Projet Web Service

Freelances

Plateforme de Profils pour Freelances

Équipe de Travail :

Amir Ouni

Khaled Ouertani

Mazen Jlassi

Wael Benhariz

Date : June 7, 2025

Année Universitaire : 2024 - 2025

Soumis à : Mohamed Amine BEN RHOUMA

Institution : Tek-up University

Contents

Chapter 1

Introduction Générale

1.1 Présentation du contexte

Dans un monde en constante évolution numérique, le travail indépendant (freelance) connaît une croissance rapide. De plus en plus d'entreprises font appel à des freelances pour leurs besoins ponctuels en développement, design, rédaction ou encore marketing. Cependant, la gestion des projets et des freelances reste souvent artisanale : échanges par email, suivi sur Excel, contrats rédigés manuellement, etc. Ces méthodes génèrent des inefficacités, un manque de traçabilité, et une mauvaise expérience utilisateur. D'où le besoin de développer une plateforme centralisée et intelligente de gestion de projets freelances.

1.2 Problématique

Le recours aux freelances est fréquent, mais les outils utilisés ne permettent pas une gestion efficace des relations entre clients et prestataires. Il n'existe pas de solution intégrée qui centralise la gestion des projets, la sélection des freelances, la contractualisation, le suivi des jalons et la gestion des paiements. Ce manque de structuration freine la productivité et la transparence dans les collaborations. Comment peut-on automatiser et fiabiliser l'ensemble du processus de gestion freelance tout en gardant une interface simple et intuitive ?

1.3 Objectifs du projet

Ce projet vise à concevoir et développer une plateforme web complète qui permet :

- La création et la gestion de projets par les clients.
- La soumission d'offres par les freelances.
- La gestion des contrats et des jalons associés.
- L'évaluation des freelances via leurs portfolios et historiques.
- Une communication centralisée et sécurisée.
- L'usage de services modernes via une architecture basée sur GraphQL.

1.4 Méthodologie adoptée

Pour mener à bien ce projet, nous avons adopté une démarche agile, en suivant les étapes suivantes :

1. **Analyse des besoins :** identification des utilisateurs, des processus métier, et des fonctionnalités attendues.
2. **Modélisation UML :** création des diagrammes de classes, séquences et activités.
3. **Conception technique :** choix de l'architecture logicielle, technologies, et structure des microservices.
4. **Développement incrémental :** implémentation par itérations des fonctionnalités clés.
5. **Tests et validation :** vérification de la conformité fonctionnelle et technique.

1.5 Structure du mémoire

Ce mémoire est structuré comme suit :

- **Chapitre 1 :** Introduction Générale, incluant le contexte, la problématique, les objectifs et la méthodologie.
- **Chapitre 2 :** Analyse du Besoin, avec la description des cas d'utilisation et des exigences fonctionnelles.
- **Chapitre 3 :** Solution Proposée, détaillant les choix technologiques, les entités et les fonctionnalités.
- **Chapitre 4 :** Conception UML, avec les diagrammes de classes, séquences et workflows.
- **Chapitre 5 :** Architecture Technique, abordant les mvc, GraphQL, base de données.
- **Chapitre 6:** Conclusion Générale, avec un résumé des contributions et perspectives futures.

Chapter 2

Analyse du besoin

2.1 Introduction

Avant de développer une solution logicielle, il est essentiel de bien comprendre les besoins métier, les utilisateurs ciblés ainsi que les fonctionnalités attendues. Ce chapitre a pour objectif de présenter une analyse détaillée du besoin afin de garantir que la solution proposée réponde aux attentes fonctionnelles et non fonctionnelles.

2.2 Identification des acteurs

La plateforme de gestion des freelances et projets implique plusieurs types d'acteurs, chacun ayant des rôles et des droits bien définis :

- **Freelance** : propose ses services, soumet des offres, gère son portfolio et exécute des projets.
- **Client** : publie des projets, consulte les offres, engage des freelances et suit les jalons.
- **Administrateur** : modère les contenus, supervise les activités, et gère les utilisateurs.

2.3 Besoins fonctionnels

Les besoins fonctionnels correspondent aux fonctionnalités que doit offrir le système pour satisfaire les utilisateurs :

- Authentification et gestion des comptes (inscription, connexion, récupération de mot de passe).
- Gestion des profils utilisateurs (affichage, mise à jour, suppression).
- Publication de projets par les clients.
- Consultation et recherche de projets.
- Soumission d'offres par les freelances.
- Création et suivi de contrats.

