Compte rendue XML

18 mai 2024

Date	18 mai 2024
Rédigé par	OLIVIER Thomas

Table des matières

1 Introduction

Ce projet consiste au développement d'un service RESTful destiné à la gestion de documents XML de type "CV24", conformes à la structure définie dans le fichier XSD établi lors du TP1. L'objectif principal de ce projet réside dans la mise en place d'un mécanisme de persistance permettant le stockage des CV dans une base de données. Le travail demandé pour ce projet consiste à mettre en place plusieurs endpoints permettant la gestion et la visualisation des données de notre service. Pour réaliser ce projet, j'ai utilisé une application Spring Boot, comme vu lors des travaux pratiques, que j'ai enrichie en combinant différentes bibliothèques ce qui m'a permis de répondre au mieux à toutes les exigences spécifiées.

2 Architecture général

La première partie de notre architecture concerne le modèle de données de l'application. Elle se trouve dans le package model et a pour objectif de définir des classes Java qui représenteront les différentes sections du CV.

Le model d'un CV dans notre application correspond au diagramme suivant :

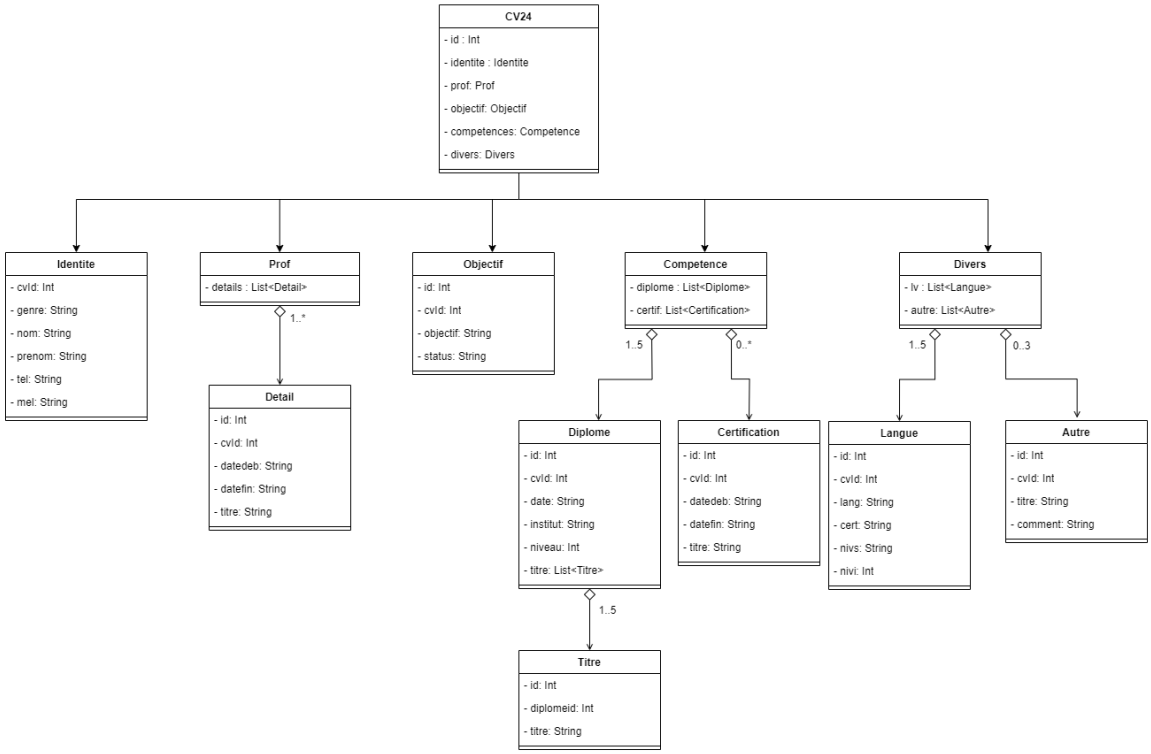

FIGURE 1 – Schéma du model de données

Chaque classe contient également tous les getters et setters nécessaires pour chaque attribut. Cependant, pour des raisons de lisibilité, ils n'ont pas représentés sur le schéma.

Dans ce package, nous avons une classe "ListCVResume" et une classe "CVResume" dont l'objectif est de représenter un CV sous une forme résumée.

