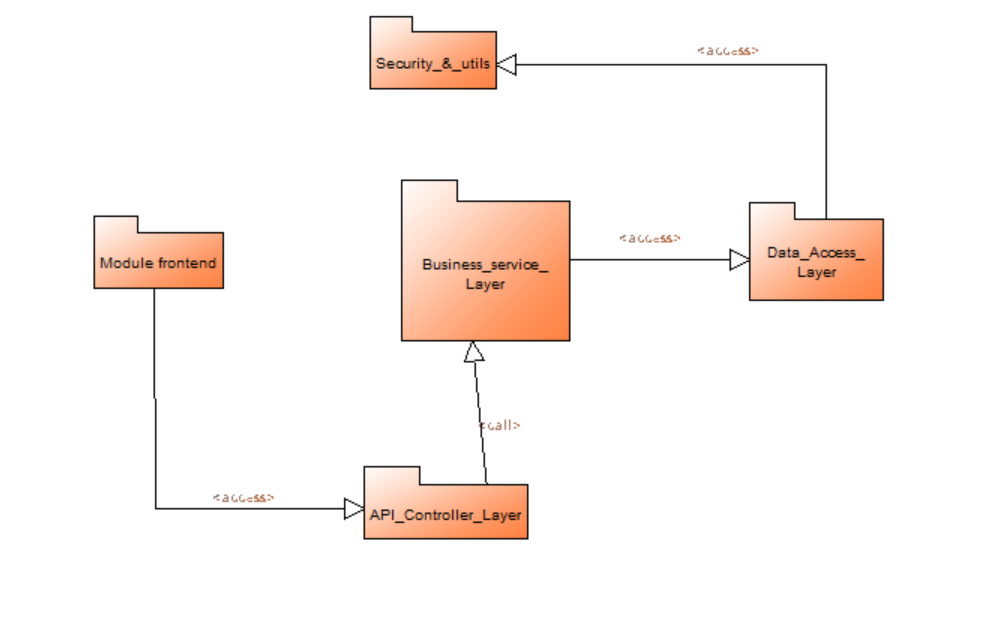


Figure 16 : Diagramme de paquetage

### Diagramme d’état transition

#### Définition

Un diagramme d’états-transitions UML est un type de diagramme comportemental qui montre les transitions entre différents états d’un objet. Il est utilisé pour décrire le comportement interne d’un objet à l’aide d’un automate à états finis. Les états sont représentés par des formes rectangulaires avec des coins arrondis et les transitions sont représentées par des flèches qui relient les états. Les événements qui déclenchent les transitions sont représentés sur les flèches. Les diagrammes d’états-transitions sont souvent utilisés pour modéliser des systèmes réactifs, tels que des systèmes de contrôle de processus ou des systèmes de contrôle de trafic. Ils sont également utilisés pour modéliser des objets qui ont un comportement complexe et qui passent par plusieurs états différents.

#### Formalisme

|  |  |  |
| --- | --- | --- |
| **COMPOSANTS** | **DESCRIPTIONS** | **FORMALISMES** |
| **Etat initial** | Représenté par un cercle plein, il marque l’état d’un objet avant toute transition. |  |
| **Etat** | L’état est une situation durable dans laquelle se trouve un objet. |  |
| **Transition** | Il permet de modéliser le changement d’état d’un objet provoqué généralement par un événement. Elle est symbolisée par une flèche orientée. |  |
| **Etat final** | Il permet de représenter la destruction d’un objet. |  |

Tableau 12 :Formalisme du diagramme d’état-transition

1. Diagramme état-transition

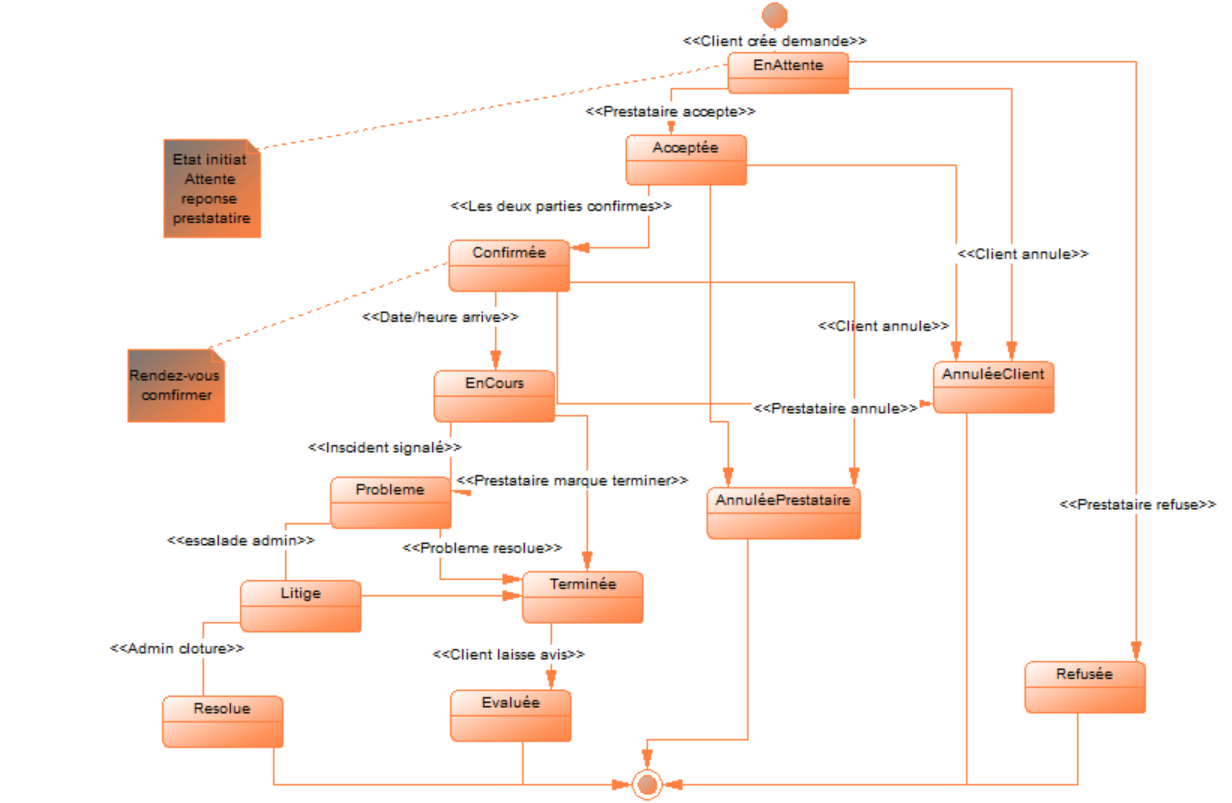


Figure 17 : Diagramme d’état transition

### Diagramme de communication

#### Définition

Un diagramme de communication est un type de diagramme UML qui montre les interactions entre objets ou composants dans un système, en mettant l'accent sur les échanges de messages entre eux. Il illustre comment les objets collaborent pour accomplir une fonction ou un processus en représentant les messages échangés dans un ordre spécifique. Ce diagramme est souvent utilisé pour comprendre et modéliser la dynamique des interactions dans un système logiciel.

