Master 1 GIL - XML et Services web
Projet : Service Rest

04 avril 2024

Version	1
Date	29 avril 2024
Rédigé par	KHABOURI Izana

Mises à jour du document

Version	Date	Modification réalisée
1	23 mai 2024	Création du document

Table des matières

1 Objectifs **4**

2 Solution architecturale **4**
2.1 Rôle de chaque classe 5
2.1.1 Le contrôleur 5
2.1.2 Les entités 5
2.1.3 Les templates 5

3 Le service REST **6**

4 Déploiement **6**
4.1 En local 6

5 Postman **7**

6 Choix technologiques **9**

7 Navigation **9**

8 Difficultés rencontrées **13**

9 Améliorations possibles **13**

10 Le lien github **13**

1 Objectifs

L'objectif étant de réaliser un service Restfull permettant de gérer les CV et de réaliser différentes requêtes qui ont pour conséquence d'ajouter un CV, de supprimer ce dernier, de visualiser un CV, et également de visualiser une liste de CV.

2 Solution architecturale

J'ai décidé de séparé les éléments de la manière suivante :

- Les entités, qui représente les différents éléments d'un CV telles que l'identité d'une personne **Identite**, l'objectif du CV **Objectif**, les professions réalisées **Prof**, les diplomes obtenus **Diplome**, les certificats obtenus **Certif**, etc... On définit seulement le CV avec les élémments précédemment cités.
- Le lien avec la base de donnée qui est nommée repository. Ici, il est utilisé uniquement en cas d'utilisation de la base de donnée tel que la recherche de différents CV, l'ajout d'un nouveau CV et enfin la suppression d'un CV existant. Une modification d'une ressource ou un appel d'une de ces ressources entraînent l'utilisation de repository.
- Le contrôleur permettant de réaliser l'objectif principal qui est de réaliser un service REST permettant de gérer la notion de CV sous forme d'entité. Chacune de ces méthodes sont associé à une route et à un type de requête tels que :
  - POST permettant l'ajout d'une ressource
  - GET permettant la récupération d'une ressource
  - PUT permettant la modification d'une ressource
  - DELETE permettant la suppression d'une ressource

  Cette partie gère toutes les routes et leur requête respectives.
- Les énumérations sont mis dans un paquetage à part afin de ne pas les perdre dans les entités. Ils définissent toutes les valeurs dont il n'existe que des valeurs finies telles que :
  - CertType définissant toutes les valeurs liées à la certification d'une langue. On a donc MAT, CLES et TOEIC.
  - GenreType définissant toutes les valeurs liées au genre d'une personne. On a donc M. et Mme.
  - NivsType définissant toutes les valeurs liées au niveau d'aptitude à une langue. On a donc A1 à C2.
  - StatutType définissant toutes les valeurs liées au type d'objectif de CV. On a donc emploi et stage.
- Le service sert ici à la manipulation du CV. En d'autres termes il représente la partie modèle de ce projet de gestion de CV. Le contrôleur appelle ce service pour la manipulation de ces ressources en fonction des requêtes.
- La partie templates sert à donner un affichage en fonction de l'entité donné. L'affichage dépend donc du retour souhaité, j'ai donc réparti l'affichage selon la donnée souhaitée, et cela en fonction du format choisi. Si l'on souhaite affiché l'objectif en format XML, j'appellerais donc la méthode String getInformationOnObjective(Objectif objectif) mais depuis la classe **XMLFormat**.
- La partie utilitaire sert pour la conversion d'un document XML en un CV, la validation d'un document XML. Il sert également pour la conversion des dates. Etant donné qu'il ne s'agit pas là des informations les plus importantes de ce projet, ils se trouvent donc dans ce paquetage.

