Rapport de projet

Seasons Force

Frezier Romain
Italiano Lorenzo

IG5
2023-2024

Sommaire

Table des figures

Introduction

Dans le cadre de notre cinquième année d'Informatique et Gestion à Polytech Montpellier, nous avons été confrontés à un défi stimulant : le développement d'une application complexe destinée aux recruteurs de travailleurs saisonniers, incluant également une interface pour un administrateur. Ce projet, s'étalant sur une période de deux mois, et par groupe de deux, était non seulement une opportunité d'approfondir nos connaissances théoriques, mais aussi une chance de nous immerger dans une expérience pratique intensément professionnalisante.

L'objectif principal était de maîtriser l'architecture en microservices, un paradigme de développement logiciel de plus en plus prisé dans l'industrie pour sa modularité et sa flexibilité. En parallèle, une application mobile avec React Native doit être développée, ce qui permettra de découvrir cette technologie qui est très utilisée dans le milieu du développement mobile.

Les défis étaient multiples : non seulement nous devions concevoir et implémenter une application fonctionnelle répondant à un cahier des charges complexe, mais nous devions également nous assurer que chaque microservice fonctionne harmonieusement au sein de l'architecture globale. Cette expérience nous a plongés dans un environnement simulant les conditions réelles d'un projet informatique en entreprise, mettant à l'épreuve notre capacité à gérer simultanément les aspects techniques, logistiques et collaboratifs d'un projet de développement logiciel.

Ce rapport présente en détail notre parcours à travers ce projet, mettant en lumière les technologies utilisées, les méthodologies adoptées, les défis rencontrés et les apprentissages acquis. Il témoigne de notre évolution en tant que développeur et de notre capacité à relever des défis complexes, tout en offrant un aperçu de ce que peut être la gestion d'un projet informatique dans un contexte académique avancé.

1. Conception

1.1. Architecture

Nous avons produit ce diagramme d'architecture afin de se rendre compte des différentes parties prenantes de notre application. Il est amené à évoluer en fonction de l'ajout de nouveaux micro services et lorsque nous ajouterons Apache Kafka.

Voici ci-après le schéma d'architecture complet (vous pouvez également le trouver en plusieurs parties en Annexes pour plus de lisibilité).

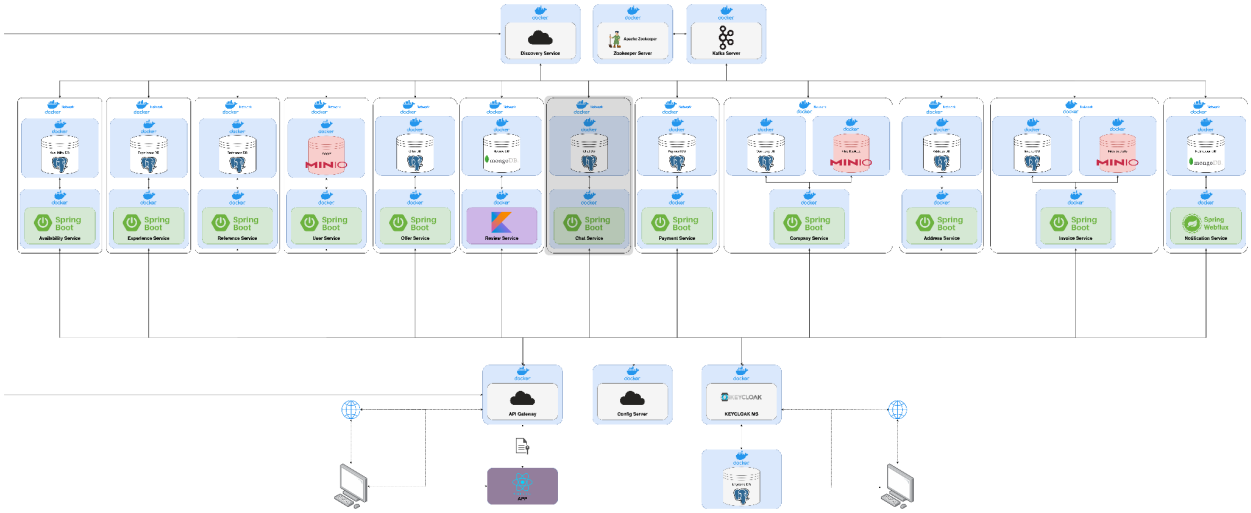


Figure n°1 - Schéma de l'architecture micro-services

Nous avons également mis en place un début de CI / CD grâce à Github Actions. L'objectif de cette CI / CD est de lancer les tests du micro service concerné lors d'un merge entre develop et main afin de refuser le merge si les tests ne passent pas. S'ils passent, on peut alors procéder au merge.

Dans un contexte microservice, il est très important d'ordonnancer les microservices. Ainsi nous avons réalisé un schéma (Figure 3) montrant quel service dépend de qui et comment. Par exemple, on remarque que le Config-Server doit être lancé en avance car plusieurs services dépendent de lui (avec une action de fetch-config) et ce service ne dépend de personne.

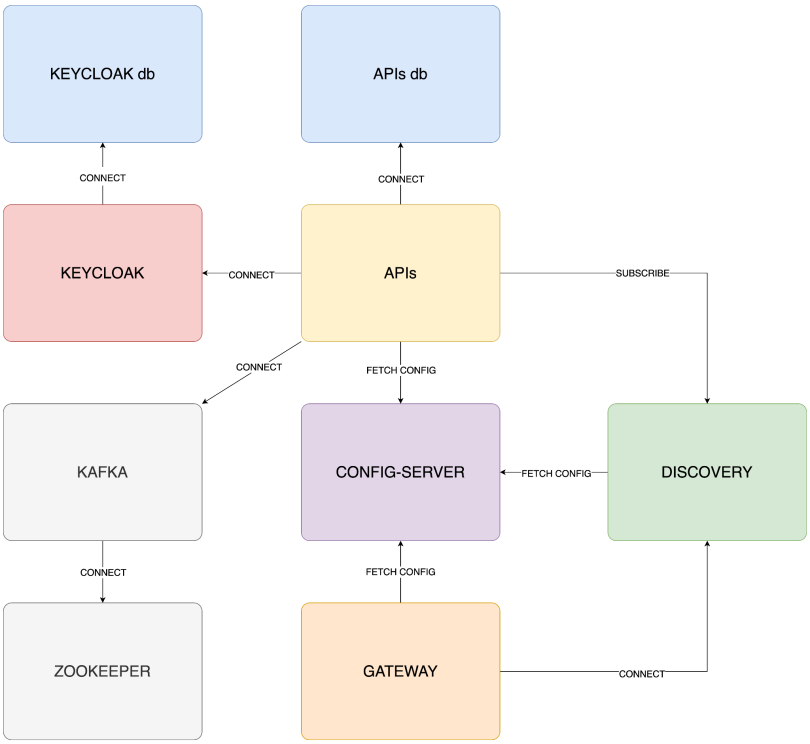


Figure n°2 - Schéma de dépendances inter-services

1.2. Git Flow

Nous utilisons des submodules, chaque service ayant son propre repository git, il est enregistré comme sous module de notre application complète. Ainsi, il est important d'avoir une harmonie entre les différents repositories et c'est pourquoi nous avons réalisé le diagramme suivant, qui définit le flow git a suivre. Un flow git c'est une stratégie de gestion de branches qui définit un modèle standardisé pour le développement, les fonctionnalités, les versions et les corrections de bugs.