#### Présentation du diagramme de communication de notre système

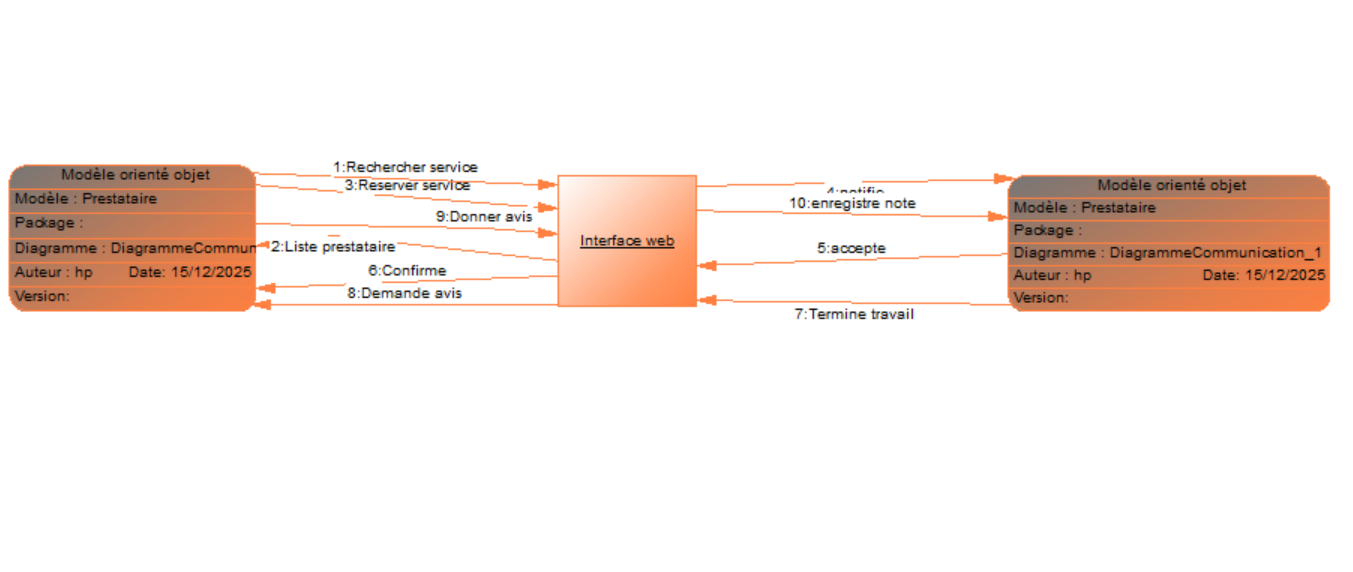


Figure 18 : Diagramme de communication

## Conclusion

Il était question pour nous de ressortir la structure des données de notre base de données. A l’issu de cette phase, nous avons pu ressortir les différentes classes et relations entre elle.

# REALISATION

La réalisation consiste à l’implémentation du projet en s’appuyant sur les technologies et langages choisis. A l’issu de cette phase de réalisation, les diagrammes tels que le diagramme de composant et de déploiement seront établit afin de mettre en évidence l’architecture du logiciel.

## Diagramme de déploiement

### Définition

Le diagramme de déploiement modélise les composants matériels utilisés pour implémenter un système et l’association entre ces composants. Il peut être mit en œuvre dès la phase de conception pour documenter l’architecture physique du système.

### Formalisme

|  |  |  |
| --- | --- | --- |
| **COMPOSANTS** | **DESCRIPTIONS** | **FORMALISMES** |
| **Composant** | Il modélise une entité logicielle du système. |  |
| **Nœud** | Le nœud représente un ensemble d’éléments matériels du système. |  |
| **Association** | Permet de montrer le lien de communication entre les nœuds. |  |
| **Dépendance** | Il permet de traduire un lien de dépendance entre des nœuds. |  |

Tableau 13 :Formalisme du diagramme de deploiement

### Représentation graphique du système du diagramme de déploiement

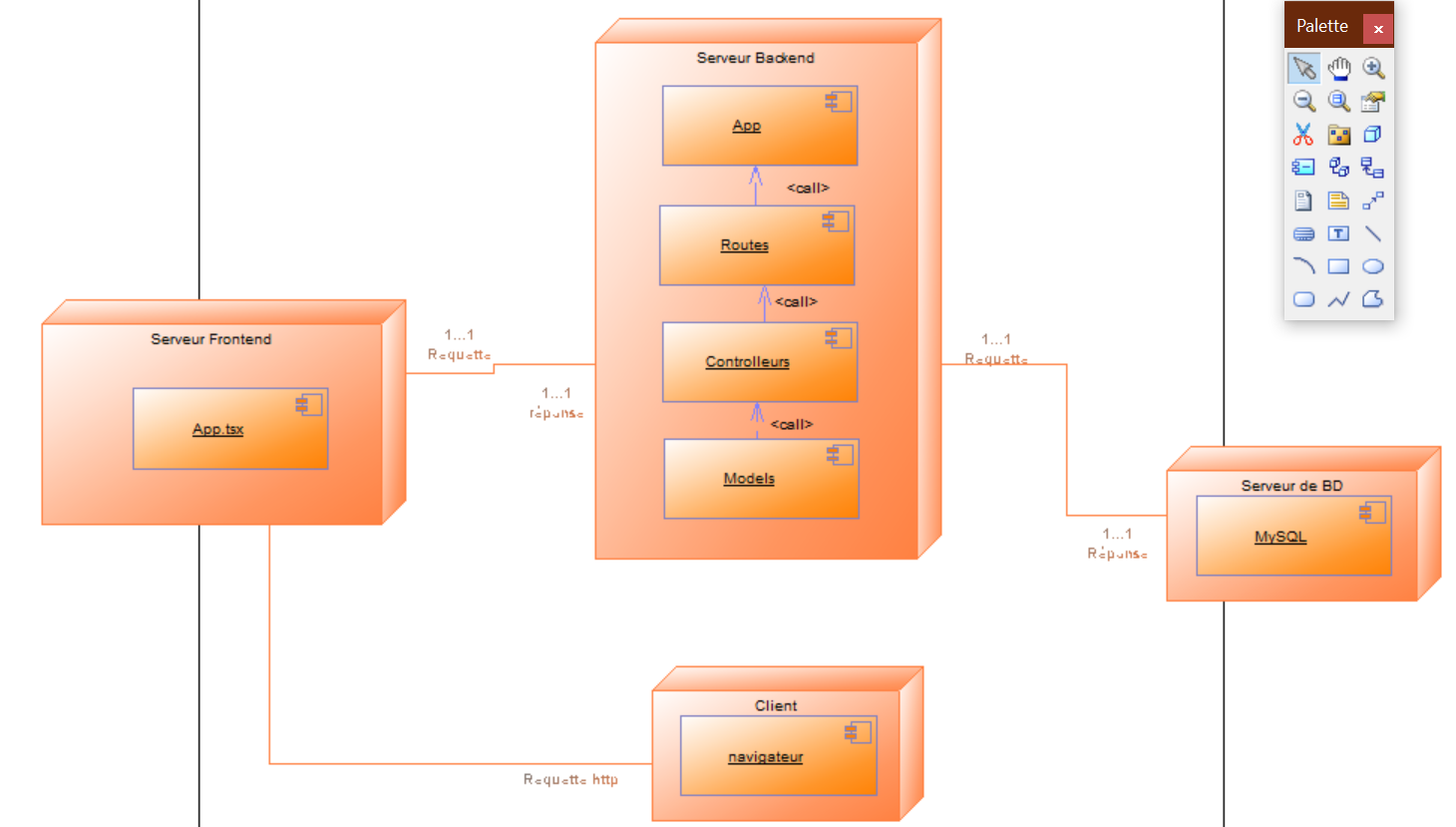


Figure 19 : Diagramme de deploiement

## Diagramme de composant

### Définition

Le diagramme de composant décrit le système modélisé sous forme de composants réutilisables comme les données, modules, éléments de configuration et mettent en évidence leurs relations de dépendance.

### Formalismes

|  |  |  |
| --- | --- | --- |
| **COMPOSANTS** | **DESCRIPTIONS** | **FORMATIONS** |
| **Composants** | Entité (représentée par un classeur structuré) tenue d’exécuter une fonction précise |  |
| **Port** | Point de connexion entre un classeur et son environnement. |  |
| **Interface requise** | Interfaces où un composant produit des informations utilisées par l’interface requise d’un autre composant. |  |
| **Interface fournie** | Interface où un composant produit des informations utilisées par l’interface requise d’un autre composant. |  |
| **Dépendance** | Indique la hiérarchie d’utilisation entre les différentes parties du système. |  |