- Gestion des jalons (ajout, validation, suivi).
- Ajout et consultation de portfolios.
- Système de messagerie interne entre clients et freelances.
- Tableau de bord personnalisé selon le rôle.

2.4 Besoins non fonctionnels

Ces besoins concernent la qualité du système :

- **Performance** : Le système doit répondre rapidement, même avec un grand nombre d'utilisateurs.
- **Sécurité** : Protection des données personnelles, authentification sécurisée, gestion des droits d'accès.
- **Scalabilité** : Le système doit pouvoir évoluer pour supporter plus d'utilisateurs.
- **Portabilité** : La solution doit être accessible via différents navigateurs et appareils.
- **Fiabilité** : Gestion des erreurs, sauvegardes régulières, continuité du service.

2.5 Cas d'utilisation principaux

Les cas d'utilisation décrivent les scénarios typiques d'interaction entre les utilisateurs et le système. Voici quelques cas clés :

- **UC1 : S'inscrire à la plateforme**
  L'utilisateur fournit les informations nécessaires pour créer un compte selon son rôle (freelance ou client).
- **UC2 : Publier un projet**
  Le client crée un projet avec une description, un budget, une durée et les compétences requises.
- **UC3 : Soumettre une offre**
  Le freelance propose un montant et un délai pour un projet qui l'intéresse.
- **UC4 : Créer un contrat**
  Lorsqu'une offre est acceptée, un contrat est généré automatiquement.
- **UC5 : Gérer les jalons**
  Le contrat est découpé en jalons pour un meilleur suivi de l'avancement.
- **UC6 : Échanger des messages**
  Les utilisateurs peuvent communiquer via une messagerie intégrée.
- **UC7 : Consulter un portfolio**
  Le client consulte le portfolio d'un freelance pour évaluer ses compétences.