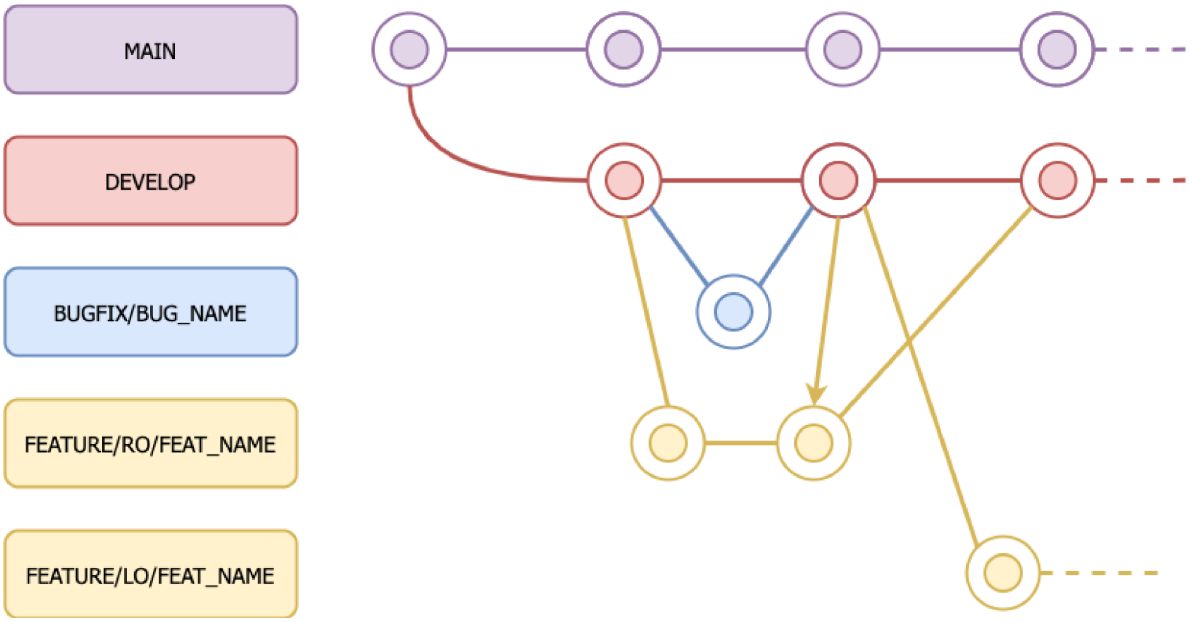


Figure n°3 - Schéma du git flow

1.3. Bases de données

Il est à noter que que le design de bases de données n'est pas tout à fait figé, en effet, celle-ci pourrait encore évoluer par rapport aux microservices qu'il reste encore à ajouter. Tout comme le schéma d'architecture, une version détaillée est disponible en annexe.

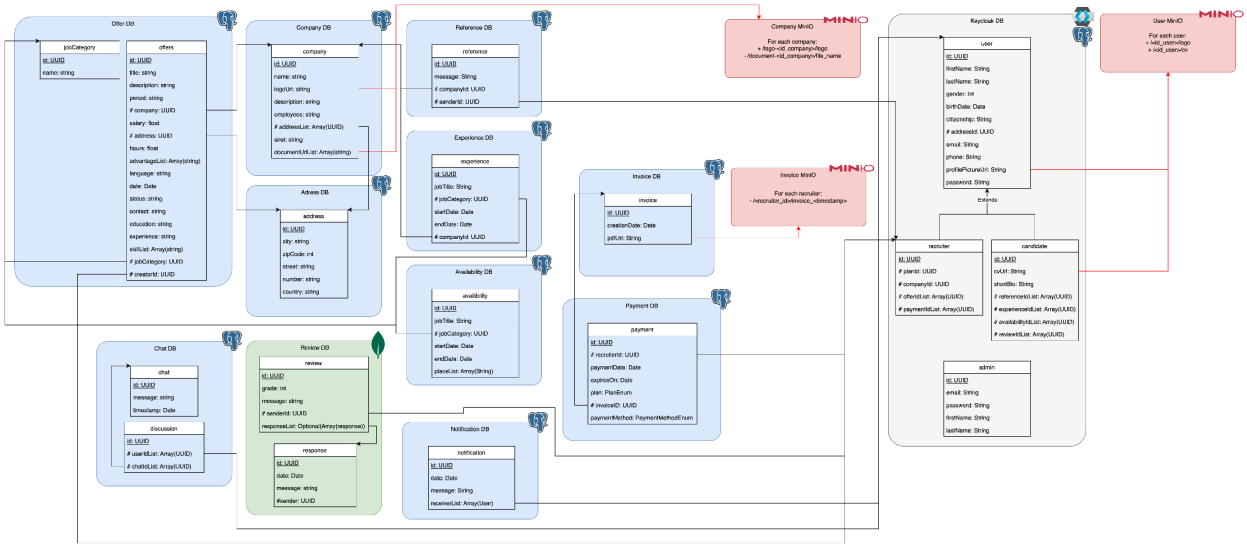


Figure n°4 - Schéma de la bases de données

2. Technologies utilisées

2.1. Back-end

- Spring Boot
  - Spring Boot
- Spring Webflux (pour les notifications push)
  - Spring WebFlux
- Quarkus (pour optimiser le déploiement)
  - Quarkus - Supersonic Subatomic Java
- Maven
  - Maven – Welcome to Apache Maven
- Kotlin
  - Kotlin Language
- Kafka
  - Apache Kafka
- Docker, docker-compose
  - Docker: Accelerated Container Application Development

2.2. Front-end

- React Native
  - React Native · Learn once, write anywhere
  - UseContext
    - useContext – React
  - TanStack Query
    - TanStack Query | React Query, Solid Query, Svelte Query, Vue Query
  - TS
    - JavaScript With Syntax For Types.
  - I18n
    - Introduction
  - Expo
    - Expo
- Yarn
  - Home page | Yarn

2.3. Stockage de données

- PostgreSQL
  - PostgreSQL

- MongoDB
    - MongoDB: La Plateforme De Données D'application
- Min.io (pour le stockage des images/fichiers)
    - MinIO | High Performance, Kubernetes Native Object Storage
- Hibernate (pour la persistence des données)
    - Hibernate. Everything data.

2.4. CI/CD

- GitHub Actions
    - Documentation GitHub Actions - Documentation GitHub

2.5. Securité

- Spring Security
    - Spring Security
- KeyCloak
    - Keycloak

3. Fonctionnalités implémentées

Lors de ce projet, nous avons implémenté beaucoup de fonctionnalités dans plusieurs micro-services dans le back-end. Nous avons également implémenté un maximum de ces fonctionnalités dans le front-end afin de l'alimenter et de rendre l'application utilisable dans sa première version. Nous allons voir dans cette partie les fonctionnalités qui sont déjà mises en place dans le front-end puis dans le back-end.