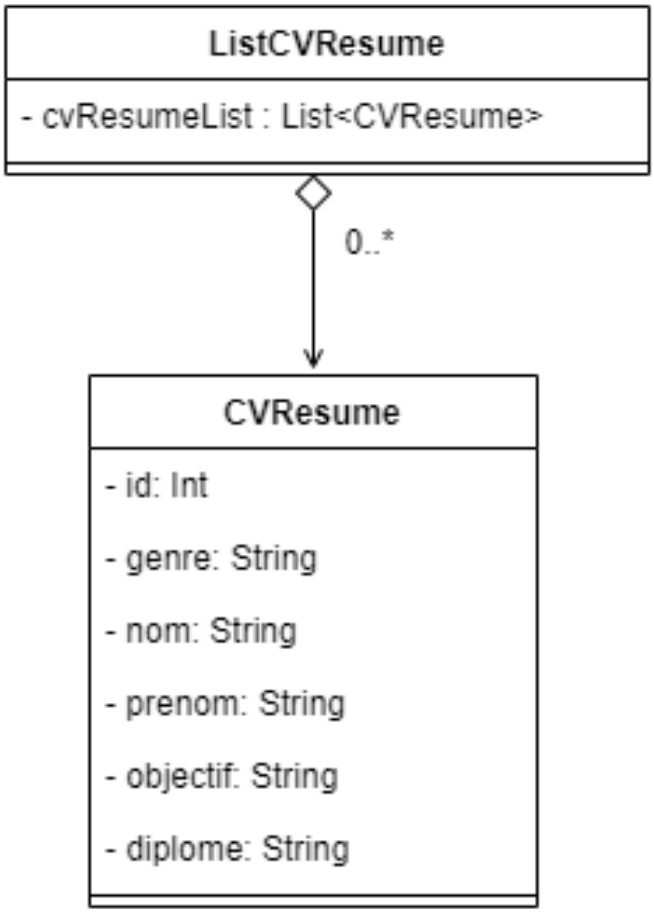

FIGURE 2 – Schéma du model de données pour les cv résumer

La dernière classe de ce package est "IdentitePK", qui permet de définir la clé primaire pour la classe "Identite" et, par conséquent, pour le CV lui-même.

Une fois le modèle défini, nous utilisons JpaRepository, une interface qui simplifie considérablement la gestion des entités dans les applications Spring Data JPA en fournissant une API pour les opérations de persistance. Dans notre cas, nous avons défini une interface repository pour chaque classe du modèle, ce qui

nous permet de gérer les opérations de base de données liées à chaque entité et de définir des opérations personnalisées pour certaines entités.

Chaque interface auras des fonctionnalité définit par JPA :

- `save(S entity)` : Enregistrer ou mettre à jour une entité.
- `findById(ID id)` : Trouver une entité par son identifiant.
- `findAll()` : Récupérer toutes les entités.
- `deleteById(ID id)` : Supprimer une entité par son identifiant.
- `delete(S entity)` : Supprimer une entité donnée.
- `findAll(Sort sort)` : Récupérer toutes les entités triées selon un critère donné.

Certaine interface on aussi des méthodes personnalisé pour récupérer les données souhaité :

- `CVRepository` : `getCV24ById(int id)` qui récupère un CV en fonction de l'ID donné.
- `AutreRepository` : `findAllByCvId(int id)` qui récupère toutes les entités `Autre` qui correspondent à l'ID.
- `CertifRepository` : `findAllByCvId(int id)` qui récupère toutes les entités `Certification` qui correspondent à l'ID.
- `DetailRepository` : `findAllByCvId(int id)` qui récupère toutes les entités `Detail` qui correspondent à l'ID.
- `DiplomeRepository` : `findAllByCvId(int id)` qui récupère toutes les entités `Diplome` qui correspondent à l'ID.
- `IdentiteRepository` : `findByCvId(int id)` qui récupère l'entité `Identite` en fonction de l'ID du CV et `findByGenreAndNomAndPrenomAndTel(String genre, String nom, String prenom, String tel)` qui récupère l'entité `Identite` en fonction des quatre paramètres pour vérifier qu'un CV n'existe pas déjà avec cette clé.
- `LangueRepository` : `findAllByCvId(int id)` qui récupère toutes les entités `Langue` qui correspondent à l'ID.
- `ObjectifRepository` : `findByCvId(int id)` qui récupère l'entité `Objectif` qui correspond à l'ID.
- `TitreRepository` : `findAllByDiplomeId(int id)` qui récupère toutes les entités `Titre` qui correspondent à l'ID du diplôme.