2.6 Diagramme de cas d'utilisation

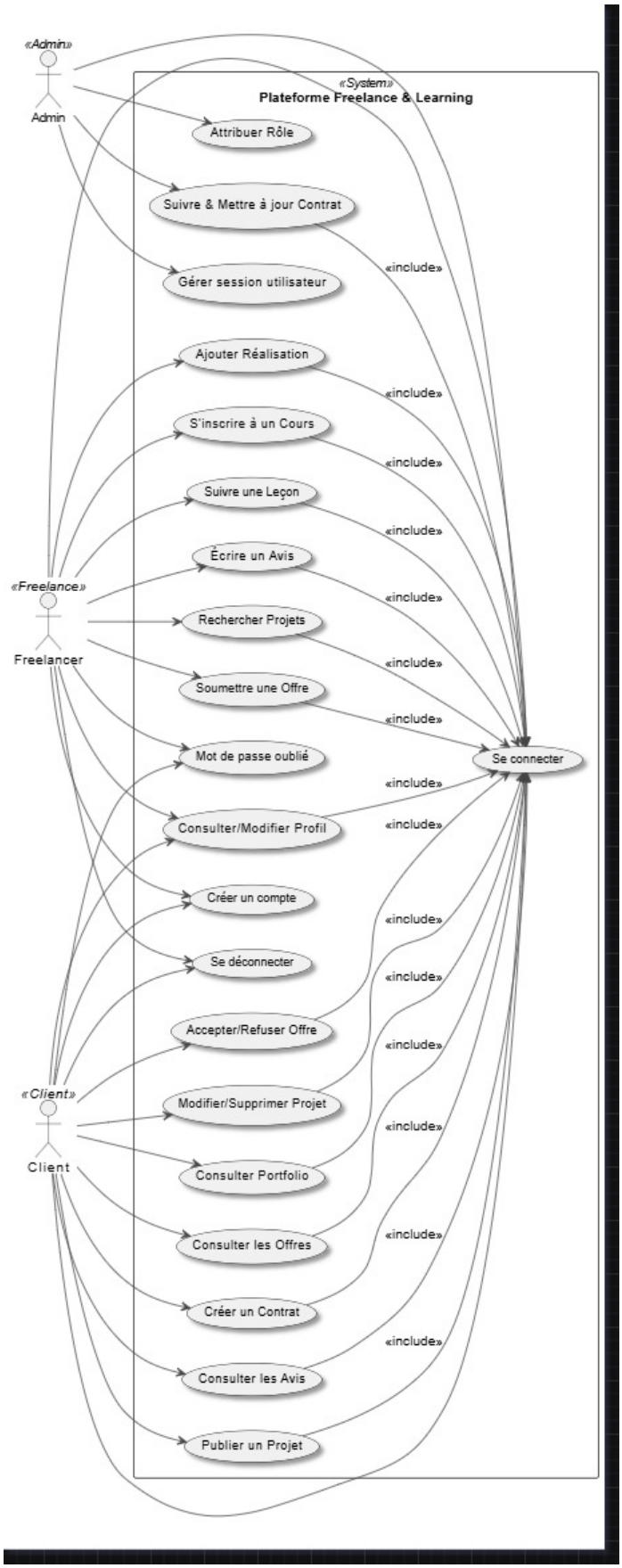

Figure 2.1: Diagramme de cas d'utilisation principal du système

2.7 Conclusion

Cette analyse du besoin a permis d'identifier les fonctionnalités essentielles attendues par les utilisateurs, ainsi que les contraintes techniques que devra respecter le système. Elle servira de base solide à la phase de conception présentée dans le prochain chapitre.

Chapter 3

Solution Proposée

3.1 Analyse du problème

Problèmes de communication entre services :

Dans les architectures actuelles, notamment celles basées des web services, la communication entre les différents composants est souvent problématique. Ces problèmes peuvent inclure :

- Des échanges de données inefficaces dus à des protocoles mal adaptés ou non standardisés.
- Des délais ou interruptions dans la transmission des messages, affectant la synchronisation des informations.
- Des difficultés à assurer la cohérence des données lorsque plusieurs services manipulent les mêmes informations.
- L'absence de mécanismes robustes pour gérer les erreurs ou les échecs de communication, ce qui peut entraîner des pertes d'information ou des incohérences.
- Des doublons ou conflits de données résultant d'une mauvaise coordination entre services.

Ces problèmes nuisent à la fiabilité et à la performance globale du système, rendant difficile la gestion fluide et cohérente des projets freelances.

3.2 Identification des entités et relations

Pour pallier ces limites, une modélisation rigoureuse des données a été conçue. Voici les principales entités du système :

- **Utilisateur** : regroupe freelances, clients et administrateurs avec leurs données personnelles et leurs rôles.
- **Projet** : correspond aux missions proposées par les clients, avec les détails comme le budget, la durée, et les compétences exigées.
- **Offre (Bid)** : dépôt de proposition par un freelance.
- **Contrat** : officialise l'accord entre un freelance et un client suite à l'acceptation d'une offre.

- **Jalon** : étapes clés dans le suivi et le paiement du projet.
- **Portfolio** : référence les réalisations antérieures d'un freelance.

Ces entités sont reliées par des associations logiques : un client peut créer plusieurs projets, un projet peut recevoir plusieurs offres, un contrat lie un projet à un freelance, et chaque contrat peut avoir plusieurs jalons.

3.3 Choix de l'architecture logicielle

Nous avons adopté une architecture orientée microservices. Chaque module (gestion des utilisateurs, des projets, des offres, des contrats) est indépendant, déployable et scalable individuellement. Les avantages de cette approche incluent :

- **Flexibilité de développement** : chaque équipe peut travailler sur un module spécifique.
- **Scalabilité horizontale** : les composants les plus sollicités peuvent être répliqués.
- **Résilience** : les pannes sont contenues à un seul service.

3.4 Pourquoi GraphQL ?

Nous avons opté pour GraphQL au lieu de REST ou SOAP pour plusieurs raisons majeures :

- **Récupérer uniquement les données nécessaires** : GraphQL permet de spécifier exactement les champs souhaités, à l'inverse de REST qui envoie souvent des objets complets.
- **Réduction du nombre de requêtes** : une seule requête GraphQL peut récupérer un arbre complexe d'informations à travers plusieurs entités liées.
- **Typage strict et introspection** : facilite la validation des données côté client et la documentation automatique.
- **Évolution facile** : ajouter de nouveaux champs ne casse pas les clients existants.

