TEK-UP
Private Higher School of Technologies & Engineering

Projets Web Services

Sujet : Suivi des candidatures dans une entreprise tech

Réalisé par : Ridene Saif Eddine

Année universitaire : 2024-2025

Table des Matiéres

Introduction

Dans le cadre de la transformation digitale des entreprises, la gestion efficace des candidatures à des offres d'emploi et de stage constitue un enjeu majeur.

Ce projet vise à développer un service web permettant à une entreprise technologique de centraliser, rationaliser et automatiser la gestion de ses candidatures. Le service permettra aux candidats de soumettre leurs candidatures, et aux recruteurs de suivre le processus d'évaluation, depuis la réception jusqu'à la décision finale.

Ce rapport détaille les différentes phases du projet, depuis l'analyse initiale jusqu'à la mise en œuvre technique, en s'appuyant sur une architecture moderne utilisant NestJS, GraphQL et MySQL.

Contexte du projet

Les entreprises technologiques reçoivent quotidiennement un grand nombre de candidatures pour des postes variés, allant des stages aux emplois à temps plein. Gérer ces candidatures manuellement est souvent source d'erreurs, de pertes d'informations et de délais importants dans le traitement des dossiers.

L'entreprise cliente souhaite ainsi disposer d'un outil numérique performant capable de centraliser toutes les candidatures, d'assurer un suivi clair et transparent des statuts, et de conserver un historique des décisions prises pour chaque candidature.

Ce contexte nécessite la conception d'un service web robuste et flexible, accessible via une API moderne, afin d'améliorer la productivité des équipes de recrutement et offrir une meilleure expérience aux candidats.

Objectifs du service web

L'objectif principal de ce service web est de faciliter la gestion des candidatures dans un environnement numérique sécurisé et efficace. Plus précisément, les objectifs sont :

- Permettre aux candidats de soumettre leurs candidatures en ligne pour différents postes disponibles.
- Offrir la possibilité de suivre le statut de chaque candidature en temps réel, avec des étapes clairement définies (soumis, en revue, accepté, rejeté).
- Enregistrer et consulter l'historique complet des décisions et commentaires associés à chaque candidature, afin d'assurer la traçabilité des processus de recrutement.
- Fournir une interface d'API GraphQL intuitive pour permettre aux clients internes (recruteurs, RH) et externes (candidats, services tiers) de consulter et manipuler les données relatives aux candidatures, postes, et candidats.
- Garantir une architecture évolutive permettant l'ajout futur de fonctionnalités complémentaires comme la gestion des entretiens ou la notification automatique.

Phase 1 : Analyse et Planification

La phase d'analyse et planification constitue une étape cruciale dans la réussite du projet. Elle permet de bien comprendre le problème à résoudre, de définir précisément le périmètre fonctionnel et technique, ainsi que d'identifier les ressources nécessaires pour la mise en œuvre. Cette phase sert aussi de base pour la conception du modèle de données et la définition des interfaces API.

1.1 Analyse du problème

L'enjeu principal est de créer un système permettant de gérer efficacement les candidatures pour différents postes proposés par l'entreprise. Actuellement, la gestion manuelle entraîne plusieurs difficultés :

- Difficulté à centraliser toutes les candidatures, ce qui complique leur suivi.
- Risques d'erreurs dans le traitement, avec des candidatures perdues ou mal classées.
- Manque de transparence pour les candidats sur l'état d'avancement de leur dossier.
- Complexité pour les recruteurs à obtenir une vue globale et historique des décisions.

Pour répondre à ces problèmes, le projet doit proposer une solution numérique intuitive, sécurisée, et capable d'automatiser certaines étapes du processus, tout en offrant un accès contrôlé aux différents profils utilisateurs.

