Gestionnaire de CV - Rapport du projet d'XML

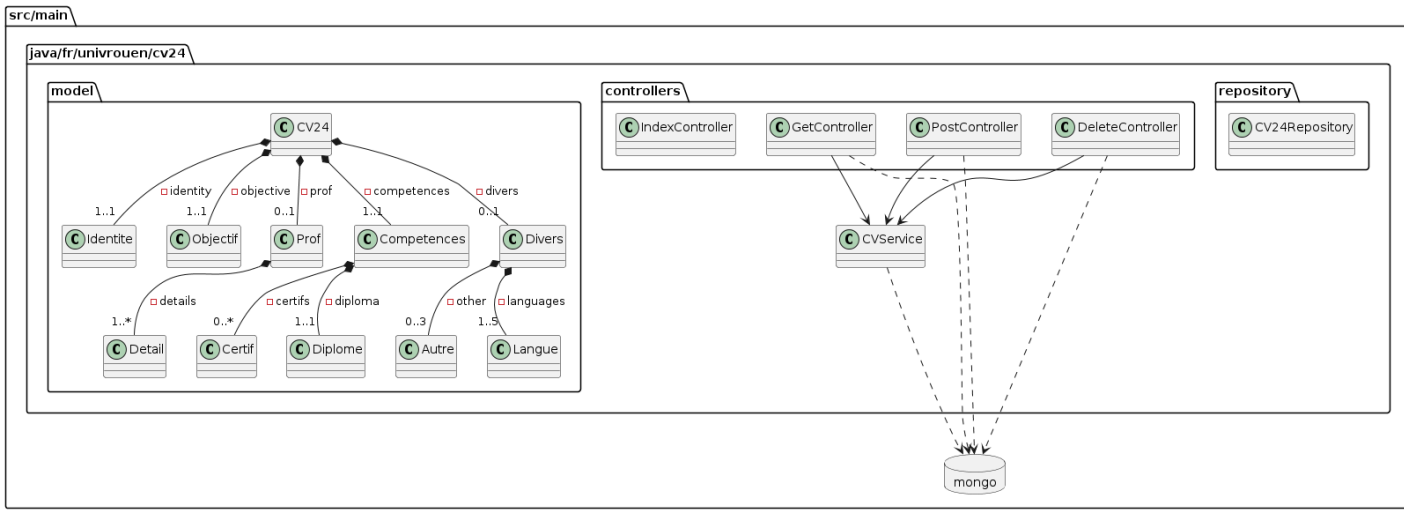
MONTAGNE Erwann
VAN LIEDEKERKE Florian

22 mai 2024

Table des matières

1 Architecture globale

Nous allons commencer par présenter l'architecture générale de notre application web. Voici son schéma :



1.1 Architecture prévue

A l'origine, nous avions prévu d'avoir une API Rest reposant sur le modèle MVC. Une interaction utilisateur agirait ainsi sur l'application :

1. l'utilisateur appelle une route ;
2. le contrôleur récupère la requête et délègue la partie métier en appelant le service ;
3. le service, qui sert à manipuler le code métier, va manipuler les entités et repository en effectuant une succession d'actions dessus ;
4. une fois le traitement effectué, le service renvoie son résultat au contrôleur ;
5. ce dernier traite le résultat et renvoie le bon code de retour avec le bon format.

Notre application serait donc divisée en plusieurs couches. Tout d'abord nous avons le contrôleur qui traite les requêtes HTTP et délègue au service. Le service, lui, est une classe orchestrant toute la logique métier. Enfin nous aurions nos entités qui auraient été "mappées" avec les données dans la base de données ; ainsi qu'un repository permettant d'effectuer des requêtes simples sur la BDD, comme retrouver un CV par rapport à un identifiant.

1.2 Différences avec l'architecture prévue

Aujourd'hui, notre application comporte bien toutes les notions décrites plus haut, nous avons juste rencontré des difficultés par rapport au mapping entre nos classes Java et nos entités en BDD.

Initialement, nous avions rencontré des difficultés pour mapper correctement nos entités et générer nos classes depuis un fichier XSD. Nous avons finalement réussi mais nous nous sommes retrouvés face à un problème de MongoDB (le SGBD que nous avions choisi, détaillé dans la sous-section 3.1).

En effet, nous avions bien nos classes générées automatiquement et tout fonctionnait correctement, cependant, la spécificité du XSD ainsi que le fonctionnement de MongoDB ne nous ont pas permis de réaliser notre application de la manière initialement prévue.