2.1 Rôle de chaque classe

2.1.1 Le contrôleur

Nom de la classe	Rôles
CvController	Gère les différentes actions sur les CV
HomeController	Gère les différentes actions sur les pages d'accueils
HelpController	Gère les différentes actions sur les pages d'aide

2.1.2 Les entités

Nom de la classe	Rôles
Autre	Représente les informations complémentaires
Certif	Représente les certificats obtenus
Competences	Représente la liste de compétences acquises (Certif, Diplome)
Cv	Représente le CV
Detail	Représente les informations d'une profession
Diplome	Représente le diplome
Divers	Représente la liste des informations complètaires (Autre, Lv)
Identite	Représente la propriétaire du CV
Lv	Représente la langue
Objectif	Représente l'objectif du CV
Prof	Représente les professions avec leurs informations

2.1.3 Les templates

Nom de la classe	Rôles
HTMLFormat	Décrit un document HTML
XMLFormat	Décrit un document XML

3 Le service REST

Le service, se trouvant dans la partie contrôleur de ce projet, est séparé en trois fichiers qui sont donc :
- CvController permettant de faire manipuler les CV
- HelpController permettant d'afficher la page d'aide
- HomeController permettant d'afficher la page d'accueil

J'ai réalisé pour chaque requête demandée, une méthode décrivant l'action résultante de la requête. J'ajoute également un chemin qui est donc associé à la requête. J'ajoute également un format de retour qu'il s'agisse un retour sous format XML ou HTML. Pour donner un exemple de méthode, prenons la méthode **getAllCVInXMLFormat()** qui permet de retourner, sous format XML, la liste des CV présent dans la base de données. On remarque les éléments cités auparavant :
- GetMapping représente le type de requête, ici on réalise une requête GET. Effectivement nous récupérons une liste de CV issues de la base de données.
- La route est ici resume/xml, c'est sur ce chemin qu'on aura une liste de CV retournées sous un format XML.
- application/xml représente le format de retour afin qu'il soit correctement affiché.
- Le corps de la méthode manipule donc avec le service afin de récupérer la liste des CV, pour permettre l'affichage de ces derniers.

Pour rappel, toutes les requêtes obligatoires sont implantées ainsi que la requête bonus sur la recherche de CVs.

```
@GetMapping(value = "resume/xml", produces = "application/xml")
public ResponseEntity<String> getAllCVInXMLFormat() {
    Page p = new XMLFormat();
    List<Cv> cvs = this.cvService.getAllCvs();

    return ResponseEntity.status(HttpStatus.OK.value()).body(p.getAllCv(cvs));
}
```

4 Déploiement

4.1 En local

Pour ce faire, le fichier **application.properties** doit avoir le contenu suivant

```
spring.application.name=cv24
spring.datasource.url=jdbc:mysql://localhost:3307/cv24
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Lancer la commande suivante dans le terminal depuis le répertoire src/main/resources