1.2 Identification des entités et relations

Pour modéliser le domaine fonctionnel, plusieurs entités principales ont été identifiées, avec leurs relations :

- **Candidat** : représente une personne postulant à un poste. Contient les informations personnelles, coordonnées, CV, etc.
- **Offre de poste** : décrit un emploi ou stage proposé, avec titre, description, département, type de contrat, etc.
- **Candidature** : lien entre un candidat et une offre, incluant la date de soumission, statut actuel, et commentaires.
- **Statut de candidature** : représente les différentes étapes du traitement (ex. soumis, en cours de revue, accepté, rejeté).
- **Utilisateur (recruteur, RH)** : gestion des accès et permissions pour gérer les candidatures, offres et utilisateurs.

Ces entités seront liées par des relations de type One-to-Many (ex. un candidat peut avoir plusieurs candidatures, une offre plusieurs candidatures), et Many-to-One pour les statuts.

1.3 Liste des fonctionnalités (Requêtes et Mutations)

Le service web sera basé sur une API GraphQL, proposant les fonctionnalités suivantes, regroupées en requêtes (queries) et mutations :

Requêtes (Queries) :

- Récupérer la liste des offres disponibles.
- Consulter les détails d'une offre spécifique.
- Obtenir la liste des candidatures d'un candidat donné.

- Consulter le détail d'une candidature (statut, historique, commentaires).
- Lister les candidats selon différents critères (ex. par offre, par statut).

Mutations :

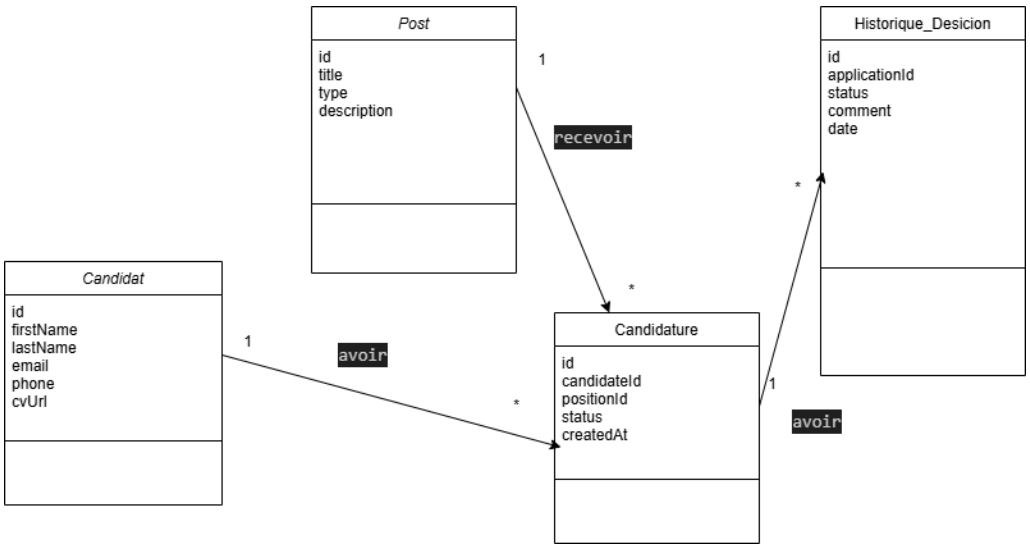
- Soumettre une nouvelle candidature pour une offre.
- Modifier ou annuler une candidature en cours (selon statut).
- Mettre à jour le statut d'une candidature (par un recruteur).
- Ajouter des commentaires ou décisions sur une candidature.
- Créer, modifier ou supprimer une offre de poste (par un utilisateur habilité).

Phase 2 : Conception

Cette phase consiste à traduire les besoins fonctionnels et les résultats de l'analyse en une architecture logicielle claire et structurée. Elle prépare la mise en œuvre en définissant précisément les modèles de données et les interfaces de communication.

2.1 Diagramme des entités/classes (UML)

Le diagramme UML des entités permet de visualiser la structure des données ainsi que les relations entre les différents objets métier du système. Il constitue une base solide pour le développement des classes en backend.