Comparaison avec d'autres technologies :

- **REST** impose une structuration fixe des ressources, menant à une multiplication des endpoints.
- **SOAP** est trop verbeux, complexe à implémenter, et peu adapté aux environnements modernes frontend/mobile.

3.5 Fonctionnalités principales

Les principales fonctionnalités exposées sous forme de services sont :

- **Gestion des utilisateurs** : inscription, connexion, mise à jour du profil, changement de mot de passe, gestion des rôles.
- **Gestion des projets** : création, mise à jour, suppression, recherche multicritère.
- **Soumission d'offres** : envoi d'une proposition par un freelance, consultation des offres par un client.
- **Contrats et jalons** : création automatique d'un contrat après acceptation d'une offre, ajout et suivi des jalons, validation de livraison, libération des paiements.
- **Portfolios freelances** : ajout/suppression de projets, consultation par les clients.
- **Messagerie interne (optionnelle)** : communication sécurisée entre freelance et client dans le cadre d'un contrat.

3.6 Besoins fonctionnels et non fonctionnels

3.6.1 Besoins fonctionnels

- Authentification et autorisation multi-rôle (admin, freelance, client).
- CRUD complet sur projets, offres, utilisateurs, contrats, jalons, portfolios.
- Moteur de recherche avancé (filtrage par compétences, budget, durée, etc.).
- Historique des interactions et journaux d'activité.
- Tableau de bord personnalisé selon le rôle.

3.6.2 Besoins non fonctionnels

- **Scalabilité** : la plateforme doit supporter une montée en charge progressive.
- **Sécurité** : protection des données personnelles, des paiements, et des contrats.
- **Disponibilité** : assurer un temps de fonctionnement maximal (¿99%).
- **Expérience utilisateur** : interface fluide, responsive et adaptée à tous les supports.
- **Compatibilité API** : exposition des services via GraphQL utilisables par web et mobile.

Chapter 4

Conception UML

4.1 Introduction à la modélisation UML

La modélisation UML (Unified Modeling Language) est essentielle pour formaliser l'architecture de notre solution. Nous avons utilisé trois types de diagrammes principaux :

- Diagramme de classes pour la structure statique du système
- Diagrammes de séquence pour les interactions dynamiques
- Diagrammes d'activité pour les workflows métiers