Tableau 14 :Formalisme du diagramme de composant

### Présentation du diagramme de composant

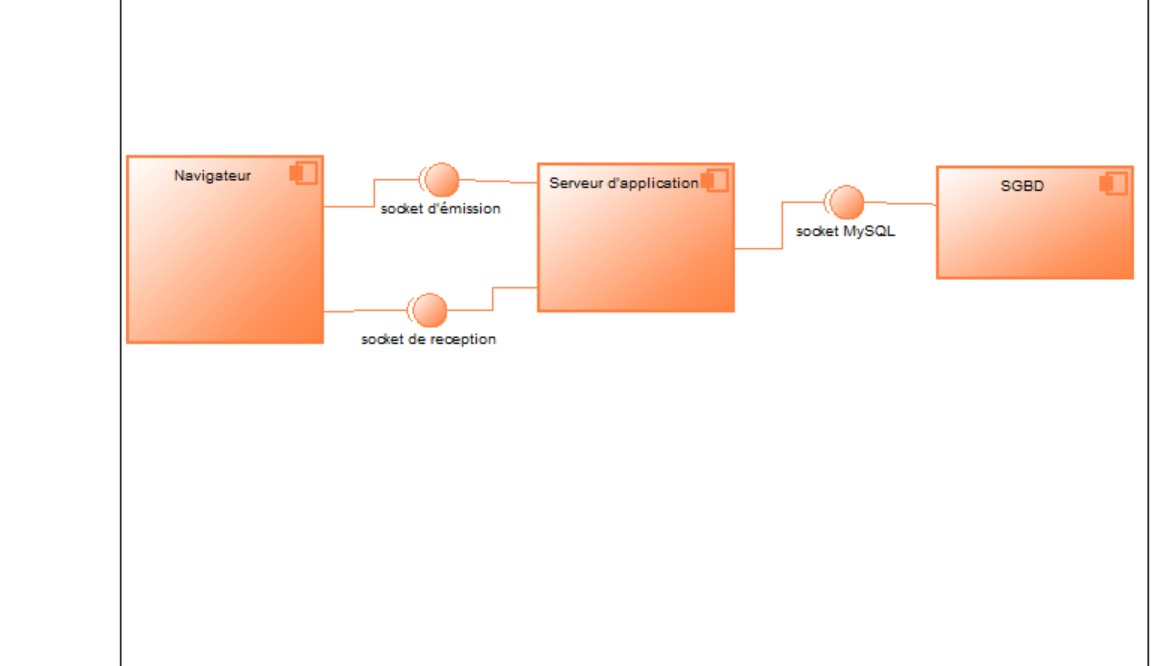


Figure 20 : Diagramme de commposant

## Présentation de l’architecture du système

### Architecture physique

Notre site web est une application informatique structurée en trois (03) couches à savoir : la couche présentation, la couche traitement, et la couche donnée. Le noyau d'une application est constitué de la logique de présentation et de la logique de traitement, et c'est la répartition de ce noyau entre les différentes couches qui détermine les architectures caractéristiques des systèmes client/serveurs, nous pouvons citer l’architecture 1-tiers, 2-tiers, 3-tiers. Notre plateforme sera déployée sur une architecture 3-tiers. Cette architecture peut être décrite par la figure ci-dessous :

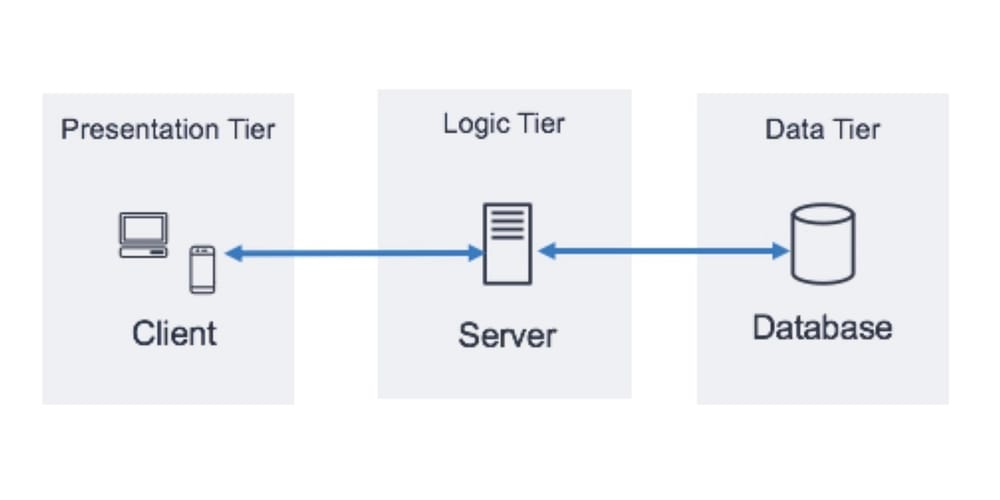


Figure 21 : Architecture physique

### Architecture logique

La logique applicative de notre projet est basée sur le design pattern MVC qui est une façon très pratique d’organiser ses différentes classes dans un projet de développement d’application facilitant ainsi la détection d’erreurs, la collaboration et la réutilisabilité du code.

Le design pattern MVC (Modèle Vue Contrôleur) est une architecture et une méthode de conception qui organise l’interface homme-machine (IHM) d’une application logicielle. Ce paradigme divise l’IHM en un modèle (modèle de données), une vue (interface utilisateur) et un contrôleur (logique de contrôle, gestion des événements, synchronisation), chacun ayant un rôle précis dans l’interface.

Le design pattern MVC permet de bien organiser son code source. Il va vous aider à savoir quels fichiers créer, mais surtout à définir leurs rôles. Le but du MVC est justement de séparer la logique du code en trois parties que l’on retrouve dans des fichiers distincts.

* **Modèle** : cette partie gère les données de votre application. Son rôle est d’aller récupérer les informations « brutes » dans une base de données locale ou distante, de les organiser.
* **Vue** : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher.
* **Contrôleur** : cette partie gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue.

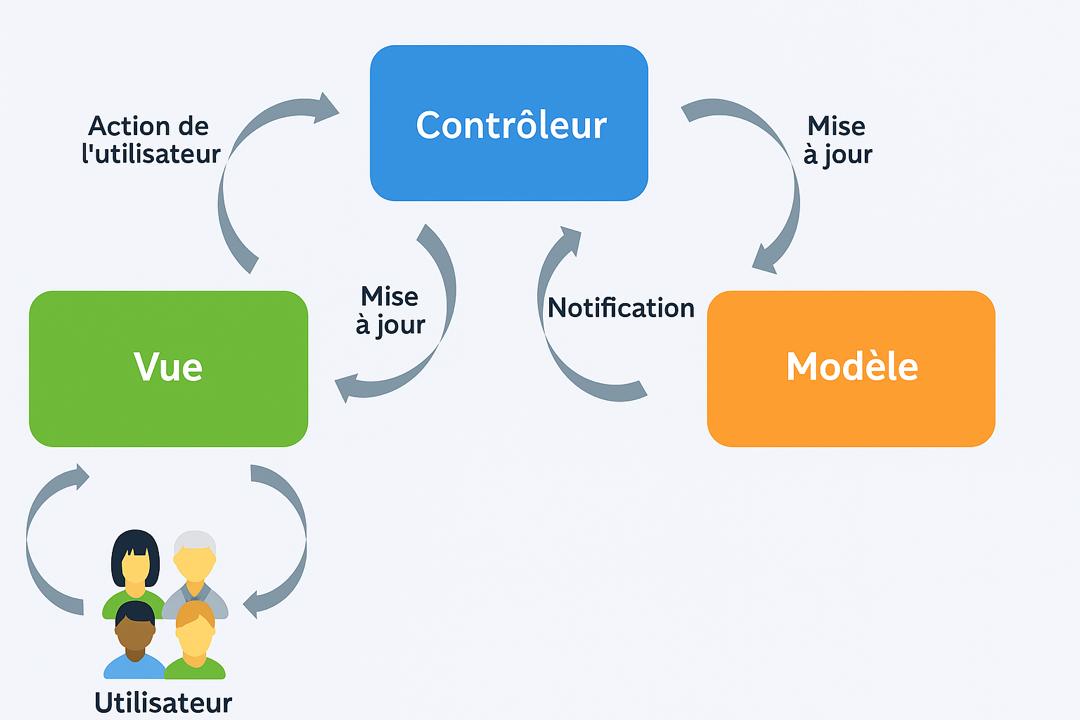


Figure 22 : Architecture logique

## Méthodologie de développement

Pour la gestion et l'organisation du développement de notre projet de service local, notre équipe a adopté la méthode Scrum, une méthode agile particulièrement adaptée aux projets logiciels en constante évolution. Cette approche nous a permis de structurer notre travail en cycles courts appelés sprints, favorisant ainsi une livraison progressive et régulière des fonctionnalités.

Durant chaque sprint, nous nous sommes concentrés sur le développement et le test de fonctionnalités précises de l’application, qu’il s’agisse de la version Web ou de la Progressive Web App (PWA) pour mobile. La nature itérative de Scrum a permis d’intégrer rapidement les retours utilisateurs et d’ajuster les priorités en fonction des besoins réels.

L’utilisation de Scrum nous a également aidés à maintenir une communication fluide au sein de l’équipe grâce aux réunions quotidiennes (daily scrum), durant lesquelles nous faisions le point sur l’avancement, les difficultés rencontrées et les prochaines tâches à réaliser. Par ailleurs, pour garantir la qualité et la robustesse de l’application, nous avons intégré des tests unitaires automatisés à l’aide de Robot Framework. Ces tests, réalisés à chaque sprint, nous ont permis de détecter rapidement les bugs et de valider le bon fonctionnement des différentes parties du système.