Description du diagramme :

- **Candidat** : classe avec les attributs id, nom, prénom, email, téléphone, CV (fichier ou lien), etc.
- **Poste** : classe avec id, titre, description, datePublication, typeContrat, departement, etc.
- **Candidature** : classe faisant le lien entre Candidat et OffreDePoste, avec id, dateSoumission, statut, commentaires, etc.
- **Historique_decision** : classe ou énumération définissant les différents états possibles (Soumis, En cours, Accepté, Rejeté).

Les relations principales sont :

- Un **Candidat** peut avoir plusieurs **Candidatures** (relation 1-N).
- Une **Poste** peut être liée à plusieurs **Candidatures** (relation 1-N).
- Une **Candidature** possède un unique **Statut**.

Ce diagramme permettra une organisation claire des classes Java/Spring Boot ou autres langages choisis pour le backend.

2.2 Conception du schéma GraphQL

Le schéma GraphQL formalise les types, requêtes, mutations et leurs relations. Il est conçu pour refléter fidèlement le modèle métier tout en optimisant les interactions avec le client.

Types GraphQL principaux :

- **CandidateType** : type avec champs id, nom, prenom, email, telephone, cv.
- **PositionType** : champs id, titre, description, datePublication, typeContrat, departement.
- **ApplicationType** : champs id, candidat (référence), offreDePoste (référence), dateSoumission, statut, commentaires.
- **StatusEnum** : énumération pour les statuts possibles de candidature.

Requêtes (Query) :

- Position: liste des offres disponibles.
- Position (id: ID!): détail d'une offre spécifique.
- applicationsByCandidate(candidateId: ID!): candidatures d'un candidat.
- application(id: ID!): détail d'une candidature.
- candidates(filter: CandidateFilter): recherche de candidats selon critères.

Mutations :

- submitApplication(candidateId: ID!, offerId: ID!): soumettre une candidature.
- updateApplicationStatus(applicationId: ID!, newStatus: StatusEnum!, comment: String): mise à jour du statut.
- addOffer(input: OfferInput!): création d'une offre.
- updateOffer(id: ID!, input: OfferInput!): modification d'une offre.
- deleteOffer(id: ID!): suppression d'une offre.

Exemples de schéma GraphQL (simplifié) :

```
enum StatusEnum {
  SOUMIS
  EN_COURS
  ACCEPTE
  REJETE
}
```

```
type Candidate {
  id: ID!
  nom: String!
  prenom: String!
  email: String!
  telephone: String
  cv: String
}

type Position {
  id: ID!
  titre: String!
  description: String!
  datePublication: String!
  typeContrat: String!
  departement: String
}

type Application {
  id: ID!
  candidat: Candidate!
  offreDePoste: Position!
  dateSoumission: String!
  statut: StatusEnum!
  commentaires: String
}

type Query {
 Position: [JobOffer!]!
```

```
 Position (id: ID!): JobOffer
  applicationsByCandidate(candidateId: ID!): [Application!]!
  application(id: ID!): Application
}

type Mutation {
  submitApplication(candidateId: ID!, offerId: ID!): Application
  updateApplicationStatus(applicationId: ID!, newStatus: StatusEnum!, comment: String): Application
  addOffer(input: OfferInput!): Position
  updateOffer(id: ID!, input: OfferInput!): Position
  deleteOffer(id: ID!): Boolean
}

input OfferInput {
  titre: String!
  description: String!
  typeContrat: String!
  departement: String
}
```

2.3 Conception du schéma GraphQL

2.3.1 Types d'objets

Les **types d'objets** définissent la structure des données qui seront retournées par le serveur GraphQL. Ils représentent les entités métier principales avec leurs champs et relations.

- Candidate :
  - id: ID!
  - nom: String!
  - prenom: String!
  - email: String!
  - telephone: String
  - cv: String (URL ou base64)
- Position :
  - id: ID!