3.1. Front-end

Dans ce projet, nous n'avons pas mis l'accent sur le front-end. Notre objectif principal était de découvrir l'architecture micro-services et de la pousser au maximum de nos capacités. Nous avons donc fait le front-end avec sérieux et rigueur. Cependant, nous avions implémentés trop de choses en back-end et le temps nous a manqué pour les lier à l'interface. Voici une liste exhaustive des fonctionnalités implémentées dans le front-end de l'application:

- I18n Friendly
- Connexion
    - Token et Refresh Token
- Inscription Candidat / Recruteur
- Gestion des offres
- Gestion des avis
- Profil utilisateur (disponibilités,
- expériences, avis et références)
- Notifications (avec subscribe SSE)
- Suppression de comptes
- Panel administrateur
- Compatibilité Web et iOS
- Matching et gestion des recrutements
- Affichage dépendant des rôles

Nous aurions cependant aimé faire plus dans ce front-end et implémenter toutes les fonctionnalités que nous avons mises en place dans le back-end. Cependant, le temps nous a manqué pour finir cette partie de l'application.

3.2. Back-end

Le back-end est la partie principale de ce projet. En effet, nous avons vraiment mis l'accent sur cette partie-là, car notre objectif principal était de découvrir et mettre en œuvre l'architecture micro-services. Nous avons donc beaucoup travaillé sur la propreté de notre code back-end, la quantité de fonctionnalités et le respect des patterns professionnels. Voici une liste des fonctionnalités principales développées dans le back-end:

- Gateway, Config Server et Discovery Service
- Communication inter-services via Kafka
- Notifications envoyées par Server-Sent Event (SSE)
- Sécurité SSO avec Keycloak, OpenID Connect et OAuth 2
- Gestion des offres d'emploi
- Gestion des utilisateurs et de leur rôles
- Matching et gestion des recrutements
- Gestion des entreprises, de leur paiement et leur facture
- Gestion automatique du statuts des offres

Voici comment est géré le statut d'une offre une fois que celle-ci est terminée:

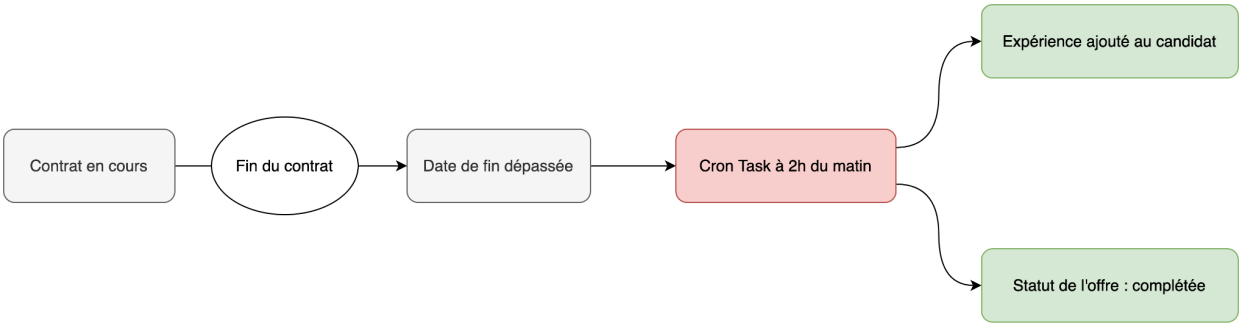


Figure n°5 - Schéma du fonctionnement de fin de contrat pour une offre

4. Stratégie de tests

4.1. Stratégie initiale

Notre projet étant structuré autour d'une architecture microservices avec l'utilisation de technologies comme Docker, Spring Boot et PostgreSQL, il requiert une stratégie de tests rigoureuse et bien définie. Cette stratégie vise à assurer non seulement la performance et la fiabilité de chaque micro service, mais aussi leur intégration harmonieuse au sein du système global. Cette stratégie se décline en trois parties principales : les tests unitaires, les tests d'intégration et l'automatisation des tests.

4.1.1. Tests unitaires

L'approche des tests unitaires est fondamentale dans notre processus de développement. Ces tests se concentrent sur la validation des fonctionnalités individuelles de chaque microservice. L'objectif est de s'assurer que chaque composant fonctionne de manière autonome, conformément à ses spécifications. En testant les unités de manière isolée, nous pouvons rapidement identifier et corriger les bugs, simplifiant ainsi le processus de débogage.

L'utilisation d'outils comme H2 et Mockito facilite ces tests en fournissant un environnement léger et contrôlé, idéal pour des tests précis et efficaces.

4.1.2. Tests systèmes et tests d'intégrations

Les tests d'intégrations prennent une importance particulière dans une architecture composée de multiples microservices. Ces tests visent à évaluer la façon dont ces services distincts interagissent les uns avec les autres. L'objectif est de s'assurer que l'ensemble du système fonctionne de manière cohérente et efficace, une fois que les composants individuels sont combinés.

L'utilisation de Postman pour simuler les requêtes API est essentielle pour tester ces interactions. Cela nous aide à vérifier que les services communiquent correctement entre eux, garantissant ainsi l'intégrité de notre backend. De plus notre application front-end permettra des tests de notre système complet.

4.1.3. Automatisation

L'automatisation des tests est un pilier central de notre stratégie. Elle vise à intégrer les tests dans le cycle de vie du développement logiciel, de manière à ce qu'ils soient exécutés de façon systématique et régulière. L'objectif est de minimiser les erreurs humaines, de réduire le temps de test et d'accélérer le cycle de publication.

En utilisant Jenkins pour notre CI/CD, nous pouvons automatiser les tests à chaque étape du développement, offrant ainsi une assurance qualité continue. Cela permet de détecter rapidement les problèmes, réduisant ainsi le temps et les coûts associés à leur résolution, et garantit que le produit final est fiable et prêt pour la mise en production.

4.1.4. Conclusion

En somme, cette stratégie de tests couvre tous les aspects cruciaux pour assurer la qualité et l'efficacité de notre système basé sur les microservices. Elle est conçue pour répondre aux défis spécifiques posés par ce type d'architecture, garantissant ainsi la fiabilité et la robustesse de notre application.

4.2. Mise en oeuvre des tests

Dans le cadre de notre projet, la mise en œuvre des tests est structurée en plusieurs étapes, chacune ciblant différents aspects et utilisant des technologies spécifiques pour garantir l'efficacité des tests.

4.2.1. Tests Unitaires

Pour les tests unitaires, nous avons choisi une combinaison de technologies adaptées à nos besoins. Nous utilisons Mockito pour simuler le comportement des dépendances, ce qui est essentiel pour tester les composants de manière isolée. H2, une base de données en mémoire, est utilisée pour simuler notre base de données PostgreSQL, permettant des tests rapides sans affecter la base de données réelle. Spring Boot Test et Spring Security Test fournissent un cadre robuste pour tester nos applications Spring Boot, y compris les aspects de sécurité.