Le package services définit des classes représentant des services dédiés à la gestion des entités dans l'application. Les services utilisent les interfaces définies précédemment afin de gérer la persistance et la récupération des données. Le package contient les classes suivantes :

- `CVService` : Gère les opérations liées aux CV, telles que la sauvegarde, la suppression et la récupération de CV complets ou résumés.
- `IdentiteService` : Gère les opérations liées à l'identité, y compris la vérification de l'existence d'une identité, la sauvegarde et la récupération des informations d'identité.
- `ObjectifService` : Gère les opérations liées aux objectifs professionnels dans les CV, y compris la sauvegarde, la construction du format résumer et la récupération des objectifs.
- `ProfService` : Gère les opérations liées aux informations professionnelles dans les CV, y compris la sauvegarde et la récupération des expériences professionnelles.
- `CompetenceService` : Gère les opérations liées aux compétences dans les CV, y compris la sauvegarde et la récupération des compétences.
- `DiversService` : Gère les opérations liées aux informations diverses dans les CV, y compris la sauvegarde et la récupération des informations diverses.

Notre application repose sur le format XML, que ce soit pour ajouter des CV à notre système ou pour récupérer des CV. Pour cela, nous avons défini un package xml qui a pour objectif de regrouper toutes les opérations liées au XML.

La première classe est la classe XMLValidator, qui contient une seule méthode : validateXMLAgainstXSD(String xmlContent, String xsdFilePath). Son objectif est de valider qu'un fichier XML est conforme à un schéma XSD donné. Le premier paramètre correspond au contenu du fichier XML et le second paramètre, xsdFilePath, correspond au chemin où se trouve le schéma XSD.

La deuxième classe est la classe XMLTransform fournit des méthodes pour transformer des objets Java en XML et vice versa. Elle utilise l'API de JAXB pour la sérialisation et la désérialisation des objets Java en XML et inversement. Cettte clase fournie trois méthodes :

- `ConvertXmlToCv(String xmlContent)` : trasnforme un fichir xml représenter par le paramètre xmlContent en un objet java CV24 qui corresond à la base de notre modèle de données.
- `convertCVtoXML(CV24 cv)` : Transforme un CV24 en un String qui représente le cv en xml.
- `convertCVResumeToXML(ListCVResume cv)` : Transforme une Lisye de CV résumer en un fichier XML.

La dernière classe est la classe XMLBuildMessage qui a pour objectif de construire différent message au fromat XML correspondant par exemple au message d'erreur de l'application ou au message de validation des requètes.

Le projet contient un package exception qui lui contient une seul classe qui est "GlobalExceptionHandler". Cette classe centralise la gestion des exceptions dans l'application en interceptant toutes les exceptions qui ne sont pas gérées spécifiquement ailleurs dans l'application. Une fois l'exception intercepter, elle assure que toutes les erreurs sont traitées de manière cohérente et renvoient des réponses claires et structurées aux clients de l'API.

Pour finir, nous avons le package controllers qui définit les classes correspondant à des contrôleurs en fonction de l'opération demandée. Dans notre cas, nous avons trois types de contrôleurs :

- `deleteController` : c'est un contrôleur Spring MVC pour gérer les requêtes DELETE liées à l'application. Il contient une méthode pour supprimer un CV en fonction de l'ID.
- `postController` : c'est un contrôleur Spring MVC pour gérer les requêtes POST liées à l'application. Il contient une méthode pour insérer un CV.
- `getController` : c'est un contrôleur Spring MVC pour gérer les requêtes GET liées à l'application. Il permet de retourner des pages HTML ou des réponses XML en fonction de la requête. On a par exemple la méthode pour renvoyer la page d'accueil, la page d'aide ou encore la liste résumée de tous les CV.