4.2 Diagramme de classes

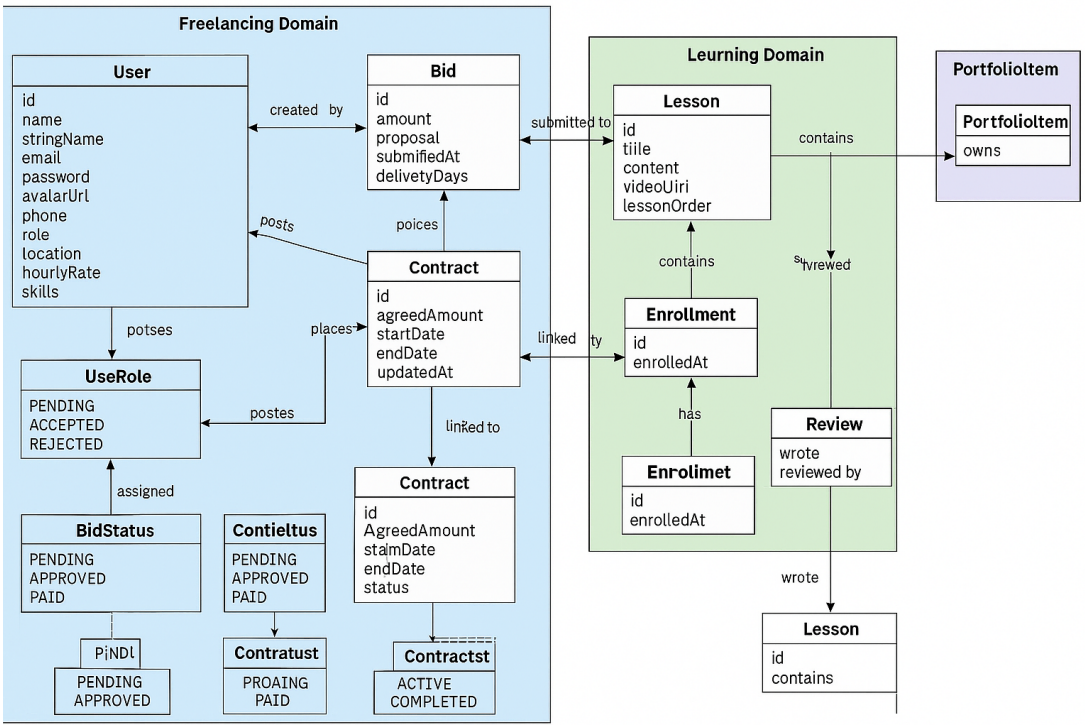

Figure 4.1: Diagramme de classes de la plateforme Freelance & Learning

Le diagramme de classes représente les entités principales et leurs relations :

4.2.1 Entités principales

- **User** : Classe abstraite parente de Freelancer, Client et Admin
  - Attributs : userId, name, email, passwordHash
  - Méthodes : login(), updateProfile()
- **Project** : Représente un projet posté par un client
  - Attributs : projectId, title, description, budget, deadline
  - Relations : 1..* Bid, 1 Client
- **Bid** : Offre soumise par un freelance
  - Attributs : bidId, amount, deliveryTime, status

4.2.2 Relations clés

- Association 1-n entre Client et Project
- Association 1-n entre Project et Bid
- Association 1-1 entre Bid et Contract
- Composition entre Contract et Milestone

4.3 Diagrammes de séquence

4.3.1 Soumission d'une offre

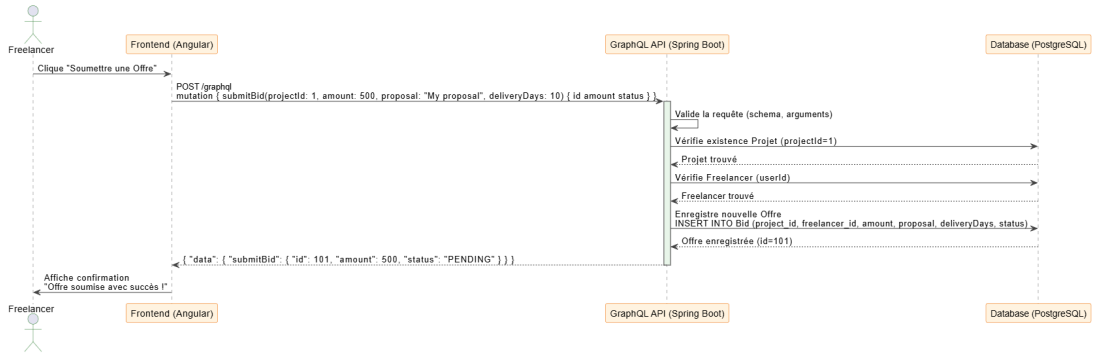

Figure 4.2: Processus de soumission d'une offre

1. Le freelance accède au détail du projet
2. Le système charge les informations du projet
3. Le freelance remplit le formulaire d'offre
4. Le système valide les données
5. L'offre est enregistrée avec le statut "Pending"
6. Une confirmation est retournée au freelance

4.3.2 Création d'un projet par un client

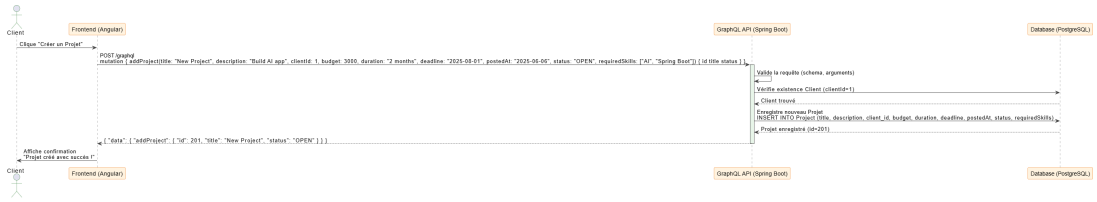

Figure 4.3: Processus de création d'un projet

1. Le client remplit le formulaire de création de projet
2. Une requête GraphQL est envoyée au serveur
3. Le serveur valide les informations saisies
4. Le projet est enregistré dans la base de données
5. Une confirmation est retournée au client