Dans le cadre de nos tests unitaires, nous avons concentré nos efforts sur trois API : API Experience, API Availability et API Reference. Chacune de ces APIs a été soumise à une série de tests approfondis.

- API Experience : Pour cette API, nous avons testé à la fois le contrôleur et le service. Le contrôleur a été testé pour s'assurer que toutes les routes d’apis (get all, get by id, create, update, et delete) fonctionnent correctement. Du côté du service, nous avons réalisé une dizaine de tests, couvrant les fonctions et différents cas possibles, comme les erreurs de récupération par ID ou les mises à jour d'éléments inexistants.
- API Availability : Des tests similaires ont été effectués pour l'API Availability. Le contrôleur a été testé sur toute ses routes, et le service a été testé dans une dizaine scénarios différents, y compris des cas d'erreurs et de validations de données.
- API Reference : Pour cette API, le processus a été répété avec toutes les routes qui ont été testées pour le contrôleur et une dizaine de tests sur différents cas d’utilisation pour le service.

En testant trois API - Experience, Availability et Reference - nous avons pu couvrir de nombreux scénarios. Cela nous a permis de nous assurer que chaque fonctionnalité individuelle de nos microservices fonctionne correctement, renforçant la qualité et la fiabilité de chaque composant du système. Néanmoins le nombre d'API testées reste insuffisant au vu du nombre d'API présentes dans notre architecture.

4.2.2. Tests systèmes et tests d'intégrations

Pour les tests d'intégrations, nous avons principalement utilisé Postman, un outil puissant pour tester les API. Il nous a permis de simuler des requêtes client et de vérifier les réponses des services. En complément, nous avons utilisé une application front-end pour tester notre système dans son intégralité.

Dans cette phase, tous les parcours et toutes les API ont été testés, avec plus de trois cents tests réalisés. Ces tests d'intégration ont permis de valider le fonctionnement conjoint des différents microservices et de garantir que les interactions entre eux sont conformes aux attentes.

La liste de tous les tests réalisés est disponible en annexe ici.

4.2.3. Automatisation

Pour l'automatisation des tests, nous avons intégré GitHub Actions, ce qui nous a permis d'automatiser les tests unitaires pour les trois APIs testées. Bien que le projet ne soit pas encore déployé et qu'une pipeline CI/CD plus complexe ne soit pas mise en place, cette étape d'automatisation joue un rôle crucial dans le maintien de la qualité et la détection rapide des problèmes au fur et à mesure du développement. De plus, nous avons automatisé certains processus postman, par exemple le processus consistant à copier coller le token d'accès dans le champ dédié pour chaque requête.

```
// Récupération dans le body de la réponse
var jsonData = JSON.parse(responseBody);
// Le token est stocké dans une variable d'environnement et accédé par
toutes les requêtes
pm.globals.set("token", jsonData.access_token)
```

Figure n°6 - Exemple de code pour automatiser le processus de login dans postman

4.2.4. Conclusion

En résumé, cette mise en œuvre des tests, bien que technique, s'aligne étroitement avec notre objectif de maintenir une qualité élevée à travers toutes les phases du développement, en assurant que chaque composant fonctionne comme prévu de manière individuelle et en interaction avec les autres.

5. Retour sur expérience

5.1 Découverte de Nouvelles Technologies

Au cours de ce projet, nous avons eu l'occasion de nous familiariser avec un large éventail de technologies. Parmi celles-ci, Keycloak nous a permis de gérer efficacement l'authentification et l'autorisation, tandis que l'écosystème Spring, avec des composants comme Spring Gateway, Spring Discovery, Spring Config et Spring Boot, a été essentiel pour construire une architecture robuste et évolutive. Spring Webflux nous a offert une approche réactive, et l'intégration de Kafka a ajouté une dimension puissante de traitement de flux de données en temps réel. Pour la gestion de fichiers, Minio a été une découverte pratique, et Kotlin s'est révélé être un langage de programmation polyvalent et efficace. Enfin, Docker Compose et React Native ont complété notre boîte à outils, nous permettant de créer des environnements de développement cohérents et de développer des applications mobiles réactives.

5.2 Renforcement des Compétences Existantes

Parallèlement à ces découvertes, le projet nous a permis de renforcer nos compétences dans des technologies que nous connaissions déjà. Docker a été un outil indispensable pour la gestion de conteneurs, garantissant la cohérence entre nos environnements de développement et de production. Java a continué d'être notre pilier pour le développement backend, avec son écosystème riche et sa robustesse. La gestion des données a été assurée par l'utilisation combinée de PostgreSQL pour les données relationnelles et MongoDB pour les besoins non relationnels, nous donnant ainsi une grande flexibilité dans la manipulation des données.

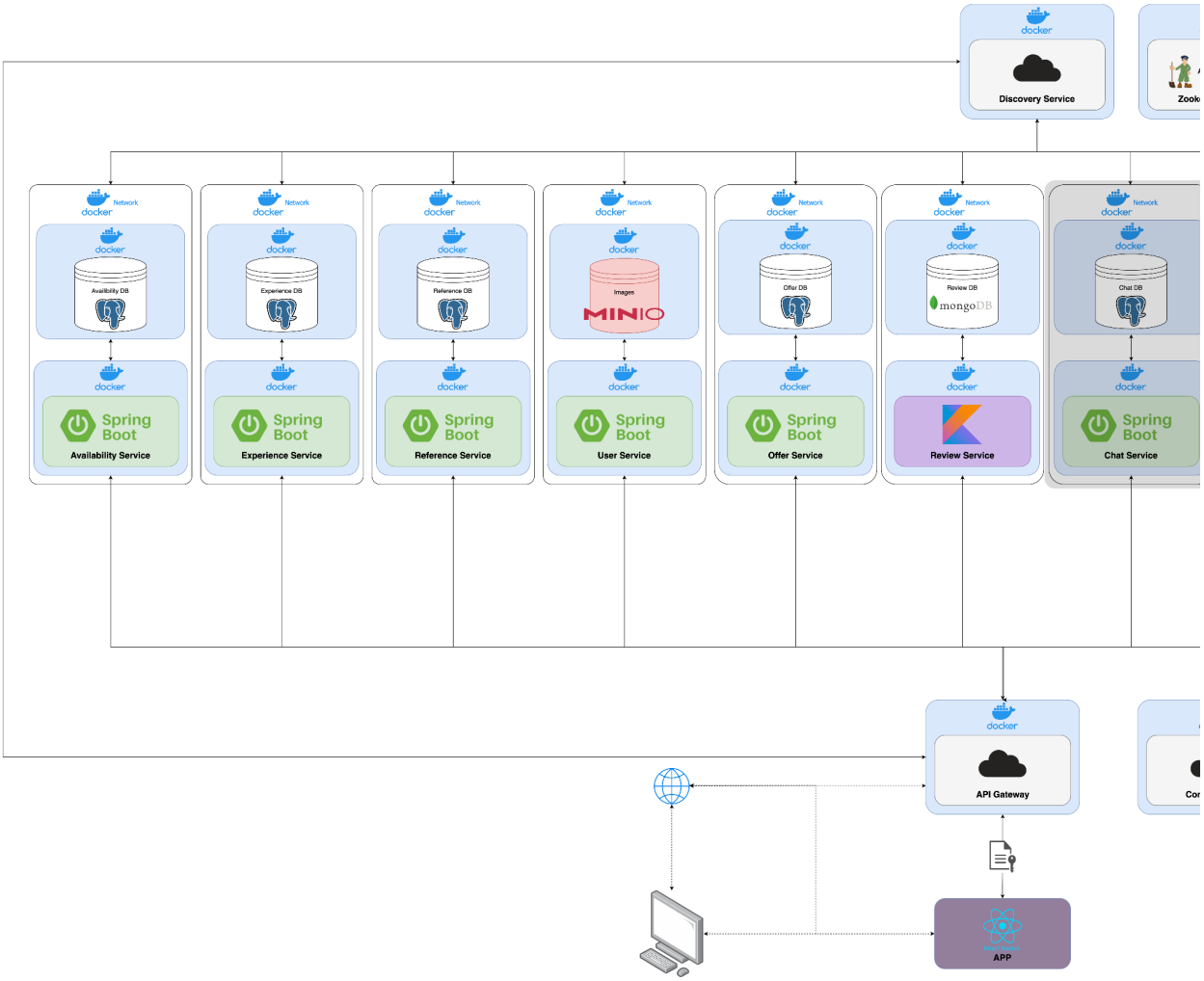
5.3 Un Projet Professionnalisant et Stimulant