```
docker compose up -d
```

Recopie du contenu du fichier **bdd.sql** qui se trouve dans le répertoire /src/main/resources/static/sql dans l'administration de bases de données se trouvant dans l'application docker.

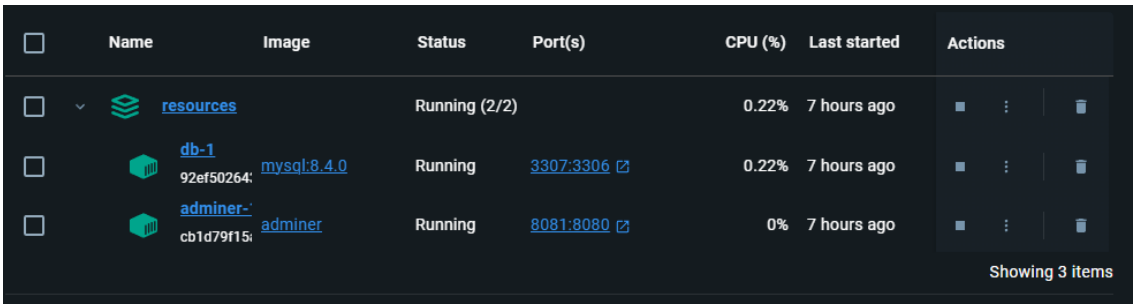

FIGURE 1 – Le conteneur docker pour la base de données

Nous cliquons 8081 :8080 pour voir une page qui représente la page d'accueil de MySQL.

— user : root

— password : root

Ensuite nous cliquons sur **Requête SQL** permettant de voir un espace ou l'on peut saisir nos requêtes. C'est ici qu'on colle notre contenu. Ensuite nous validons, la base de donnée est maintenant prête à l'utilisation.

Et enfin nous pouvons lancer l'application, elle est maintenant opérationnelle.

5 Postman

Plusieurs requêtes sont lancées depuis l'application Postman permettant de vérifier le retour. Afin de voir si le résultat retourné est bien celui souhaité. On retrouve ici dix commandes GET, quatre commandes POST et deux commandes DELETE. J'ai essayé de faire autant de requêtes afin de vérifier toutes les sorties possibles d'une seule requête. J'ai donc essayé de couvrir au mieux les sorties possibles de ces requêtes.

Afin de tester au mieux ces requêtes, le mieux étant de commencer par les requêtes POST provoquant ainsi la création de deux CV un avec toutes les informations même ceux facultatifs et le second seulement les informations facultatives. Ces fichiers se trouvent dans le répertoire static de la partie ressources de l'architecture.

Concernant les variables globales, je n'ai créé qu'une seule pour gérer le lien du serveur.

FIGURE 2 – Les requêtes depuis Postman

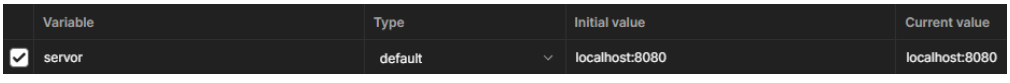

FIGURE 3 – Les variables globales

6 Choix technologiques

Le projet a été réalisé avec le framework Spring Boot afin de réaliser un service Restfull. Cela permet de bien spécifier chaque route à une requête spécifique.

La base de données étant au départ MariaDB, a été changé en MySQL car, en effet, CleverCloud ne propose pas MariaDB.

7 Navigation

Il y a trois type de pages sur la plateforme, qui sont donc :

— La page d'accueil permettant d'afficher mon nom, ainsi que le nom du projet. On y voit également deux liens. Un pour accéder à la liste de CV disponibles, la seconde la liste de requêtes possibles.