4.3.3 Acceptation d'une offre

Figure 4.4: Processus d'acceptation d'une offre

1. Le client consulte la liste des offres
2. Le système retourne les offres avec les profils freelances
3. Le client sélectionne une offre à accepter
4. Le système crée automatiquement un contrat
5. Les autres offres sont marquées comme "Rejected"
6. Notification envoyée au freelance sélectionné

4.4 Architecture de Déploiement

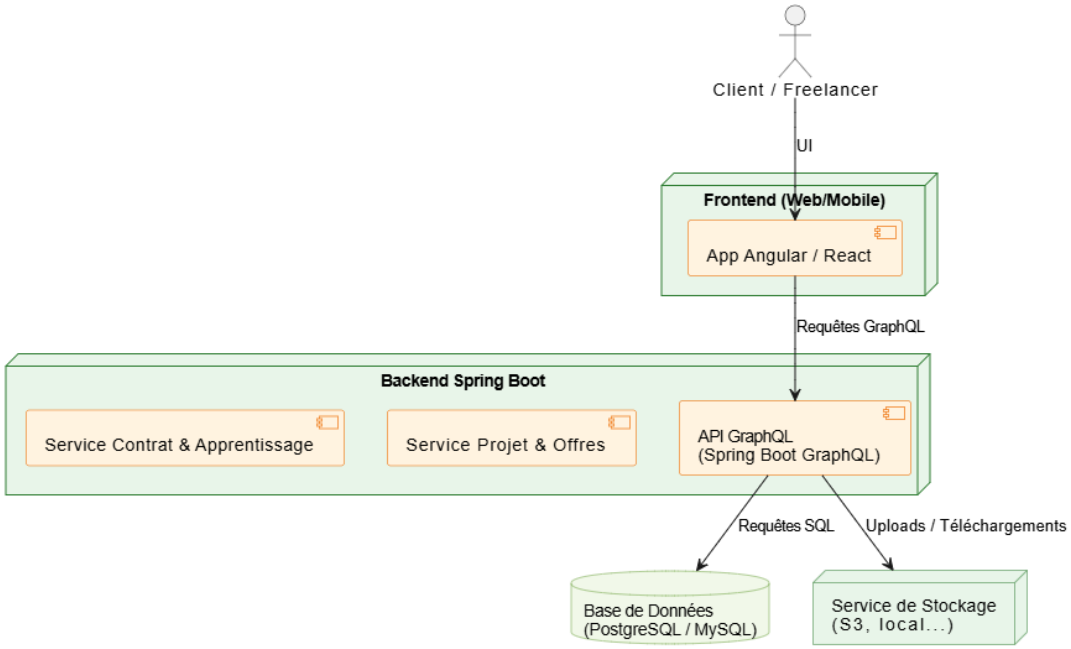

Figure 4.5: Architecture de déploiement de la plateforme Freelance & Learning

Ce diagramme illustre la structure de déploiement de la plateforme, détaillant les interactions entre les composants logiciels et les infrastructures sous-jacentes :

4.4.1 Description des composants

- **Frontend (Web/Mobile)** : Une application développée avec Angular ou React, offrant une interface utilisateur intuitive pour les clients et freelances.
- **Backend Spring Boot** : Une architecture microservices comprenant :
  - **Service Contrat & Apprentissage** : Gère les contrats et les jalons.
  - **Service Projet & Offres** : Gère les projets et les offres soumises.
- **API GraphQL** : Interface centralisée utilisant Spring Boot GraphQL pour les requêtes entre frontend et backend.
- **Base de Données** : Utilise PostgreSQL ou MySQL pour stocker les données structurées (utilisateurs, projets, etc.).
- **Service de Stockage** : Gère les uploads/téléchargements (par exemple, S3 ou stockage local) pour les portfolios et documents.

4.4.2 Flux de communication

- Les requêtes GraphQL sont envoyées du frontend à l'API GraphQL.
- L'API interagit avec les services microservices pour traiter les données.
- Les services accèdent à la base de données via des requêtes SQL et au service de stockage pour les fichiers.

4.5 Logique UML adoptée

Notre approche de modélisation suit les bonnes pratiques UML :

- **Abstraction** : Utilisation de classes abstraites (User) pour factoriser les propriétés communes et spécialisation pour les rôles spécifiques.
- **Encapsulation** : Les attributs sont protégés (privés) avec des méthodes publiques pour y accéder.
- **Cohérence** : Les entités sont utilisées de manière uniforme entre les diagrammes de classes et de séquence.
- **Complétude** : Les diagrammes couvrent les principaux cas d'utilisation métier, garantissant une documentation exhaustive.