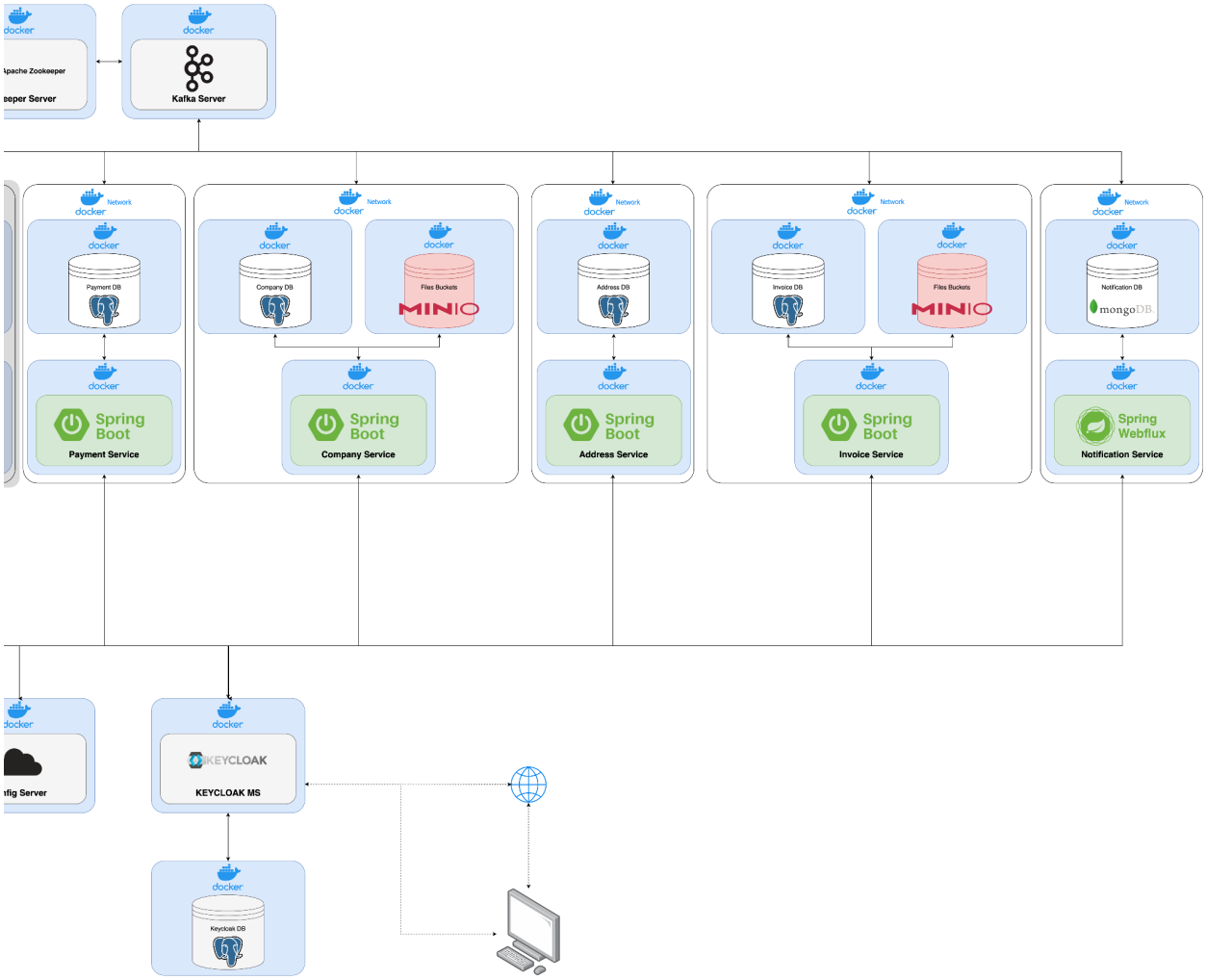
Ce projet a représenté une étape significative dans notre parcours professionnel. Avec une charge de travail importante et une complexité technique non négligeable, il nous a permis d'acquérir un bagage technique considérable juste avant la fin de nos études supérieures. Cette expérience a été une opportunité exceptionnelle pour nous préparer aux défis du monde réel, renforçant ainsi notre confiance en nos capacités et en notre potentiel.

Au-delà des aspects techniques et professionnels, ce projet a été pour nous une source d'épanouissement personnel. Nous avons particulièrement apprécié travailler sur le backend, où nous avons pu expérimenter, innover et relever des défis stimulants.