## Présentation de l’environnement

### Environnement logiciel

Dans le tableau suivant, nous présenterons de façon explicite chacun des outils logiciels intervenant dans la réalisation de notre plateforme.

|  |  |  |
| --- | --- | --- |
| **Nom du logiciel** | **Utilisation** | **Logo** |
| **Windows 10** | Système d’exploitation nécessaire pour exécuter les logiciels. | C:\Users\AFC\Desktop\rapport de fin de stage\5c6c7d03eec9703a40aee1e451d2e867.jpg |
| **Google chrome** | Navigateur web nécessaire pour utiliser notre plateforme ; nous l’avons utilisé pour nos différents tests. | C:\Users\AFC\Desktop\rapport de fin de stage\5f9842993e60daf1f2ca8c23900317a6.jpg |
| **GanttProject** | Logiciel de gestion de projet qui permet de planifier, d’organiser et de suivre les projets en utilisant le diagramme de Gantt | C:\Users\AFC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG-20251218-WA0034.jpg |
| **Visual studio code** | Editeur de code utilisé pour le codage de la plateforme ; ce logiciel nous a permis d’éditer le code source de notre plateforme. | C:\Users\AFC\Desktop\rapport de fin de stage\c0316a58e22769031fddf28f9d9b8539.jpg |
| **Github** | Plateforme open source de gestion de versions et de collaboration destinée aux développeurs de logiciels. | C:\Users\AFC\Desktop\rapport de fin de stage\b2c5e65384b1b0ea4cd351b6155e1b29.jpg |
| **Power AMC** | Atelier de génie logiciel utilisé pour la modélisation de la solution. Nous nous sommes servis de cet outil pour la modélisation des différents diagrammes de notre sytème. | C:\Users\AFC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG-20251218-WA0033.jpg |
| **NodeJS** | Environnement permettant d’exécuter du code JavaScript hors d’un navigateur | C:\Users\AFC\AppData\Local\Microsoft\Windows\INetCache\Content.Word\IMG-20251218-WA0032.jpg |

Tableau 15 :Technologies utisées

### Technologies utilisées

Pour la mise sur pied de notre site et selon les besoins utilisateurs finaux, les technologies choisies sont :

• **Frontend** : : React avec une architecture en Progressive Web App (PWA) pour assurer une expérience utilisateur fluide et optimisée sur les navigateurs web et les appareils mobiles, permettant l'installation comme une application native et un fonctionnement partiel hors ligne.  
• **Backend** : Laravel , API Auth  
• **Base de données** : MySQL

## Conclusion

Dans cette partie nous avons évoqué les différentes architectures physique et logicielle utilisé pour la réalisation de notre projet. Grace aux diagrammes de composant et de déploiement, nous avons pu mettre en exergue les différents composants tant matériels que logiciel de notre plateforme. De ce fait, il est important de se poser la question de savoir si les utilisateurs seront capables d’utiliser la plateforme sans éprouver les difficultés ? la prochaine articulation de notre travail est de fournir un guide d’utilisation pour la plateforme réalisée.

**TEST ET FONCTIONNALITES**

**Introduction**

La phase de tests et fonctionnalités est une étape importante dans le développement de logiciels et d'applications. Elle consiste à tester les différentes fonctionnalités de l'application pour s'assurer qu'elles fonctionnent correctement et répondent aux exigences spécifiées. Cette phase est généralement divisée en plusieurs étapes, telles que les tests d'unité, les tests de composants, les tests de fonction, les tests de système et les tests de non-régression.

1. **Fonctionnalités**

Dans notre application ils existent plusieurs fonctionnalités parmi lesquelles :

|  |  |  |
| --- | --- | --- |
| Fonctionnalités | Descriptions | Acteurs |
| Inscription | Permet aux utilisateurs (clients ou prestataires) de créer un compte ou de se connecter. | Client , prestataire, administrateur |
| Création de profil | Chaque utilisateur remplit ses informations personnelles, localisation, disponibilités (pour les prestataires), préférences, etc. | Client, prestataire |
| Recherche de services | Le client recherche un service (plomberie, ménage, livraison…) selon sa localisation. | Client |
| Publication de demande | Le client publie une demande détaillée (type de service, lieu, date, heure, budget). | Client |
| Proposition de services | Les prestataires peuvent répondre aux demandes publiées ou proposer directement leurs services. | Prestataire |
| Réservation / Acceptation | Le client choisit un prestataire et confirme la réservation. Le prestataire accepte ou refuse. | Client , prestataire |
| Notation / Avis | Après la mission, le client et le prestataire peuvent s’évaluer. | Client , pretataire |
| Gestion des utilisateurs | L’admin gère les comptes, signale ou supprime des utilisateurs en cas de problème. | Administrateur |

Tableau 16 :Fonctionnalité de l’application

1. **Tests logiciels**

**2.1 Définition**

Les tests logiciels sont une pratique essentielle du cycle de développement de logiciels visant à évaluer et vérifier la qualité, la fonctionnalité et la performance d’une application ou d’un système informatique. Il s’agit de processus systématiques et contrôlés qui consistent à exécuter des scénarios de test, des cas de test ou des scripts automatisés afin de détecter les défauts, les erreurs et les anomalies potentielles dans le logiciel.

**2.2 Objectif**

Les tests logiciels ont pour objectif principal de garantir que le logiciel fonctionne conformément aux spécifications, aux exigences et aux attentes définies. Ils visent à identifier les erreurs ou les dysfonctionnements afin de les résoudre avant la mise en production du logiciel ou lors de la maintenance du logiciel.

**2.3 Les types de tests**

Fonction de l’objectif que l’on veut atteindre on distingue plusieurs types de tests parmi lesquels nous avons :

- Tests unitaires : vérification du bon fonctionnement des composants individuels du logiciel.

- Tests d’intégration : vérification de l’interaction entre les différents modules ou composants du logiciel.

-Tests fonctionnels : vérification des fonctionnalités du logiciel conformément aux spécifications.

- Tests de performance : évaluation des performances du logiciel en termes de vitesse, de charge et de stabilité.

- Tests de sécurité : vérification de la sécurité du logiciel contre les attaques et les vulnérabilités.

Dans notre cas nous allons nous attarder sur les tests unitaires.

**2.4 Tests unitaires**

**a. Outils**

L’outil que nous avons choisi pour réaliser nos tests est Robot Framework. C’est un outil open-source dédié à l’automatisation des tests d’acceptation, des tests unitaires et d’intégration. Ce Framework est particulièrement adapté aux projets nécessitant une syntaxe claire, lisible et facile à maintenir. Robot Framework s’appuie sur une architecture **keyword-driven** (basée sur des mots-clés), ce qui facilite l’écriture des scénarios de test même pour les personnes n’ayant pas une forte expertise en programmation. Dans le cadre de notre projet, il a été utilisé pour tester les fonctionnalités essentielles telles que la vérification de la page d’accueil , la vérification de la navigation vers la page d’accueil , etc .

**b. Tests unitaires réalisés**

Dans le cadre du projet, plusieurs tests unitaires ont été réalisés afin de garantir le bon fonctionnement des principales interfaces de l'application. Ces tests ont porté sur les éléments suivants

- Vérification de la page d’accueil (landing page) pour s’assurer de son affichage correct.

- Vérification de la navigation vers la page de connexion, afin de garantir un accès fluide.

- Vérification de l’existence de la page d’inscription, assurant ainsi la possibilité pour les nouveaux utilisateurs de créer un compte.

- Vérification de l’accessibilité des routes publiques de connexion, pour s’assurer qu’elles ne nécessitent pas d’authentification préalable.

Ces tests ont été essentiels pour détecter rapidement d’éventuelles régressions et assurer la stabilité de l’interface utilisateur.