    - titre: String!
    - description: String!
    - typeContrat: String!
- Application :
    - id: ID!
    - candidat: Candidate!
    - offreDePoste: Position!
    - dateSoumission: String!
    - statut: StatusEnum!
    - commentaires: String
- **StatusEnum** (énumération des statuts de candidature) :
    - SOUMIS
    - EN_COURS
    - ACCEPTE
    - REJETE

2.3.2 Types d'input

Les **types d'input** sont utilisés pour structurer les données envoyées lors des mutations (modifications ou ajouts de données). Ils permettent une saisie organisée et validée.

- OfferInput :
    - titre: String!
    - description: String!
    - typeContrat: String!
- ApplicationStatusUpdateInput :
    - applicationId: ID!
    - newStatus: StatusEnum!
    - commentaire: String (optionnel)
- **CandidateInput** (optionnel si on prévoit d'ajouter/modifier un candidat via GraphQL) :
    - nom: String!
    - prenom: String!
    - email: String!
    - telephone: String
    - cv: String

2.3.3 Requêtes (Query)

Les requêtes permettent au client d'interroger le serveur pour récupérer des données sans les modifier.

- Position: [Position!]!
    - Retourne la liste complète des offres d'emploi disponibles.
- Position (id: ID!): Position
    - Retourne les détails d'une offre spécifique identifiée par son id.
- applicationsByCandidate(candidateId: ID!): [Application!]!
    - Retourne la liste des candidatures déposées par un candidat donné.
- application(id: ID!): Application

    - Retourne les détails d'une candidature spécifique.
- candidates(filter: CandidateFilter): [Candidate!]! (optionnel)
    - Recherche des candidats selon des critères (exemple : nom, département, etc.).

2.3.4 Mutations

Les mutations permettent de modifier les données côté serveur (ajout, mise à jour, suppression).

- submitApplication(candidateId: ID!, offerId: ID!): Application
    - Permet à un candidat de soumettre une candidature pour une offre donnée.
- updateApplicationStatus(input: ApplicationStatusUpdateInput!): Application
    - Permet de mettre à jour le statut et éventuellement le commentaire d'une candidature.
- addOffer(input: OfferInput!): JobOffer
    - Permet de créer une nouvelle offre d'emploi.
- updateOffer(id: ID!, input: OfferInput!): JobOffer
    - Permet de modifier une offre existante.
- deleteOffer(id: ID!): Boolean
    - Permet de supprimer une offre d'emploi. Retourne true si la suppression est réussie.

Phase 3 : Guide de Lancement et Tests

3.1 Installation des dépendances

Pour lancer le projet, il faut d'abord installer les dépendances nécessaires. Depuis la racine du projet, exécutez la commande suivante :

npm install

Cette commande installe toutes les librairies listées dans le fichier package.json, notamment NestJS, GraphQL, et les autres outils liés.

3.2 Lancement de l'application NestJS

Pour démarrer le serveur local NestJS, utilisez la commande suivante :

npm run start:dev

Cette commande lance l'application en mode développement avec rechargement automatique, le serveur écoute sur http://localhost:3004.

3.3 Accès au Playground GraphQL

Une fois le serveur lancé, vous pouvez accéder à l'interface interactive **GraphQL Playground** à l'adresse :

http://localhost:3004/graphql

Cet outil vous permet de tester les requêtes et mutations GraphQL, visualiser la documentation du schéma, et explorer les données en temps réel.

3.4 Exemples complets de tests

Pour garantir la bonne fonctionnalité du service web, plusieurs exemples de tests peuvent être réalisés depuis le Playground.