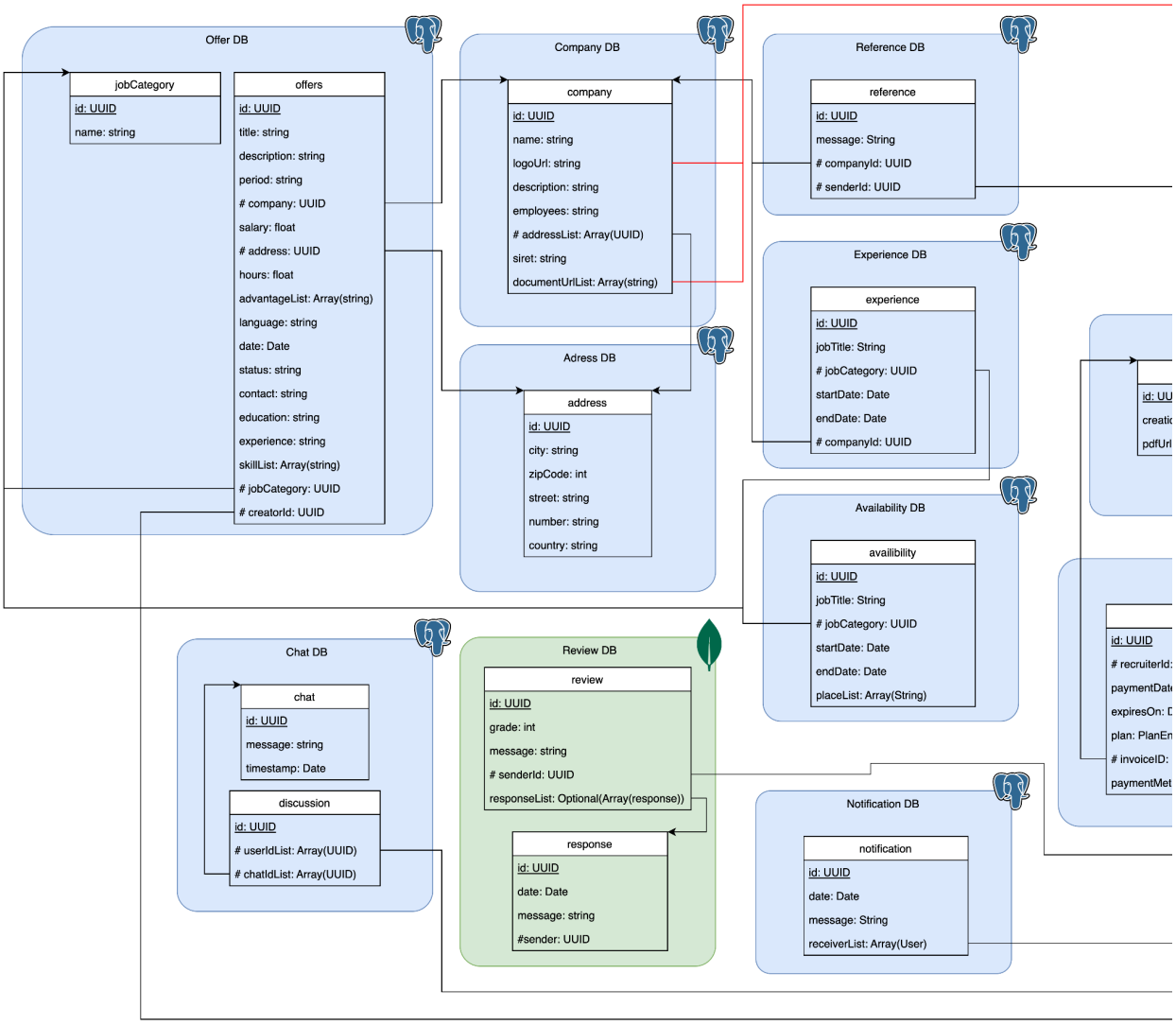
Annexes

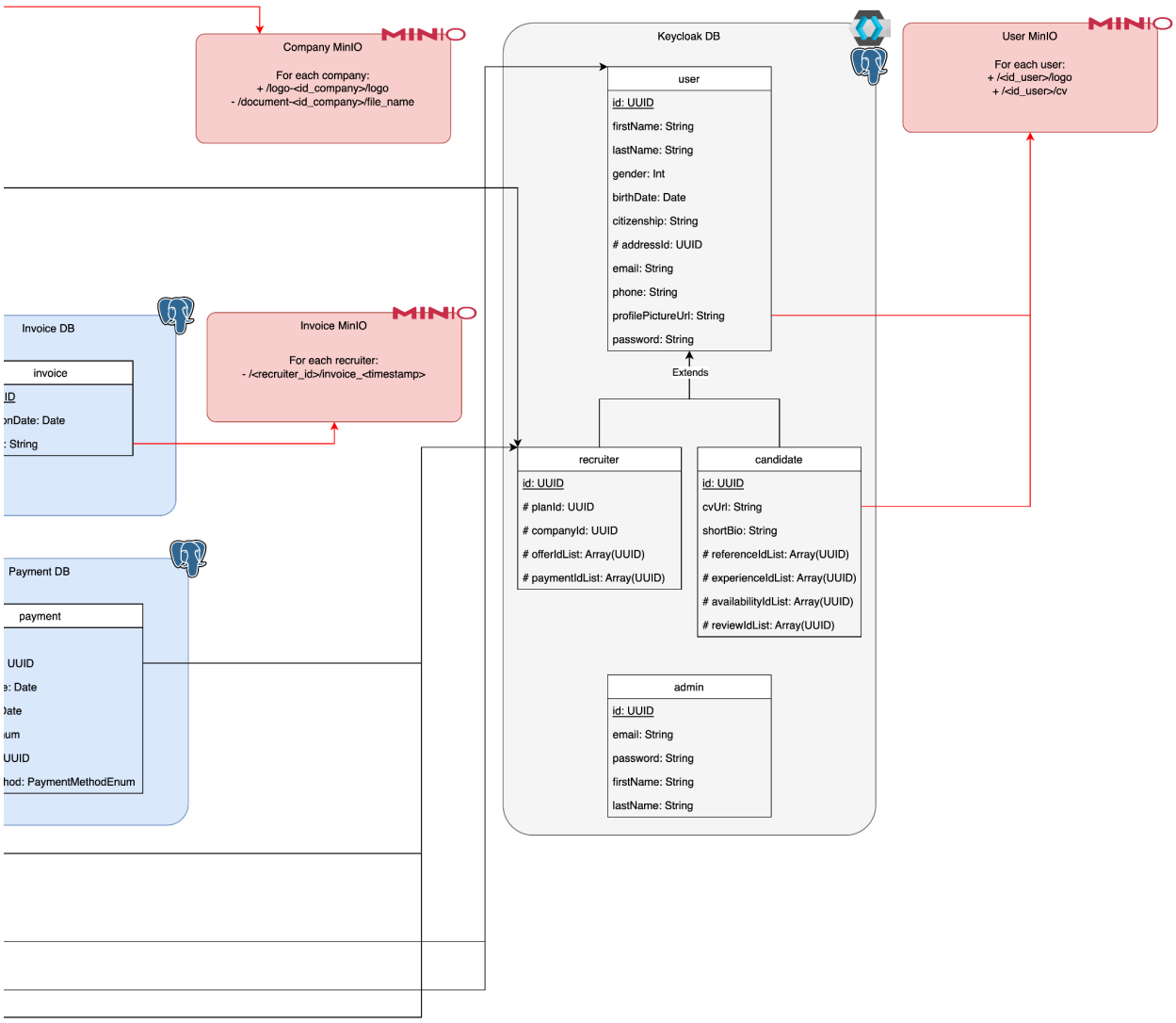
Annexe n°1: Schéma d’architecture en plusieurs parties




Kafka Server
Payment DB
Company DB
Files Buckets
Address DB
Invoice DB
Files Buckets
Notification DB
mongoDB
Spring Boot
Payment Service
Spring Boot
Company Service
Spring Boot
Address Service
Spring Boot
Invoice Service
Spring Webflux
Notification Service
KEYCLOAK
KEYCLOAK MS
Keycloak DB