Par exemple, d'après le XSD un CV peut comporter plusieurs langues (lv), la génération automatique nous a donc bien généré un attribut de type List. Or, dans MongoDB, s'il n'y a qu'une seule langue l'attribut

est de type lv et non plus de type List (ou Array). Cela posait donc des soucis pour le typage. C'est pourquoi nous n'avons pas pu manipuler vraiment les entités comme nous le voulions.

Cependant, malgré ce regret, notre application est totalement fonctionnelle et chacune des demandes a été remplie. L'insertion et suppression de CV se sont faites très facilement, l'affichage de CV au format HTML également. C'était surtout la transformation d'un document Mongo vers du XML qui était plus fastidieuse (cela aurait été plus simple avec de simples getters sur des entités mappées mais non possible à cause de la limitation décrite plus haut). Mais comme dit précédemment, l'application reste tout à fait fonctionnelle et l'expérience utilisateur n'en est pas du tout affectée.

2 Rôle des différentes classes

Avant de détailler le rôle des classes, voici les différentes ressources du projet, que nous pouvons trouver dans le répertoire resources :

- un répertoire static contenant le css de la page d'accueil ainsi que du CSS des CV transformés via XSLT. Il contient aussi le logo de l'université ;
- un répertoire templates contenant un fichier de vue, statique. Ce fichier est la page HTML de la page d'accueil ;
- un répertoire xml contenant 3 exemples de CV préalablement insérés dans la BDD, le XSD de validation, 2 xslt pour transformer les fichiers XML en flux HTML. Le premier xslt est pour le détail d'un CV, le second est pour le résumé de tous les CV ;
- le fichier application.properties contenant les propriétés du projet.

Il est donc clair que nous avons réutilisé les différents éléments mis en oeuvre lors des TP, à travers la XSD de validation pour l'insertion de CV mais aussi les XSLT de transformation, utilisant XPATH.

La page d'aide (/help) a été générée via un Swagger et peaufinée à l'aide d'annotations dans les contrôleurs.

Nous allons maintenant dresser un bilan de chaque classe pour vous spécifier le rôle qu'elle avait au sein de notre application :

- un répertoire controllers contenant
  - DeleteController qui gère les requêtes HTTP de type DELETE ;
  - GetController qui gère les requêtes HTTP de type GET ;
  - PostController qui gère les requêtes HTTP de type POST ;
  - IndexController qui gère la page d'accueil.
- un répertoire model contenant chacune des classes générées automatiquement. Chaque classe représente l'élément éponyme dans la XSD ;
- un répertoire repository contenant une classe CV24Repository permettant d'effectuer des requêtes en BDD ;
- une classe CVService représentant le service de l'application et permettant la manipulation du code métier (comme l'insertion, la suppression de CV ou encore la génération du flux XML / HTML) ;
- CV24Application représentant le point d'entrée de l'application ;
- ServletInitializer permettant d'enregistrer notre point d'entrée comme une classe de configuration de notre application.

3 Choix technologiques

3.1 Choix de MongoDB

Pour plusieurs raisons nous nous sommes tournés vers MongoDB comme SGBD pour la persistance de données. Tout d'abord, chaque CV peut différer à cause de la nature du XSD. Par exemple, un CV peut comporter une section "Expériences professionnelles" alors qu'un autre non. Le fait de ne pas avoir une structure immuable et commune à tous nous a fait nous orienter vers le NoSQL.

De plus, dans Mongo, ce ne sont pas des tables mais des documents sous format JSON qui sont stockés. Pour l'insertion, nous n'avions qu'à, après vérification des pré-conditions, convertir le XML fourni en un JSON puis à l'insérer dans la BDD directement.

3.2 Base de données en Cloud

Nous n'avons pas trouvé d'add-on Mongo sur CleverCloud qui soit gratuit. Nous nous sommes donc tournés vers un cluster Mongo en cloud (https ://cloud.mongodb.com). Notre base de données est donc hébergée sur le cloud, cela n'intrave en aucun cas la qualité du projet. Cela nous permet également de pouvoir déployer notre application gratuitement.

4 Liens utiles

Lien du GitHub : https ://github.com/Zounkla/cv24v1

Lien de l'application déployée : https ://app-37bd1855-107c-4fe5-a6f3-d1aa0e58995d.cleverapps.io ou https ://cv24-vanliedekerke.cleverapps.io.cleverapps.io