— La page présentant la liste des CVs présente pour chaque CV les informations suivantes : l'identité de la personne à qui appartient le CV, le poste recherché donc son objectif, le diplôme le plus haut ainsi que son numéro d'identifiant unique. Cet identifiant est cliquable, il permet d'accéder aux détails du CV cliqué. On a également la possibilité de retourner en arrière ou revenir directement sur la page d'accueil.

— La page d'accueil ou l'on voit absolument toutes les informations du CV, je tenais également à préciser que je n'ai cité que ceux disponibles. On peut également retourner en arrière ou même revenir directement sur la page d'accueil depuis cette page.

— La page d'aide a été grandement inspirée de ceux généré par les documentations Swagger, j'ai également repris le même code couleur que ce dernier. Nous pouvons aussi revenir sur la page d'accueil depuis cette page.

Remarque, quand je précise qu'en effet on peut revenir sur la page d'acceuil, il s'agit d'un lien le précisant en bas de page.

Voici les captures d'écran présentant l'application de manière visuelle.

FIGURE 4 – La page d'accueil

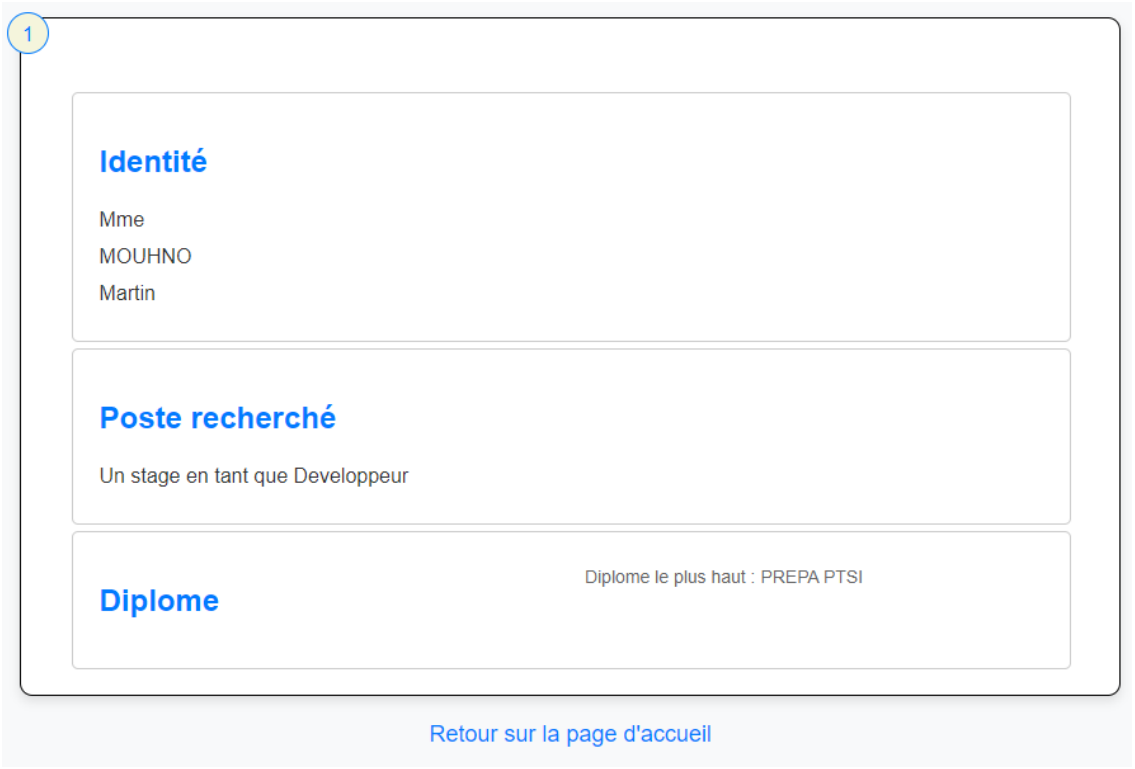

FIGURE 5 – Un CV affiché depuis la liste des CV

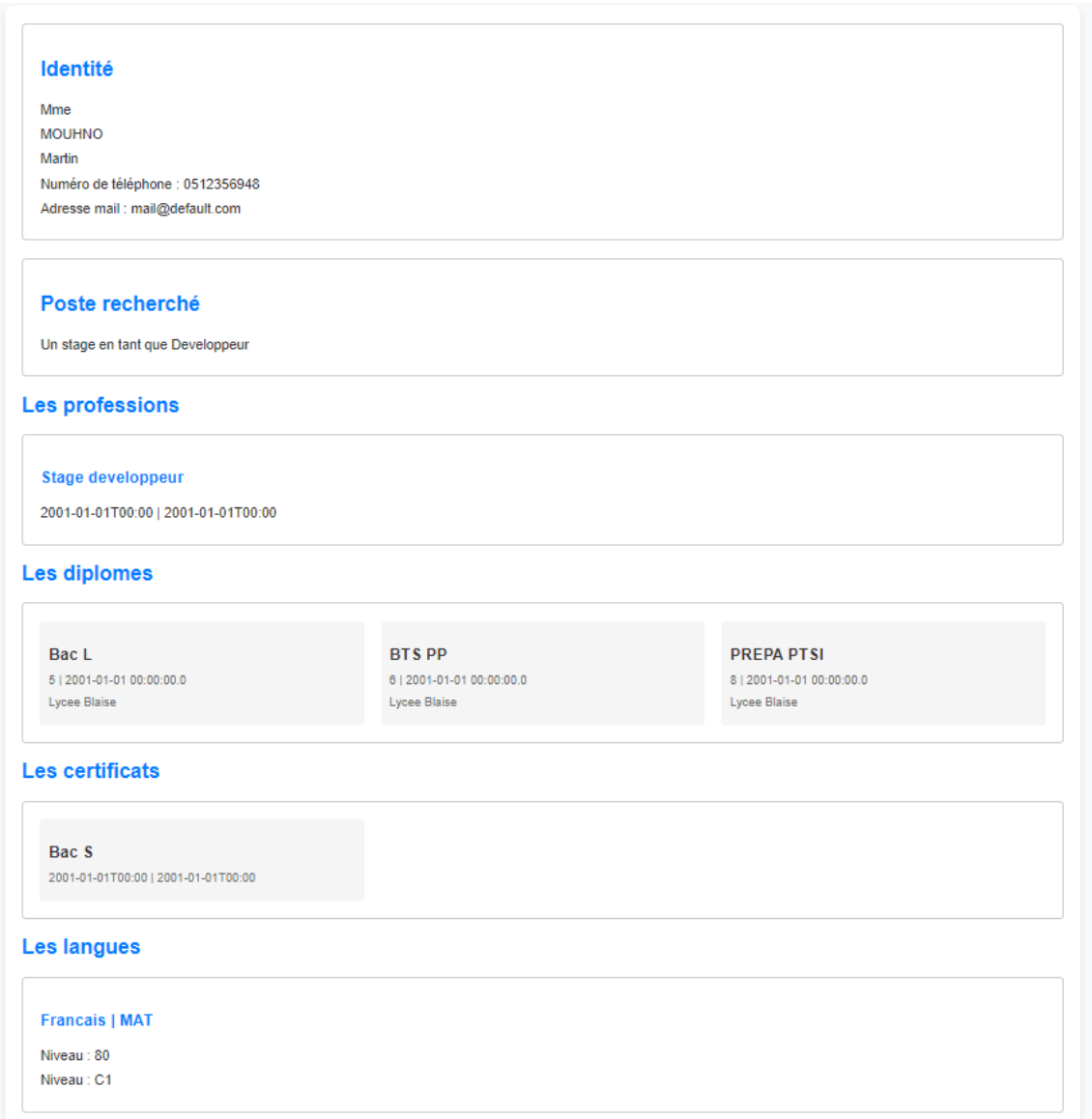

FIGURE 6 – Les détail d'un CV

URL : /cv24/insert

POST

Opération : Ajout d'un CV dans la base

Transmis : Flux XML respectant la spécification cv24 à ajouter, conforme au schéma xsd.

Description : Le flux reçu est validé par le schéma XSD de définition cv24. Si le flux est déjà présent, c'est-à-dire si les informations genre, nom, prénom et tel sont identiques, alors une indication d'erreur est retournée. Si l'opération est réussie, alors le CV est ajouté à la base et sa persistance est assurée. La valeur de <id> est générée automatiquement, par incrémentation de la dernière valeur enregistrée. Cette valeur doit obligatoirement être unique.

Retour : Le flux XML retourné contient les informations suivantes :

- id → numéro d'identifiant attribué à la spécification soumise
- status → INSERTED

En cas d'échec de l'opération, les informations de statut seront retournées :

- status → ERROR
- detail → INVALID | DUPLICATED ⇒ Selon l'origine de l'erreur

Exemple de commande :