Annexe n°2: Schéma de bases de données en plusieurs parties




Company MinIO
For each company:
+ /logo-<id_company>/logo
- /document-<id_company>/file_name
Keycloak DB
user
id: UUID
firstName: String
lastName: String
gender: Int
birthDate: Date
citizenship: String
addressId: UUID
email: String
phone: String
profilePictureUrl: String
password: String
Extends
recruiter
id: UUID
planId: UUID
companyId: UUID
offerIdList: Array(UUID)
paymentIdList: Array(UUID)
candidate
id: UUID
cvUrl: String
shortBio: String
referenceIdList: Array(UUID)
experienceIdList: Array(UUID)
availabilityIdList: Array(UUID)
reviewIdList: Array(UUID)
admin
id: UUID
email: String
password: String
firstName: String
lastName: String
User MinIO
For each user:
+ /<id_user>/logo
+ /<id_user>/cv
Invoice DB
invoice
Invoice MinIO
For each recruiter:
- /<recruiter_id>/invoice_<timestamp>
Payment DB
payment
UUID
Date
UUID
hod: PaymentMethodEnum

Annexe n°3 : Liste des tests réalisé avec Postman

API Address : `/address`

Créer une Adresse : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec un token invalide : retourne une erreur 403 (Interdit).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle adresse et retourne les détails de celle-ci.

Récupérer toutes les Adresses : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste d'adresses.

Récupérer une Adresse par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails de l'adresse.

Mettre à jour une Adresse : `/`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Mise à jour avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour l'adresse et retourne les détails mis à jour.

API Availability : `/availability`

Récupérer toutes les disponibilités : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste de disponibilités.

Récupérer une disponibilité par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails de la disponibilité.

Créer une disponibilité : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle disponibilité et retourne les détails de celle-ci.

Supprimer une disponibilité : `/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime la disponibilité et retourne une confirmation.

Mettre à jour une disponibilité : `/`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Mise à jour avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour la disponibilité et retourne les détails mis à jour.

API Company : `/company`

Récupérer toutes les entreprises : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste d'entreprises.

Récupérer une entreprise par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails de l'entreprise.

Créer une entreprise : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle entreprise et retourne les détails de celle-ci.

Mettre à jour une entreprise : `/{id}`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Mise à jour avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour l'entreprise et retourne les détails mis à jour.

Supprimer une entreprise : `/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime l'entreprise et retourne une confirmation.

API Experience : `/experience`

Récupérer toutes les expériences : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste d'expériences.

Récupérer une expérience par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails de l'expérience.

Créer une expérience : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle expérience et retourne les détails de celle-ci.

Mettre à jour une expérience : `/`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Mise à jour avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour l'expérience et retourne les détails mis à jour.

Supprimer une expérience : `/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime l'expérience et retourne une confirmation.

API Invoice : `/invoice`

Créer une Facture : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle facture et retourne les détails de celle-ci.

Récupérer toutes les Factures : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste de factures.

Récupérer une Facture par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails de la facture.

Obtenir l'URL PDF d'une Facture : `/url/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête valide avec un ID existant : retourne l'URL du PDF de la facture.

Supprimer une Facture : `/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime la facture et retourne une confirmation.

API Job Category : `/job-category`

Créer une Catégorie d'Emploi : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle catégorie d'emploi et retourne les détails de celle-ci.

Récupérer toutes les Catégories d'Emploi : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste de catégories d'emploi.

API Notification : `/notification`

Récupérer toutes les notifications : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste de notifications.

Récupérer une notification par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails de la notification.

Récupérer toutes les notifications par ID de destinataire : `/user/{userId}`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID de destinataire existant : retourne les notifications correspondantes.

Supprimer une notification par son ID : `/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime la notification et retourne une confirmation.

Proxy pour SSE : `/sse/proxy/{id}`
Cas d'utilisation testés :

- Connexion sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Connexion avec un token invalide : retourne une erreur 403 (Interdit).
- Connexion valide : permet de se connecter à un service SSE externe via un proxy.

Connecter au SSE (Server-Sent Events) : `/sse/subscribe/{token}`
Cas d'utilisation testés :

- Connexion avec un token SSE invalide : retourne une erreur de connexion.
- Connexion valide : établit une connexion SSE pour recevoir des notifications en temps réel.

API Offers : `/offers`

Créer une Offre : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle offre et retourne les détails de celle-ci.

Récupérer toutes les Offres : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste d'offres.

Récupérer une Offre par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails de l'offre.

Récupérer les Offres par son ID d'entreprise : `/company/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails de l'offre.

Mettre à jour une Offre : `/{id}`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Mise à jour avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour l'offre et retourne les détails mis à jour.

Supprimer une Offre : `/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime l'offre et retourne une confirmation.

Ajouter un Utilisateur Recruté : `/offer/{offerId}/recruited/add/{userId}`
Cas d'utilisation testés :

- Ajout sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Ajout avec des données invalides : retourne une erreur 400 (Mauvaise requête).
- Ajout valide : ajoute un utilisateur recruté à l'offre et retourne une confirmation.

Supprimer un Utilisateur Recruté : `/offer/{offerId}/recruited/remove`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression avec un ID d'offre ou d'utilisateur inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime un utilisateur recruté de l'offre et retourne une confirmation.

API Payment : `/payment`

Créer un Paiement : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création sans être un recruteur : retourne une erreur 403 (Interdit)
- Création avec des données valides : crée un nouveau paiement et retourne les détails de celui-ci.

Récupérer un Paiement par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails du paiement.

Récupérer un Paiement par ID d'Utilisateur : `/user/{userId}`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID d'utilisateur existant : retourne les paiements correspondants.

Récupérer tous les Paiements : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création sans être un admin : retourne une erreur 403 (Interdit)
- Requête valide : retourne une liste de paiements.

Récupérer les Méthodes de Paiement : `/methods/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête valide : retourne une liste des méthodes de paiement disponibles.

API Plan : `/plan`

Créer un Plan : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création sans être admin : retourne une erreur 403 (Interdit).
- Création avec des données valides : crée un nouveau plan et retourne les détails de celui-ci.

Récupérer tous les Plans : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête valide : retourne une liste de plans.

Récupérer un Plan par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête valide avec un ID existant : retourne les détails du plan.

Récupérer tous les Plans par Devise : `/currency/{currency}`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec une devise inexistante : retourne une erreur 404 (Non trouvé).
- Requête valide avec une devise spécifique : retourne les plans correspondant à cette devise.

Mettre à jour un Plan : `/{id}`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Modification sans être admin : retourne une erreur 403 (Interdit).
- Mise à jour avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour le plan et retourne les détails mis à jour.

Supprimer un Plan : `/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression sans être admin : retourne une erreur 403 (Interdit).

- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime le plan et retourne une confirmation.

API Reference : `/reference`

Récupérer toutes les Références : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste de références.

Récupérer une Référence par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide avec un ID existant : retourne les détails de la référence.

Créer une Référence : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle référence et retourne les détails de celle-ci.