3.4.1 Mutations

Exemple : Soumettre une candidature

```
mutation {
  submitApplication(candidateId: "123", offerId: "456") {
    id
    statut
    dateSoumission
    candidat {
      nom
      email
    }
    offreDePoste {
      titre
    }
  }
}
```

3.4.2 Requêtes

Exemple : Récupérer toutes les offres d'emploi

```
query {
  Position {
    id
```

```
    titre
    description
    datePublication
    typeContrat
  }
}
```

3.4.3 Mutations de mise à jour

Exemple : Mettre à jour le statut d'une candidature

```
mutation {
  updateApplicationStatus(input: {
    applicationId: "789",
    newStatus: ACCEPTE,
    commentaire: "Candidat retenu pour un entretien"
  }) {
    id
    statut
    commentaires
  }
}
```

Phase 4 Présentation du Système de Suivi de Candidatures

4.1 Description générale

Le système de suivi de candidatures est une application web conçue pour faciliter la gestion des offres d'emploi et le suivi des candidatures des postulants. Il permet aux recruteurs de publier des offres, de consulter les candidatures reçues, et de gérer le processus de sélection via une interface GraphQL moderne, sécurisée et performante.

L'architecture repose sur NestJS, un framework Node.js structuré, combiné à GraphQL pour des échanges flexibles et efficaces entre le client et le serveur.

4.2 Fonctionnalités principales

- **Gestion des offres d'emploi** : création, modification, suppression et consultation des offres.

- **Soumission de candidatures** : les candidats peuvent postuler en soumettant leur profil.
- **Suivi des candidatures** : mise à jour du statut des candidatures (en attente, acceptée, refusée).
- **Recherche et filtrage** des offres et candidatures.
- **Interface interactive GraphQL** pour tester les requêtes et mutations en temps réel.

4.3 Démarrage rapide

1. Cloner le dépôt du projet.
2. Installer les dépendances avec npm install.
3. **Lancer le serveur** en mode développement : npm run start:dev.
4. **Accéder au Playground GraphQL** via http://localhost:3004/graphql.
5. **Tester les requêtes et mutations** pour explorer les fonctionnalités.

4.4 Exemple de requête GraphQL

Récupérer les candidatures avec le statut « en attente » :

```
query {
  applicationsByStatus(status: EN_ATTENTE) {
    id
    dateSoumission
    statut
    candidat {
      nom
      email
    }
    offreDePoste {
      titre
      datePublication
    }
  }
}
```

4.5 Stack technique

- **Backend** : NestJS
- **API** : GraphQL (Apollo Server)
- **Base de données** : MYSQL (via TypeORM)

- Environnement de développement : Node.js, npm
- **Gestion des versions** : Git, GitHub
- Documentation interactive : GraphQL Playground

Conclusion

Bilan du projet

Ce projet de système de suivi de candidatures a permis de mettre en place une solution moderne et efficace pour la gestion des offres d'emploi et des candidatures associées. Grâce à l'utilisation de NestJS couplé à GraphQL, l'application bénéficie d'une architecture modulaire, facilement extensible et offrant une API flexible adaptée aux besoins des recruteurs et des candidats.

Les différentes phases du projet, depuis l'analyse initiale jusqu'aux tests, ont permis de bien cadrer les besoins fonctionnels, d'assurer la qualité du code et d'optimiser les échanges entre client et serveur. L'application est fonctionnelle, intuitive et répond aux objectifs définis en phase de conception.

Perspectives d'amélioration

Plusieurs pistes d'évolution peuvent être envisagées pour enrichir le système :

- **Ajout d'une interface utilisateur front-end** complète, permettant une navigation plus intuitive sans passer par le Playground GraphQL.
- **Implémentation d'un système d'authentification et de gestion des rôles** (candidats, recruteurs, administrateurs) pour sécuriser les accès et personnaliser les fonctionnalités.
- **Notifications en temps réel** (email ou via WebSocket) pour informer les candidats de l'évolution de leur candidature.
- **Intégration d'un moteur de recherche avancé** avec filtres multiples pour faciliter la recherche des offres et candidatures.
- Extension des types de données et gestion des pièces jointes (CV, lettre de motivation, portfolio).
- **Optimisation de la performance** et montée en charge pour gérer un grand nombre d'utilisateurs simultanés.