```
POST http://votre-domaine.com/cv24/insert
```

Exemple de retour :

```
<response>
   <id>2</id>
   <status>INSERTED</status>
</response>

<!-- En cas d'erreur -->
<response>
   <status>ERROR</status>
   <detail>DUPLICATED</detail>
</response>

<!-- En cas d'erreur de syntaxe -->
<response>
   <status>ERROR</status>
   <detail>INVALID</detail>
</response>
```

FIGURE 7 – Une aide sur la commande d'ajout d'un CV depuis la page d'aide

8 Difficultés rencontrées

Durant ce projet, j'ai rencontré deux soucis majeurs :

- Le premier réside dans l'élaboration des différentes entités en fonction de la base de données qui avait été préalablement réfléchi. Je me suis rendue compte lors de ma seconde version de ce projet qu'il s'agissait essentiellement des erreurs de nommages. Je sais maintenant que la base de données et les entités sont reliées par les noms des champs qui les composent.
- Le second concerne l'utilisation de CleverCloud. En effet, j'ai du répéter en vain le déploiement, qui avait toujours connu le même sort. Je ne sais encore pas la raison. Les étapes sont les suivantes :
  - L'ajout de l'application issue du service REST
  - L'ajout de la base de données utilisée
  - Faire le lien entre la base de donnée de part les propriétés de l'application

  Je me rend compte d'un cas récurrent lors d'un arrêt immediat du déploiement, c'est lorsqu'il s'agit d'utiliser la base de données.

9 Améliorations possibles

Voici une liste d'améliorations qui ont été mises de côté par manque de temps.

- L'amélioration du schéma de validation de document XML, car en effet ne correspond pas exactement à celui demandé par le tableau présenté dans le sujet du TP1. Une amélioration quant à l'ordre des éléments ainsi que sur la présence ou non des éléments.
- Une possibilité d'ajouter un CV depuis un formulaire. Le formulaire participerait à la création du CV, et lors de la validation de ce dernier, ajouterait réellement le CV à la base de données.
- Une possibilité de modifier les données d'un CV de la même façon que l'ajout d'un CV. Cette fois ci, on aurait utilisé la requête PUT.
- Une amélioration de la recherche de CVs, on peut ajouter multiple recherche diverses et variées afin d'être plus fin sur la recherche de ces derniers.

10 Le lien github

Je tiens tout de même à préciser que le github n'est pas opérationnel pour le déploiement via CleverCloud. Voici le lien : https://github.com/IzaalalaCoder/cv24.