3 Endpoint

Le service propose une gamme de requête diverses :

— **Page d'accueil** - GET : Affiche la page d'accueil (/).
— **Aide** - GET : Affiche les informations d'aide (/**help**).
— **Liste des CV (XML)** - GET : Affiche la liste des CV au format XML (/**cv24**/**resume**/**xml**).
— **Détail d'un CV (XML)** - GET : Affiche le contenu complet d'un CV au format XML (/**cv24**/**xml ?id=**<**id**>).
— **Liste des CV (HTML)** - GET : Affiche la liste des CV au format HTML (/**cv24**/**resume**).
— **Détail d'un CV (HTML)** - GET : Affiche le contenu complet d'un CV au format HTML (/**cv24**/**html ?id=**<**id**>).
— **Ajout d'un CV dans la base** - POST : Ajoute un CV dans la base de données (/**cv24**/**insert**).
— **Suppression d'un CV** - DELETE : Supprime un CV de la base de données (/**cv24**/**delete**).

4 Accès au projet

4.1 Accès à la version déployer

Pour accéder à l'application vous devez utiliser l'URL suivante : projet-olivier.cleverapps.io .

4.2 Accès à la version local

Pour utiliser le projet en local, vous devrez suivre les étapes suivnate :

- `Récupérer le projet` : Utiliser la commande git clone https ://github.com/thomharry76320/Projet-XML.git
- `Installation des dépendances` : Utiliser la commande mvn clean install
- `Lancement de l'application` : Utiliser la commande mvn spring-boot :run

Si vous souhaiter modifier la base de données vous pouvez utiliser une autre base de données postgreSQL en modifiant l'url, le username et le password de votre base de données dans le fichier application.properites. Ce fichier se situe dans le dossier src/main/java/fr.univrouen.cv24/resources

5 Information complémentaire

5.1 Dossiers resources

Le dossier ressources situe dans src/main/java/fr.univrouen.cv24/resources contient :

- `Dossier database` : Contient le script de cération de la base de données.
- `Dossier Javadoc` : Contient la javadoc complète du projet (il y a un problème d'encodage que j'ai pas réussie resoudre)
- `Dossier postman` : Contient la collection postman des test au format JSON.
- `Dossier static` : Contient les fichiers css du projet.
- `Dossier templates` : Contient les fichiers html du projet.
- `Autres` : Contient le fichier xsd qui valide les xml, les deux exemples de cv en XML et le contre rendue du projet.

5.2 Modification du sujet

Une modification par rapport au sujet a été apportée pour garder une cohérence dans le projet. Le fichier XSD indiquait initialement que le numéro de téléphone était facultatif, mais je l'ai rendu obligatoire car le sujet indique qu'un CV existe déjà lorsqu'il porte le même genre, nom, prénom et numéro de téléphone. Cela rend le numéro de téléphone obligatoire pour gérer la clé primaire dans la base de données.

5.3 Templates HTML

Pour la création des templates HTML, j'ai utilisé Thymeleaf en raison de sa facilité d'utilisation et de son intégration transparente avec les projets Spring.

5.4 Choix de la base de données

Le choix de la base de données a été réalisé en fonction de plusieurs critères. Le premier est la disponibilité de la base de données sur Clever Cloud. En effet, Clever Cloud propose trois types de bases de données : PostgreSQL, MySQL et MongoDB. Après une étude approfondie du projet, j'ai constaté qu'une base de données relationnelle serait plus efficace qu'une base de données orientée objet qui aurait stocké directement les fichiers XML. Cette solution aurait grandement simplifié l'insertion des données, mais aurait limité les performances et complexifié énormément la gestion des données lors de la récupération. Le choix se limitait donc à MySQL et PostgreSQL. Le choix final s'est porté sur PostgreSQL car cette base de données offre de meilleures performances pour les requêtes et une fiabilité supérieure à MySQL.

5.5 Test

Le projet contient aussi un fichier de test Junit qui a pour objectif de valider le fonctionnement de la validation des fichiers xml. La classe de test se situe dans src/test/java/univrouen.cv24

5.6 Remarque

Il semble y avoir un problème avec le nombre de connexions à la base de données. En effet, une seule connexion simultanée semble être autorisée sur la base de données à cause des limitations imposées par Clever Cloud. Cela pourrait être une piste à explorer si un problème avec la base de données est rencontré.