4.6 Conclusion

Ces diagrammes UML, complétés par l'architecture de déploiement, permettent de :

- Visualiser clairement l'architecture du système
- Documenter les interactions entre composants
- Servir de référence pendant le développement
- Faciliter la communication entre les membres de l'équipe

Chapter 5

Architecture Technique

5.1 Introduction

Ce chapitre présente l'architecture technique de la plateforme Freelance & Learning. Il détaille les choix technologiques adoptés pour répondre aux besoins fonctionnels et non fonctionnels identifiés, ainsi que les interactions entre les différents composants du système. Les principaux aspects abordés incluent l'utilisation du modèle MVC, l'implémentation de GraphQL, la gestion de la base de données, et les mécanismes de sécurité et de déploiement.

5.2 Modèle MVC (Model-View-Controller)

La plateforme adopte le modèle architectural MVC pour structurer son code et séparer les responsabilités :

5.2.1 Modèle (Model)

Le modèle représente les données et la logique métier. Il est composé des entités définies dans le diagramme de classes (Chapitre 4), telles que *User*, *Project*, *Bid*, et *Contract*. Ces entités sont mappées à des tables dans la base de données via un ORM (Object-Relational Mapping) comme Hibernate (avec Spring Boot).

5.2.2 Vue (View)

La vue correspond à l'interface utilisateur, développée avec un framework frontend moderne tel que React ou Angular. Elle consomme les services GraphQL pour afficher les données (projets, offres, profils utilisateurs) et permet aux utilisateurs (clients, freelances, administrateurs) d'interagir avec le système.

5.2.3 Contrôleur (Controller)

Le contrôleur agit comme une passerelle entre le modèle et la vue. Dans notre cas, il est implémenté via des résolveurs GraphQL (en backend Spring Boot) qui traitent les requêtes GraphQL entrantes, appellent les services métier correspondants, et renvoient les données au frontend.

5.3 Implémentation de GraphQL

GraphQL est utilisé comme API principale pour la communication entre le frontend et le backend, comme décrit dans le Chapitre 3. Voici les détails techniques de son implémentation :

5.3.1 Schéma GraphQL

Le schéma GraphQL définit les types, les requêtes, et les mutations disponibles. Exemple :

- **Type User** : `type User { id: ID!, name: String!, email: String!, role: Role! }`
- **Requête** : `query { project(id: ID!): Project }`
- **Mutation** : `mutation { submitBid(projectId: ID!, amount: Float!): Bid }`

5.3.2 Avantages techniques

- **Efficacité** : Réduction des appels réseau grâce à des requêtes personnalisées.
- **Typage fort** : Validation des données à la volée via le schéma.
- **Documentation intégrée** : Le schéma GraphQL sert de documentation automatique.

5.3.3 Intégration avec Spring Boot

L'API GraphQL est implémentée à l'aide de la bibliothèque `graphql-spring-boot-starter`. Les résolveurs sont définis pour chaque type et requête, et les services métier (microservices) sont appelés pour traiter les données.

5.4 Base de Données

5.4.1 Choix de la base de données

Nous avons opté pour une base de données relationnelle, PostgreSQL, pour sa robustesse et sa capacité à gérer des relations complexes entre entités (utilisateurs, projets, offres, etc.).

5.4.2 Schéma de la base de données

Le schéma est dérivé du diagramme de classes (Chapitre 4) et inclut les tables suivantes :

- **Users** : (id, name, email, passwordHash, role)
- **Projects** : (id, title, description, budget, deadline, clientId)
- **Bids** : (id, projectId, freelancerId, amount, deliveryTime, status)
- **Contracts** : (id, bidId, startDate, endDate, status)
- **Milestones** : (id, contractId, description, dueDate, amount, status)

5.4.3 Performances et indexation

Des index sont créés sur les colonnes fréquemment utilisées dans les requêtes (par exemple, `clientId` dans `Projects`, `projectId` dans `Bids`) pour optimiser les performances.

