Rapport JobShop

Enguerran Best Willem Nicolas Paula Sanchez
Akina Renard Skander Sayadi

May 2022

1 Introduction

Notre application permet de mettre en relation les étudiants et les recruteurs IT sur la thématique des recherches de stage ou de premier emploi. Le but est de proposer une expérience complète à chaque partie. Le candidat doit répondre à un long questionnaire pour trouver le job idéal qui lui correspond. Le but est qu'il ne se connecte qu'une seule fois sur le site et ensuite, les annonces pertinentes lui sont envoyées par mail (fonctionnalité non implémentée pour l'instant). Le site possède un algorithme de matching qui fait des propositions de correspondance. D'un autre côté, le recruteur peut renseigner des fiches de postes et tenir à jour ses tableaux de suivi et voir l'avancement des processus de recrutement qu'il mène. D'autre part, afin de le rendre attractif, nous avons prévu un espace "blog" avec du contenu pertinent pour chaque partie.

2 Présentation technique

2.1 Frontend

Nous avons utilisé la technologie React. Nous nous sommes appuyés sur Redux pour la gestion du dataStore qui permet de stocker les données dans le cache du navigateur client. Évidemment, nous avons dû compléter le module Redux, pour le rendre persistant aux refresh web. L'architecture du frontend est classique et est disponible sur le dépôt Github. Nous avons décidé d'utiliser SASS pour déployer des animations plus efficacement et de façon plus pratique.

2.2 Backend

Nous avons décidé d'organiser les entités ainsi:

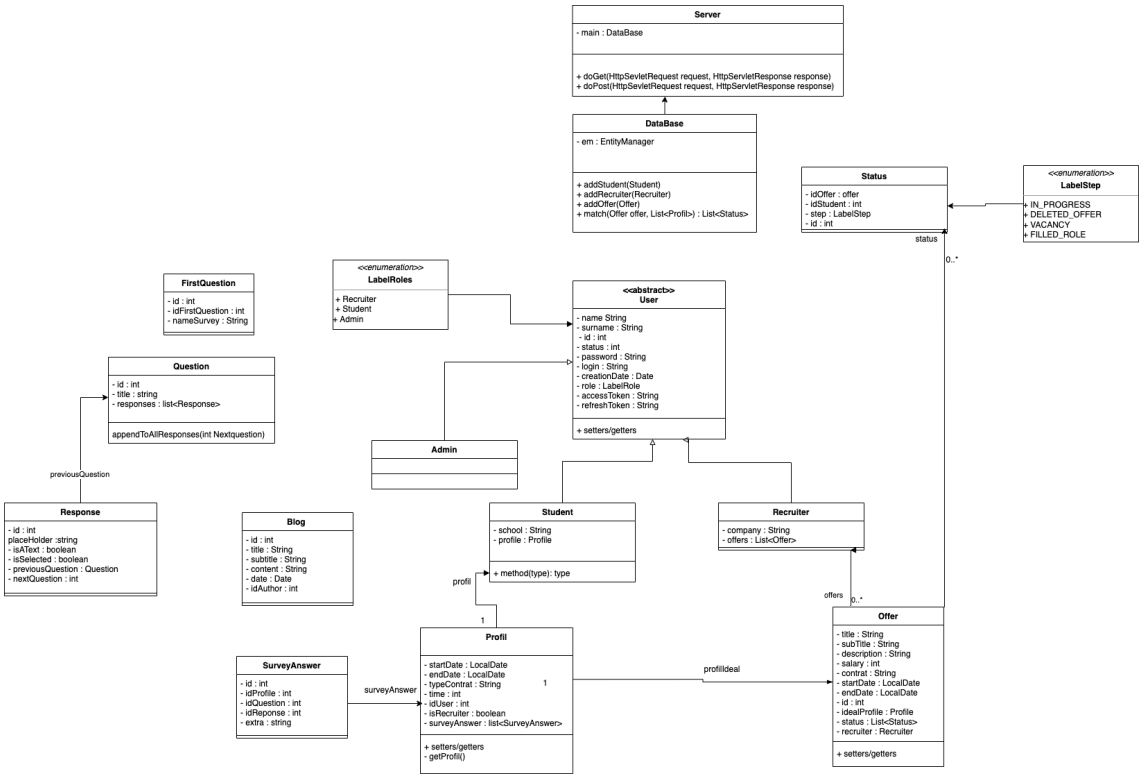

Figure 1: diagramme de classe UML de notre architecture Java

(pour plus de détails voir notre gitHub)

Les entités de notre EntityManager sont : Blog, FirstQuestion, Offer, Profile, Question, Response, Status, SurveyAnswer, User.