* Code

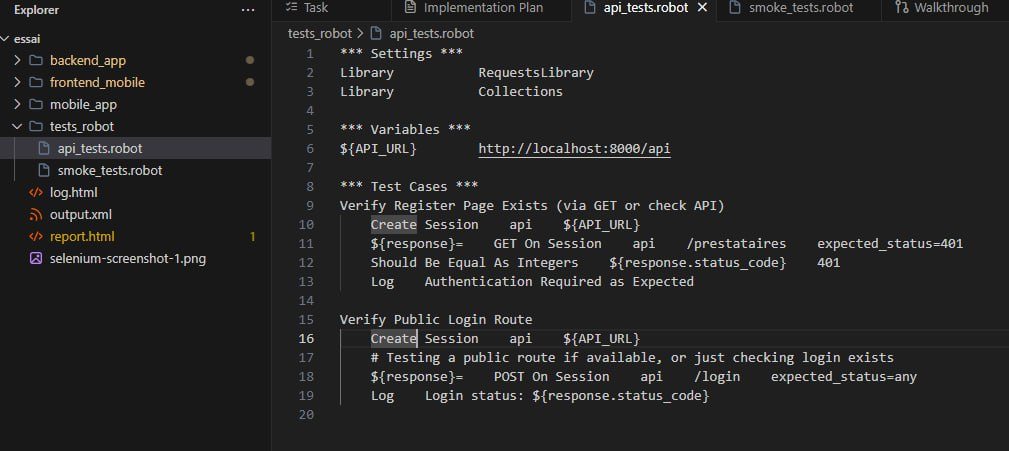


Figure 23 : Test interface utilisateur

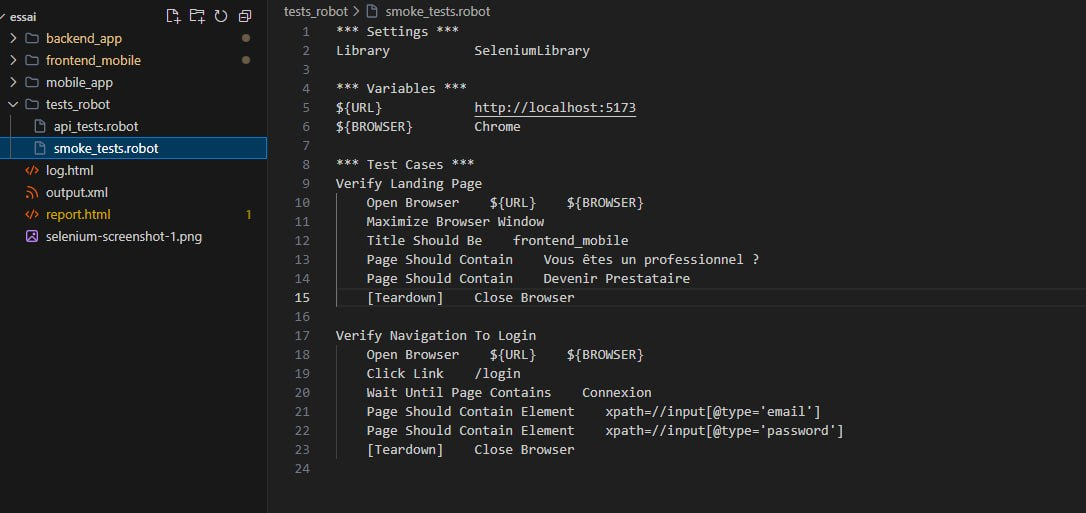


Figure 24 : Test interface utilisateur

* Résultat

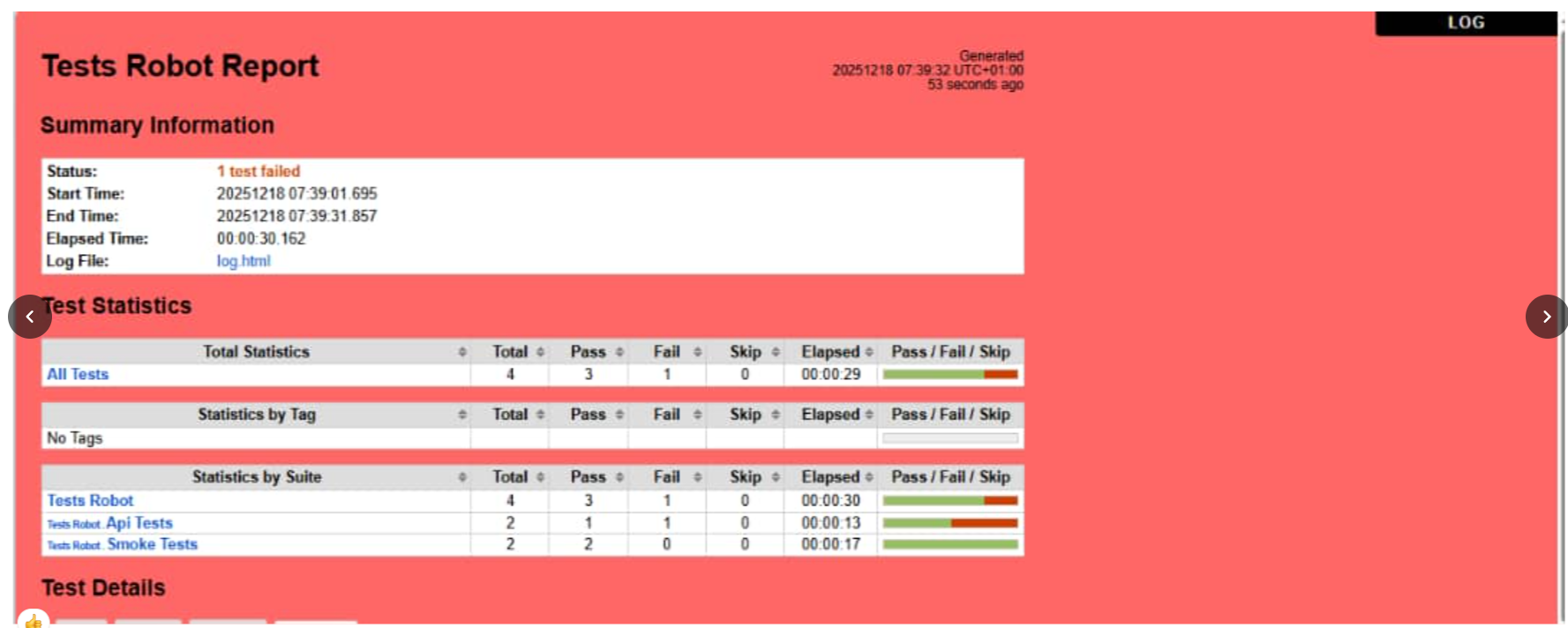


Figure 25 : Résultat

**Conclusion**

Cette phase nous a permis de ressortir les différents blocs de notre application et d’effectuer les différents tests sur nos fonctionnalités.

**GUIDE D’UTILISATION**

**Introduction**

Le guide utilisateur est un document qui permet aux futurs utilisateurs de prendre en main notre solution facilement et de l’utiliser de façon appropriée. Comment notre utilisateur va-t-il s’y prendre, pour répondre à cette question nous allons vous présenter le guide d’installation ainsi que le guide d’utilisation.

1. **Guide d’installation**
2. **Ressources minimales requises**
3. **Matériels requis**

Pour utiliser la plateforme, voici les ressources matérielles minimales requises côté utilisateur:

* **Sur ordinateur (Web) :**

- Un ordinateur ou un laptop avec un processeur standard (Intel i3 ou équivalent minimum)

- Au moins 4 Go de RAM pour un fonctionnement fluide

- Une connexion internet stable et rapide (Wi-Fi, Ethernet ou 4G/5G via modem)

- Un navigateur web moderne et à jour (Chrome, Firefox, Edge, Safari)

* **Sur mobile (application) :**

- Un smartphone ou tablette récent, avec un système d’exploitation compatible (Android 8+ ou iOS 12+)

- Au moins 2 Go de RAM recommandés pour une bonne performance

- Une connexion internet stable via 4G, 5G ou Wi-Fi

Ces ressources assurent une expérience utilisateur fluide, rapide et sans interruptions, surtout pour les échanges de données en temps réel avec la plateforme.

1. **Système d’exploitation**

Notre plateforme n’exige aucune compatibilité pour un système d’exploitation en particulier. Elle est ainsi consultable sur tout système d’exploitation au moyen d’un navigateur.

1. **Navigateur web**

Pour faire usage de toutes les fonctionnalités de la plateforme, nous vous recommandons fortement l’usage d’un navigateur internet à jour.

1. **Guide d’utilisation**

Après avoir rempli toutes les conditions précédentes, avec votre appareil déjà correctement démarré, vous exécutez votre navigateur.