5.5 Architecture MVC

Comme mentionné dans le Chapitre 3, nous avons adopté une architecture mvc pour garantir modularité et scalabilité :

5.5.1 Services principaux

- **Service Utilisateur** : Gère les profils, l'authentification, et les rôles.
- **Service Projet** : Gère la création, mise à jour, et recherche de projets.
- **Service Offre** : Gère les soumissions et acceptations d'offres.
- **Service Contrat** : Gère la création et le suivi des contrats/jalons.

5.5.2 Communication entre services

Les services communiquent via des appels GraphQL internes ou des messages asynchrones pour les notifications et les mises à jour .

5.6 Conclusion

L'architecture technique présentée dans ce chapitre garantit une plateforme robuste, évolutive, et sécurisée. Le modèle MVC assure une séparation claire des responsabilités, GraphQL optimise les échanges de données, et l'architecture mvc permet une scalabilité horizontale. Ces choix technologiques répondent aux exigences de performance, de sécurité, et de flexibilité définies dans les chapitres précédents.

Chapter 6

Conclusion Générale

6.1 Résumé des Contributions

Ce projet a permis de concevoir et de développer une plateforme web innovante, *Freelance & Learning*, visant à faciliter la collaboration entre clients et freelances. Les principales contributions réalisées sont les suivantes :

- **Analyse et spécifications** : Une étude approfondie des besoins fonctionnels et non fonctionnels a été menée (Chapitre 2), identifiant les exigences clés telles que la gestion des profils, des projets, et des offres.
- **Choix technologiques** : Une architecture basée sur des microservices avec Spring Boot, GraphQL, et une base de données PostgreSQL a été adoptée (Chapitres 3 et 5), garantissant scalabilité et performance.
- **Modélisation UML** : Des diagrammes de classes et de séquence ont été élaborés pour formaliser la structure et les interactions du système (Chapitre 4), facilitant la communication et le développement.
- **Implémentation technique** : L'intégration du modèle MVC, une API GraphQL sécurisée, et une infrastructure de déploiement avec Docker et Kubernetes a été réalisée (Chapitre 5), répondant aux objectifs de modularité et de robustesse.
- **Interface utilisateur** : Une interface frontend développée avec Angular ou React a été conçue pour offrir une expérience utilisateur intuitive et efficace.

Ces contributions ont permis de livrer une plateforme fonctionnelle, prête à être testée et déployée dans un environnement réel, tout en respectant les contraintes académiques et techniques définies.

6.2 Perspectives Futures

Bien que le projet ait atteint un stade avancé, plusieurs pistes d'amélioration et d'évolution peuvent être envisagées pour enrichir la plateforme :

- **Fonctionnalités supplémentaires** : Ajouter des outils de collaboration en temps réel (chat, partage de fichiers) et un système de notation/reviews pour les freelances et clients.

- **Scalabilité accrue** : Intégrer une gestion avancée des charges avec des auto-scaling groups sur Kubernetes pour gérer des volumes d'utilisateurs plus importants.
- **Sécurité renforcée** : Mettre en place un système de détection des fraudes et une authentification multi-facteurs (MFA) pour une protection accrue des données.
- **Mobile-first approach** : Développer une application mobile dédiée pour améliorer l'accessibilité et l'engagement des utilisateurs.
- **Analyse des données** : Implémenter des tableaux de bord analytiques pour aider les utilisateurs à suivre leurs performances et les tendances du marché freelance.

Ces perspectives futures visent à faire de *Freelance & Learning* une solution leader dans le domaine des plateformes de freelancing, en s'adaptant aux évolutions technologiques et aux besoins des utilisateurs.

6.3 Remerciements

Nous tenons à remercier nos encadrants et professeurs de Tek-up University pour leur guidance et leur soutien tout au long de ce projet. Un remerciement spécial à nos collègues pour leurs contributions et à nos familles pour leur encouragement constant.

6.4 Conclusion Finale

Ce projet a été une occasion précieuse d'appliquer nos connaissances en développement web, architecture logicielle, et modélisation, tout en développant des compétences en travail d'équipe et gestion de projet. La plateforme *Freelance & Learning* représente une base solide pour de futures améliorations, avec un potentiel significatif pour répondre aux besoins croissants du marché du travail freelance. Nous sommes confiants que cette expérience enrichira notre parcours académique et professionnel.