Pour accéder aux services proposés, chaque utilisateur peut créer un compte. L'identifiant doit être une adresse mail, et le mot de passe est crypté en utilisant BCrypt. Un user peut être un administrateur, un recruteur ou un étudiant. Un administrateur peut créer un questionnaire. L'objet java créé lors de la création d'un nouveau questionnaire est FirstQuestion contenant un titre ("current" si c'est le questionnaire courant) et l'identifiant de la première question. Afin de pouvoir ajouter des questions facilement mais aussi ajouter des questions uniquement après certaines réponses, nous avons décidé d'implémenter le questionnaire sous forme d'arbre. Prenons un exemple :

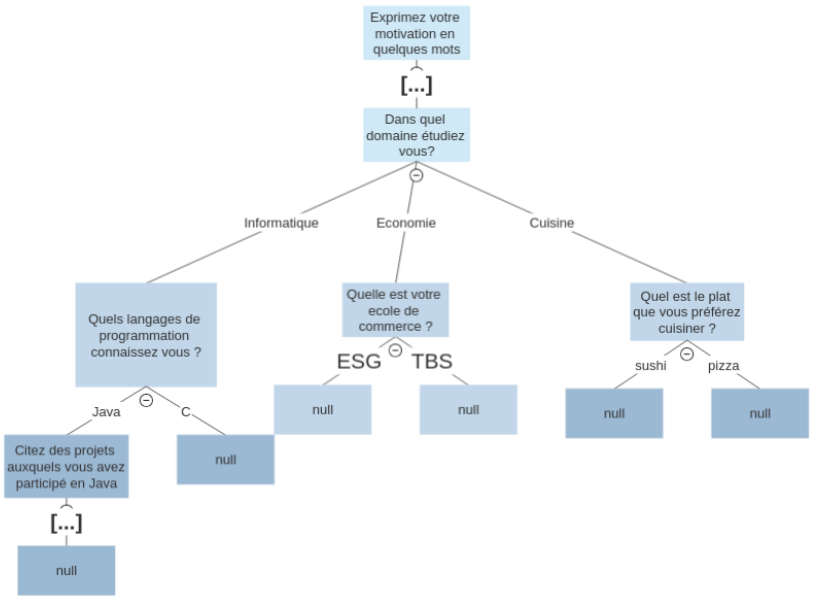

Figure 2: Exemple de questionnaire (les cases "null" correspondent à un idNextQuestion égal à 0, donc à aucune question de la base de données)

L'objet Question a une liste de réponses, et l'objet Response a l'identifiant de la prochaine question, éventuellement null si c'est la dernière réponse. L'objet réponse contient un texte à cliquer (dans cet exemple : Informatique, Economie...) ou un texte vide qui sera rempli par l'utilisateur.
Lorsqu'un étudiant crée un compte, il répond à un questionnaire de stage/emploi. À ce stade, une instance de la classe Profile est créée avec les données du profil de l'étudiant. Le profil est associé à la classe Student qui l'a comme attribut. Si un recruteur crée une offre d'emploi, une instance de la classe Offer est créée, ainsi qu'une instance de Profile contenant les données du profil du candidat idéal pour le poste. Les deux instances de Profile sont comparées afin de choisir celle qui se rapproche le plus du profil idéal pour l'offre d'emploi avec la méthode match.

Un utilisateur peut créer un blog. Afin d'ajouter un blog, on associe l'identifiant de l'auteur au blog. Ainsi on peut afficher tous les blogs d'un utilisateur en faisant la requête adéquate à la base de données.

2.3 Connexion avec le frontend

Afin de communiquer avec le frontend, nous avons créé plusieurs classes permettant de gérer nos données. Lors de requêtes, on récupère au niveau de

notre HttpServlet une chaîne de caractère en json que l'on va transformer en objet. On ajoutera dans la réponse un objet traduit en json. Par exemple pour l'action "login", le frontend envoit un json contenant un mail et un mot de passe au backend : {login:"adress@mail.com", password:"password"}. Le code métier effectue la recherche de l'entité attachée à l'adresse mail dans la base de données puis vérifie que le mot de passe est correct. Si ces étapes sont réussies, on ajoute dans la réponse un json correspondant à l'entité trouvée dans la base de données: {name:"willem", surname:"dafoe", id:3, role:"ADMIN", login:"adress@mail.com", password:"password"}.
De plus, nous avons fait en sorte qu'il soit très simple d'ajouter des informations supplémentaires dans la classe User. Par exemple pour ajouter l'attribut school (String) à Student, il suffit d'ajouter l'attribut, les méthodes set et get, et la mise à jour se fait automatiquement du côté du frontend.

3 Améliorations possibles

- Ajout des tokens : les fonctions de création de refreshToken (valide un jour) et accessToken (valide une heure) sont déjà implémentées et testées en local, mais comme nous avons utilisé la librairie extérieure JWT nous ne pouvons pas les utiliser avec la base de données.

- Possibilité de créer plusieurs questionnaires: On peut avoir plusieurs formulaires (First reponse) créés par l'administrateur et il peut changer le formulaire courant. Cela signifie que le formulaire principal à remplir par l'étudiant peut changer.

- Gérer l'effacement simultané du cache et d'un refresh de la page, qui nous ferait perdre le tracking et l'identification des personnes.