* **Page de connexion utilisateur**

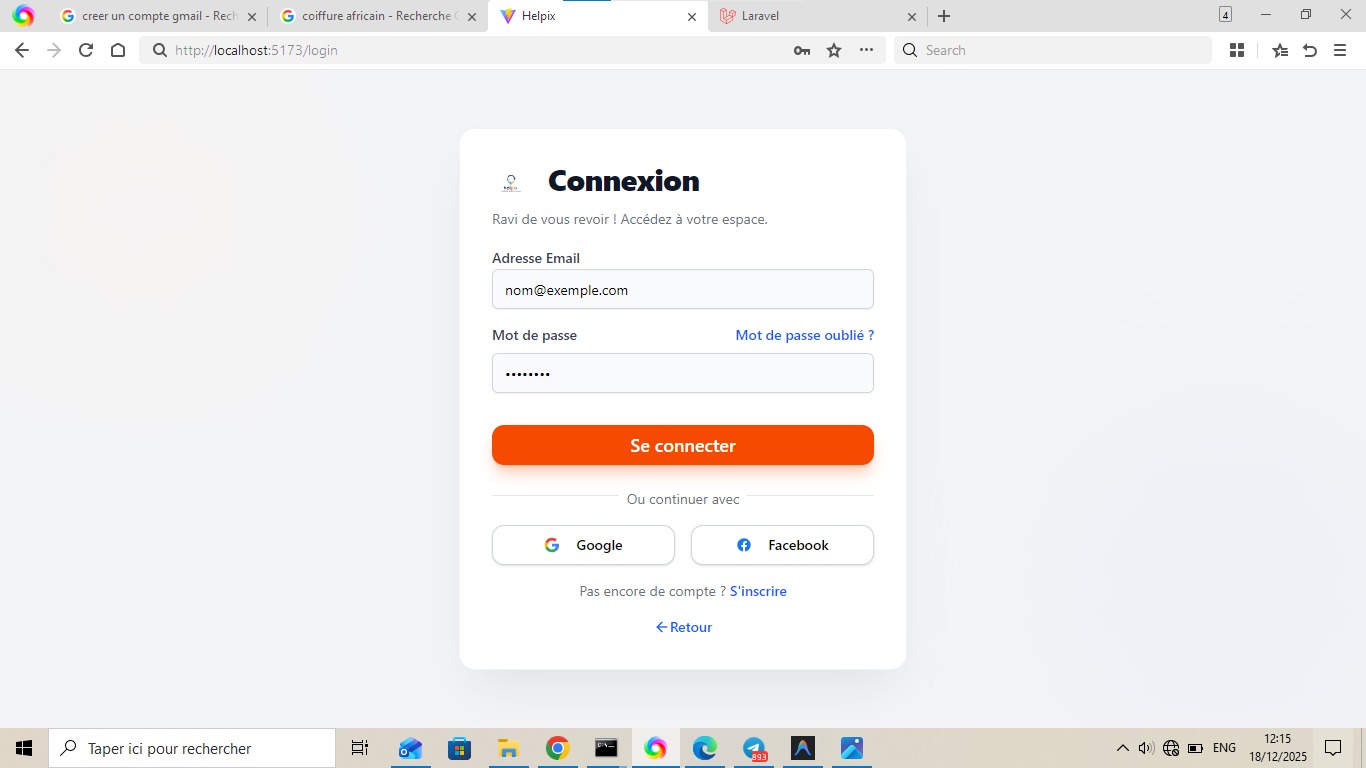
****

Figure 26 : Page connexion utilisateur

* **Page d’accueil Dashboard administrateur**

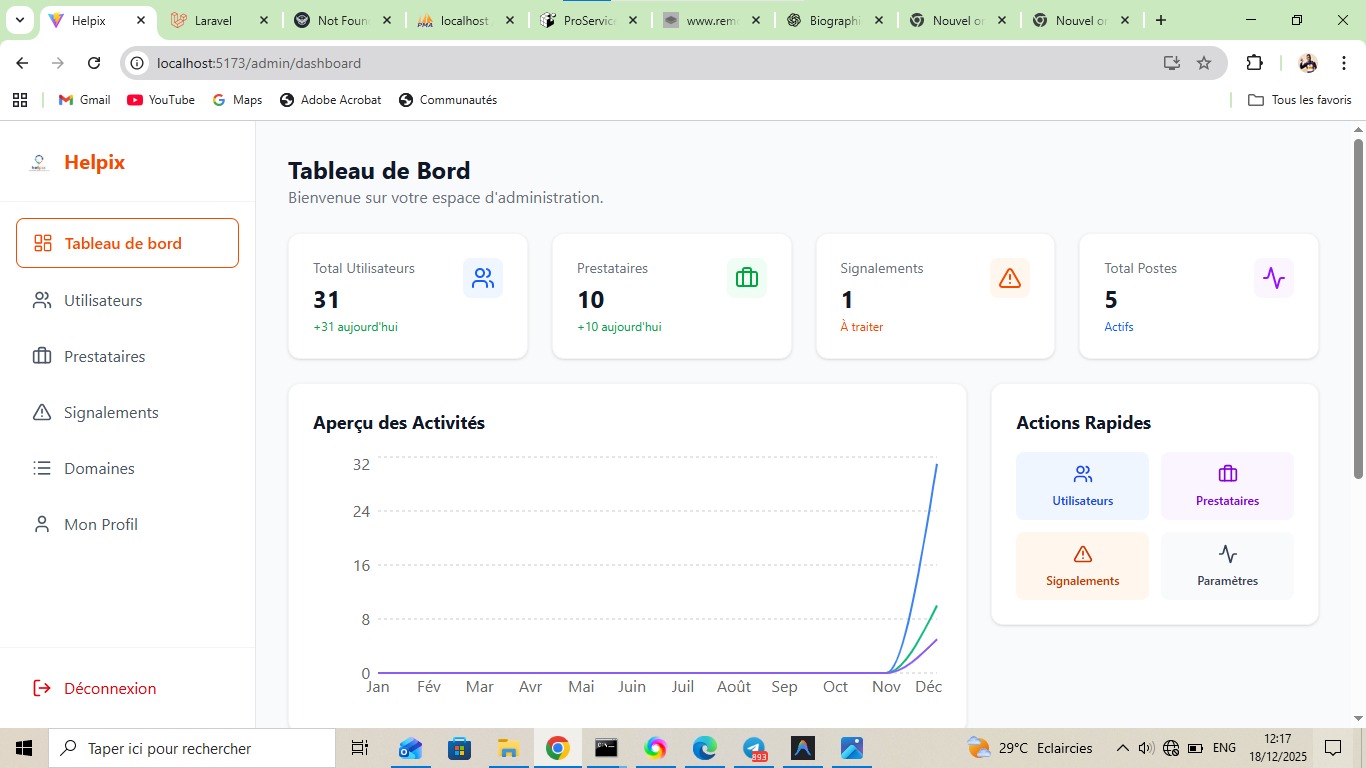


Figure 27 : Page d’accueil Dashboard administrateur

* **Page de recherche d’un prestataire**

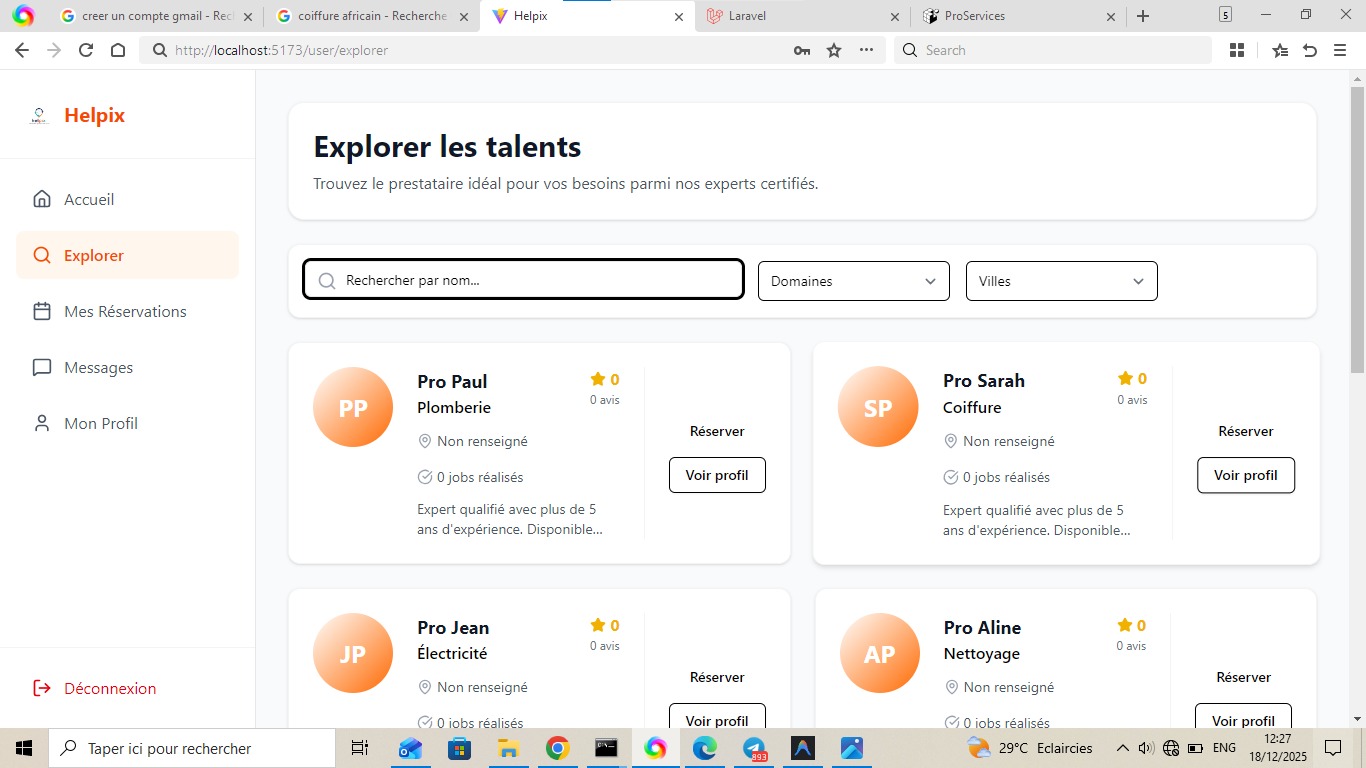


Figure 28 : Page de recherche d’un prestataire

* **Page de gestion des prestataires**

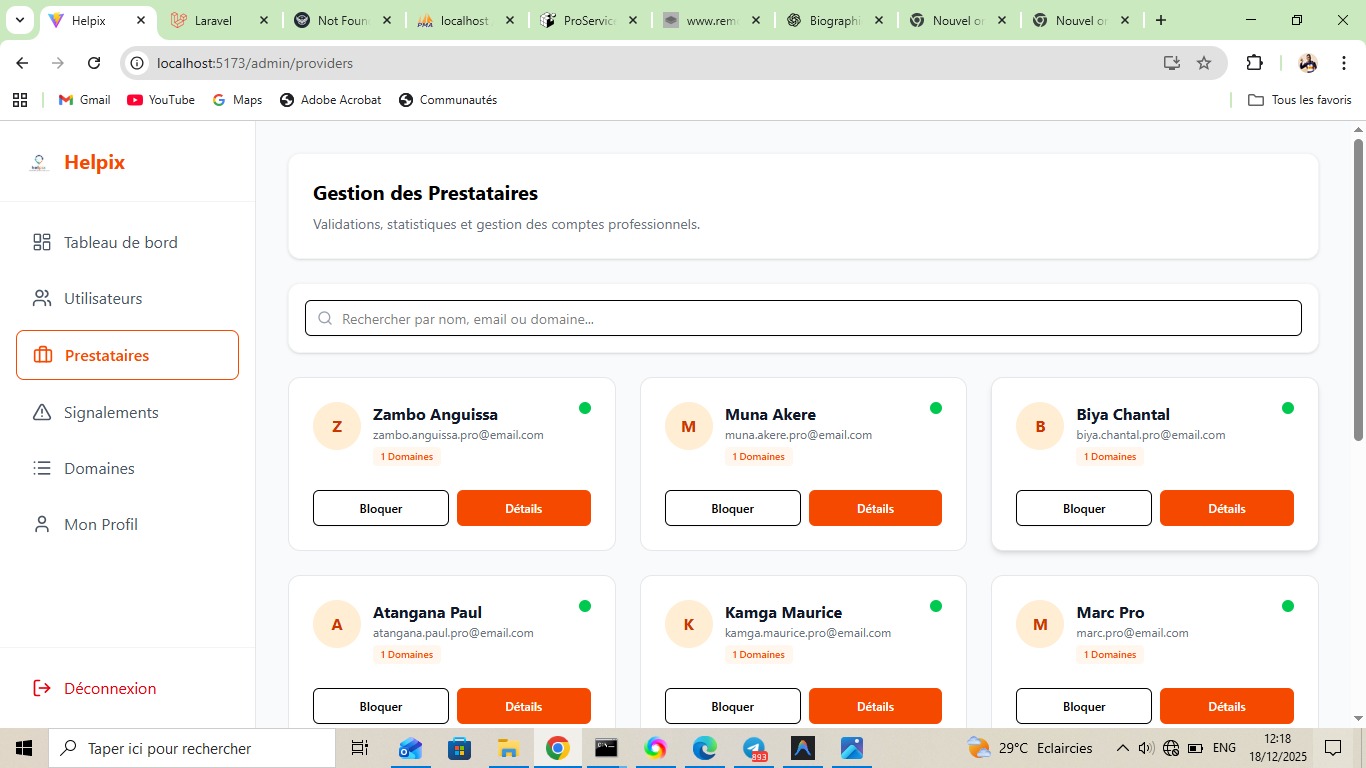
****

Figure 29 : Page de gestion des prestataires

* Page de profil d’un prestataire

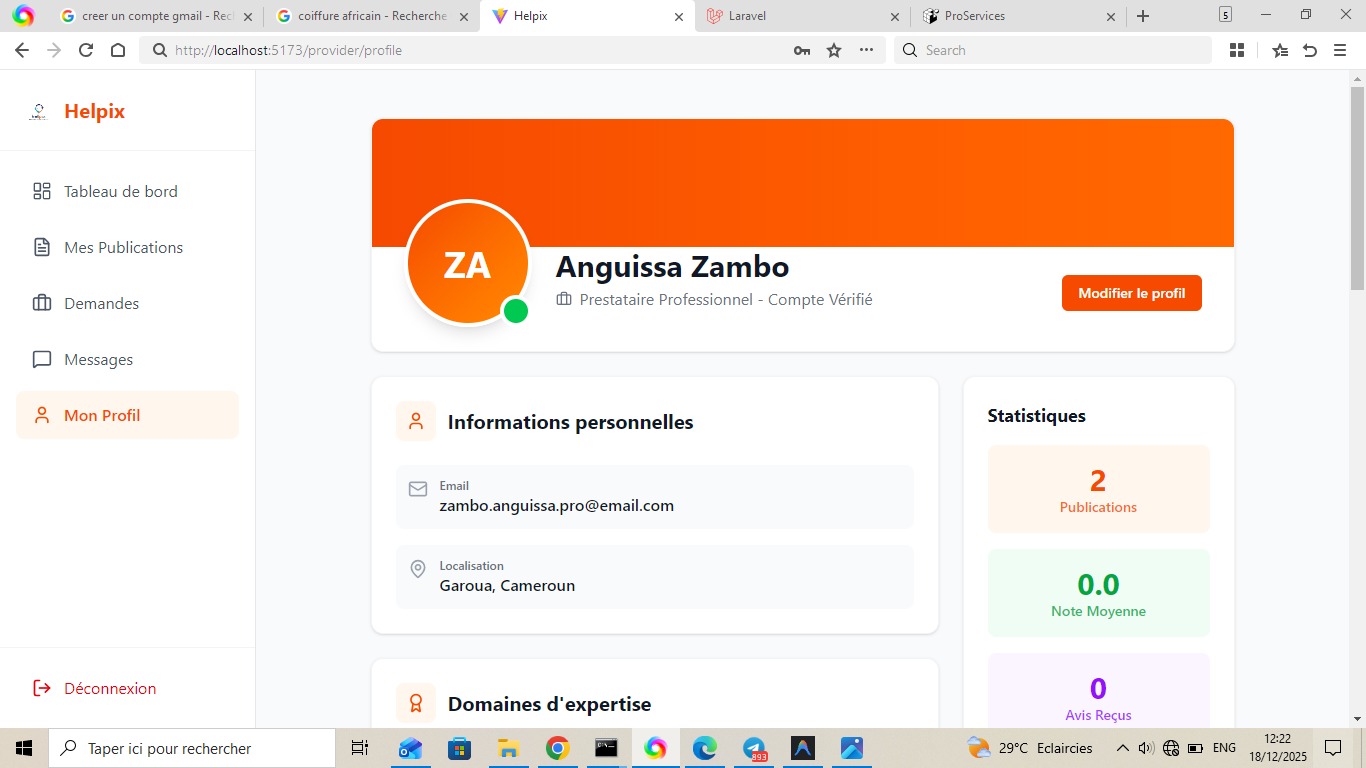


Figure 30 : Page de profil d’un prestataire

* **Page de notation d’un prestataire**

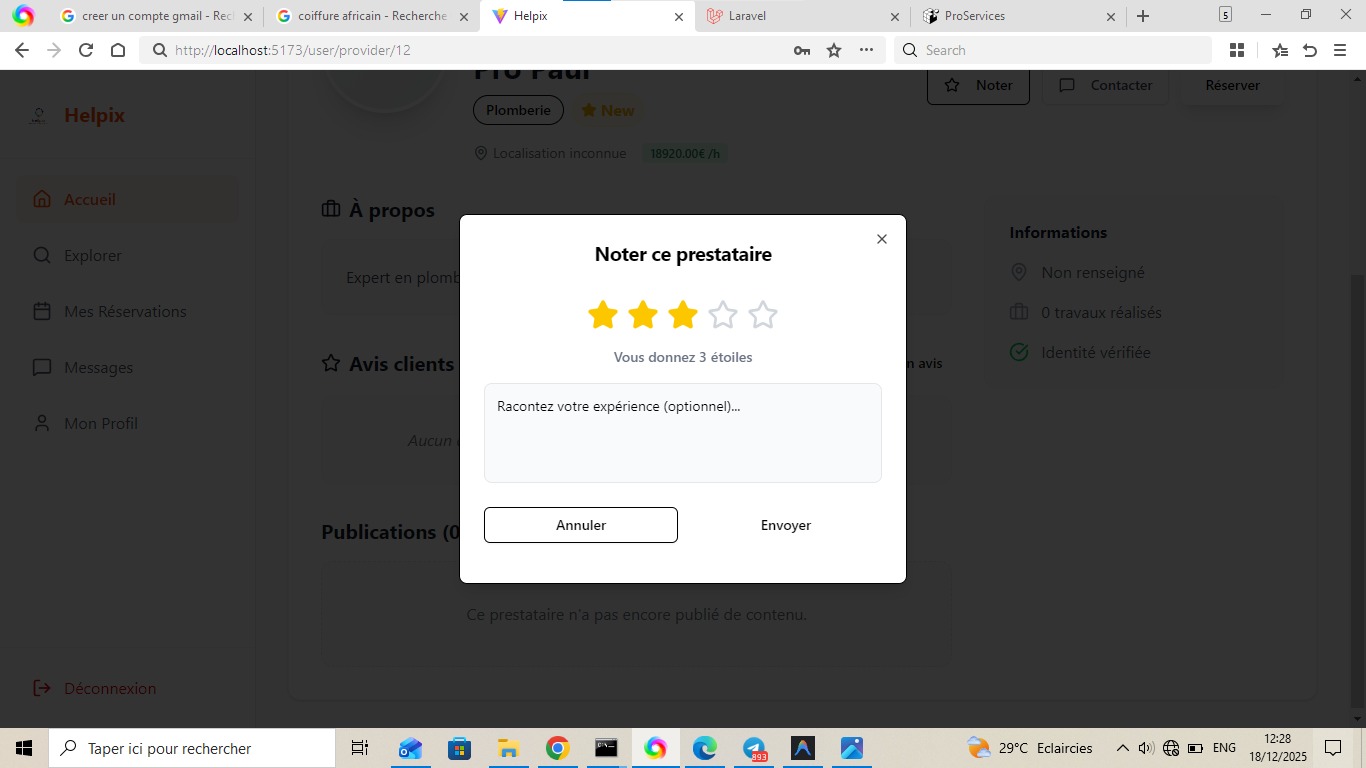