Mettre à jour une Référence : `/`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Mise à jour avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour la référence et retourne les détails mis à jour.

Supprimer une Référence : `/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime la référence et retourne une confirmation.

API Review : `/review`

Récupérer toutes les Avis : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste d'avis.

Récupérer un Avis par son ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête valide avec un ID existant : retourne les détails de l'avis.

Créer un Avis : `/`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée un nouvel avis et retourne les détails de celui-ci.

Ajouter une Réponse à un Avis : `/add/response/{reviewId}`
Cas d'utilisation testés :

- Ajout sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Ajout avec un ID de revue inexistant : retourne une erreur 404 (Non trouvé).
- Ajout valide : ajoute une réponse à la revue spécifiée et retourne les détails.

Mettre à jour un Avis : `/modify/{id}`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Mise à jour avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour l'avis et retourne les détails mis à jour.

Supprimer un Avis : `/delete/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime l'avis et retourne une confirmation.

Keycloak : `/keycloak`

Login : `/login`
Cas d'utilisation testés :

- Connexion avec des identifiants invalides : retourne une erreur d'authentification.
- Connexion avec des identifiants valides : retourne un token d'accès.

Refresh : `/refresh`
Cas d'utilisation testés :

- Rafraîchissement avec un token invalide ou expiré : retourne une erreur.
- Rafraîchissement avec un token valide : retourne un nouveau token d'accès.

Admin : `/admin`
Cas d'utilisation testés :

- Connexion en tant qu'admin avec des identifiants invalides : retourne une erreur d'authentification.
- Connexion en tant qu'admin avec des identifiants valides : retourne un token d'accès admin.

Register : `/register`
Cas d'utilisation testés :

- Enregistrement avec des informations incomplètes : retourne une erreur.
- Enregistrement avec des informations valides : crée un nouvel utilisateur dans Keycloak et retourne les détails de l'utilisateur.

Users : `/users`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête valide : retourne une liste des utilisateurs dans Keycloak.

User by id : `/users/{id}`
Cas d'utilisation testés :

- Requête pour un ID utilisateur inexistant : retourne une erreur.
- Requête valide pour un ID utilisateur existant : retourne les détails de l'utilisateur.

API User : `/user`

Authentification et Gestion des Utilisateurs

Login : `/auth/login`
Cas d'utilisation testés :

- Connexion avec des identifiants invalides : erreur d'authentification (401 Non autorisé).
- Connexion avec des identifiants valides : renvoie un token d'accès.

Refresh Token : `/auth/refresh`
Cas d'utilisation testés :

- Rafraîchissement avec un token invalide : erreur d'authentification (401 Non autorisé).
- Rafraîchissement avec un token valide : renvoie un token d'accès.

Register : `/auth/register`
Cas d'utilisation testés :

- Enregistrement avec des informations incomplètes : renvoie une erreur (400 Mauvaise requête).
- Enregistrement avec des informations complètes : renvoie un token d'accès.

Logout : `/logout/{id}`
Cas d'utilisation testés :

- Déconnexion échouée : renvoie une erreur.
- Déconnexion réussie : renvoie `true`.

Gestion des Disponibilités

Ajouter une Disponibilité : `/availability/add/{id}`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle disponibilité, l'ajoute à l'utilisateur et retourne l'utilisateur.

Supprimer une Disponibilité : `/availability/remove/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression avec des données invalides : retourne une erreur 404 (Non trouvé).
- Suppression avec des données valides : supprime la disponibilité, la retire de l'utilisateur et retourne l'utilisateur.

Gestion des Expériences

Ajouter une Expérience : `/experience/add/{id}`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle expérience, l'ajoute à l'utilisateur et retourne l'utilisateur.

Supprimer une Expérience : `/experience/remove/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression avec des données invalides : retourne une erreur 404 (Non trouvé).
- Suppression avec des données valides : supprime l'expérience, la retire de l'utilisateur et retourne l'utilisateur.

Gestion des Images de Profil

Ajouter une image de profil : `/profile-picture/add/{id}`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : sauvegarde une image, ajoute l'url à l'utilisateur et retourne l'utilisateur.

Supprimer une image de profil : `/profile-picture/remove/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression avec des données invalides : retourne une erreur 404 (Non trouvé).
- Suppression avec des données valides : supprime une image, supprime l'url de l'utilisateur et retourne l'utilisateur.

Gestion des Références

Ajouter une Référence : `/reference/add/{id}`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée une nouvelle référence, l'ajoute à l'utilisateur et retourne l'utilisateur.

Supprimer une Référence : `/reference/remove/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression avec des données invalides : retourne une erreur 404 (Non trouvé).
- Suppression avec des données valides : supprime la référence, la retire de l'utilisateur et retourne l'utilisateur.

Gestion des Reviews

Ajouter un Avis : `/review/add/{id}`
Cas d'utilisation testés :

- Création sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Création avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Création avec des données valides : crée un nouvel avis, l'ajoute à l'utilisateur et retourne l'utilisateur.

Supprimer un Avis : `/review/remove/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression avec des données invalides : retourne une erreur 404 (Non trouvé).
- Suppression avec des données valides : supprime l'avis, le retire de l'utilisateur et retourne l'utilisateur.

Recherche et Gestion des Utilisateurs

Recherche de Recruteurs :
`/search-recruiters?firstName={value}&lastName={value}`
Cas d'utilisation testés :

- Recherche sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Recherche sans être candidat : retourne une erreur 403 (Interdit).
- Recherche avec des données invalides : retourne une erreur 400 (Mauvaise requête).
- Recherche avec des données valides : renvoie une liste de recruteurs.

Obtenir tous les Utilisateurs : `/`
Cas d'utilisation testés :

- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête avec un token invalide : retourne une erreur 403 (Interdit).
- Requête valide : retourne une liste d'utilisateurs.

Obtenir un Utilisateur par ID : `/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête valide avec un ID existant : retourne un utilisateur.

Obtenir un Utilisateur détaillé par ID : `/detailed/{id}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête valide avec un ID existant : retourne un utilisateur détaillé.

Obtenir les utilisateurs concerné par une offre d'emploi : `/match/{offerId}`
Cas d'utilisation testés :

- Requête avec un ID inexistant : retourne une erreur 404 (Non trouvé).
- Requête sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Requête valide avec un ID existant : retourne une liste d'utilisateurs.

Mettre à jour un Utilisateur : `/{id}`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Mise à jour avec des données incomplètes ou invalides : retourne une erreur 400 (Mauvaise requête).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour l'utilisateur et le retourne.

Demande de suppression de compte : `/delete-me/{id}`
Cas d'utilisation testés :

- Mise à jour sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Mise à jour d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Mise à jour valide : met à jour le statut de l'utilisateur.

Supprimer un Utilisateur : `/{id}`
Cas d'utilisation testés :

- Suppression sans token d'authentification : retourne une erreur 401 (Non autorisé).
- Suppression d'un ID inexistant : retourne une erreur 404 (Non trouvé).
- Suppression valide : supprime l'utilisateur et retourne une confirmation.