****

Figure 31 : Page de notation d’un prestataire

* **Aperçu de la base de données**

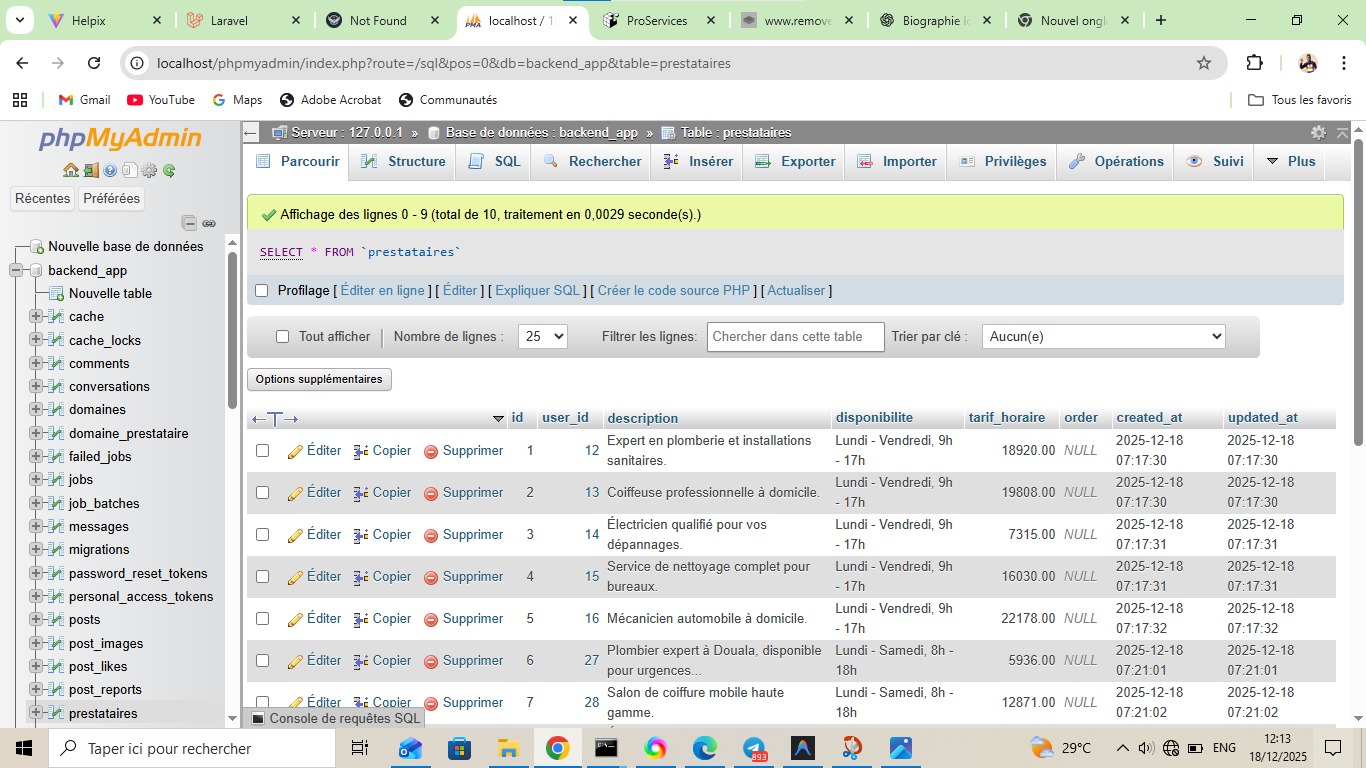


Figure 32 : Aperçu de la base de données

**Conclusion**

Après avoir simulé le fonctionnement de notre plateforme **Helpix**, il en ressort que le guide d’utilisateur est nécessaire pour l’utilisation aisée de l’application. La fin de cette partie nous conduit ainsi à la présentation des différentes difficultés rencontrées, les perspectives et pour finir une conclusion générale de notre projet.

**DIFFICULTES RENCONTREES**

Tout au long de notre travail, nous avons fait face à certaines difficultés telles que :

✓ Les difficultés liées à la connexion internet utilisées tout au long de notre projet

✓ Les coupures intempestives d’électricité qui ont été un grand frein à l’avancement du projet

# CONCLUSION GENERALE

La plateforme que nous avons développé vise à répondre efficacement aux besoins des citoyens en facilitant l’accès à des services essentiels via une plateforme web et mobile. Grâce à une approche méthodologique agile (Scrum), une conception centrée utilisateur et l’usage de technologies modernes telles que les PWA, nous avons pu concevoir une solution fonctionnelle, intuitive et accessible. Ce projet nous a permis de renforcer nos compétences techniques, notre capacité à travailler en équipe et à mener un projet de bout en bout. À l’avenir, des évolutions pourraient être envisagées pour enrichir les services proposés et intégrer davantage d’intelligence dans la gestion des demandes.