TP2 JSON-B, JAX-B & Validation

Objectifs :
- personnaliser le parsing et le rendu d'objets en JSON et en XML
- Valider les attributs des objets
- Gérer les erreurs (de validation) et retourner des messages JSON appropriés

1- Mise en place et analyse du service existant

Télécharger le projet cours2.
Ce projet consiste en un service qui permet de faire des opérations basiques sur des personnes : Create, Read, Update, Delete (CRUD). Il permet aussi de récupérer la liste des personnes et deux méthodes de mise à jour (Update) : la mise à jour totale via la méthode PUT et la mise à jour partielle via la méthode PATCH.

Testez les différentes méthodes.

Dans la méthode `create` :
A quoi sert le paramètre `@Context UriInfo uriInfo` ?
- Quelles sont les informations que l'on peut récupérer sur un objet de type `UriInfo` ?
- A quoi sert l'annotation `@Context` et avec quelles classe peut on l'utiliser ?

A quoi servent les différents appels de méthodes de la ligne
`return Response.created(location).entity(p).build();` ?
Que peut on faire avec la classe `Response` ?

2- Personalisation du mapping JSON-B

Testez la méthode de mise à jour complète (PUT) en changent l'identifiant (id) de la personne ? Qu'observez vous ?
Pour pallier ce type de problème, il faut que la transformation de JSON vers un objet java ignore la propriété/attribut `id`. Pour cela, JSON-B fourni l'annotation `@JsonbTransient` qui permet d'ignorer la propriété soit en lecture (placement sur le getter), soit en écriture (placement sur le setter), soit sur les deux (placement directement sur l'attribut ou sur les deux méthodes).
Pour plus d'information sur la personnalisation du mapping JSON – objets Java, allez voir la documentation officielle : https://javaee.github.io/jsonb-spec/users-guide.html

Testez et vérifiez que cela fonctionne. Que se passe t il si l'on fournit des propriétés dans le contenu JSON qui ne sont pas déclaré dans la classe Personne ?

En vous aidant de la documentation, proposez un moyen de renommer, dans le JSON, l'attribut « `id` » par « `_id` » sans changer son nom dans la classe java.

Testez.

3- Mapping vers XML JAX-B

De la même manière que pour JSON, on peut faire de la transformation objet vers XML. Il existe pour cela l'API JAX-B. Elle fonctionne également avec des annotations.
En vous basant sur la documentation : https://www.baeldung.com/jaxb, et en ajoutant un type sur le Consume de la méthode `read`, faites en sorte d'obtenir un rendu en XML. Il faut modifier dans Postman le header `Accept` de votre requête pour demander du `application/xml`.
Vous verrez que par défaut la date n'est pas rendue dans le XML, vous pouvez ajouter sur la méthode `getBirthdate()`, l'annotation `@XmlJavaTypeAdapter(MyLocalDateXmlAdapter.class)` qui fait référence à une classe adaptateur qui est fournie dans le projet.

4- La validation.

Pour l'instant, vous pouvez créer des Personnes avec des emails non valides, des nicknames vides, etc. Vous pouvez tester pour vous en convaincre.
Si l'on veut valider des contraintes sur des attributs, on peut utiliser l'API Bean Validation.

En vous aidant des la documentation : https://www.baeldung.com/javax-validation et https://beanvalidation.org/2.0/spec/#builtinconstraints , ajoutez des annotations sur la classe Personne pour que (1) le nickname ne soit pas vide ou seulement constitué d'espaces, (2) le mail soit valide, (3) que la date soit renseignée et dans le passé.

Pour tester le fonctionnement des annotations dans votre application, il faut annoter avec `@Valid` le paramètre de type Personne de la méthode `create`.
Testez.
Quelle erreur obtenez-vous en retour ?
Comme vous pouvez le voir le message d'erreur n'est pas très informatif. Nous allons remédier à cela.
Lorsque le validateur est exécuté (automatiquement avec l'annotation @Valid), si au moins une erreur est détectée alors une `Exception` de type `ConstraintViolationException` est levée.
Cela engendre la l'envoi d'une réponse avec le statut 400 Bad Request.

Si l'on veut générer une réponse avec un message explicite, il faut alors créer un `ExceptionMapper`.
Un `ExceptionMapper` permet de récupérer les exceptions d'un certain type et de générer une réponse « personnalisée ». Pour cela, il faut créer une classe implémentant l'interface `ExceptionMapper` :

```
@Provider
public class ExceptionMapperQuiNeFaitRien implements ExceptionMapper<T> {
   @Override
   public Response toResponse(T e) {
      return null;
   }
}
```

où T sera remplacé par le type de l'exception à gérer (`ConstraintViolationException` dans notre cas). L'annotation `@Provider` permet d'enregistrer notre mapper auprès de l'environnement d'exécution JAX-RS

Dans l'exemple de l'application, il est fourni une classe « `ValidationExceptionMapper` » qui permet de générer une réponse avec un objet JSON détaillant l'erreur. Pour l'activer, il suffit d'enlever le commentaire devant `@Provider`.
Testez et essayez de comprendre le code et son fonctionnement.

Personnalisation et internationalisation des messages d'erreur.
Les messages correspondant à une erreur de validation peuvent être personnalisés et internationalisés. Pour personnaliser un message, il suffit d'ajouter « (message="un msg") » derrière les annotations de validation. Par exemple :

```
@Past(message="You are not born, ${validatedValue} is in the past !")
private LocalDate birthdate;
```

Vous pouvez voir quelques exemples de messages sur https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/#_examples

Les messages d'erreurs de validation peuvent être externalisés du code et internationalisés.
Pour cela, il faut créer un fichier « `ValidationMessages.properties` » par langue à la racine des sources (dans resources sur Idea). Ces fichiers sont déjà donnés dans le projet. Le principe de nommage est le suivant : `ValidationMessages_CODELANG_CODEPAYS.properties`. Par exemple, si l'on veux du français de France, il faut créer un fichier `ValidationMessages_fr_FR.properties`.
Au niveau du code, il faut associer une clé au message :

```
@NotNull(message="{cours2.person.email.notnull}")
private String email;
```

et cette clé doit avoir un mapping dans les fichiers `ValidationMessages`.
Testez…

5- Pour aller plus loin…

JSON-B permet de redéfinir le mapping par défaut via des adapters ou des serializer (voir les sections correspondantes dans la documentation officielle https://javaee.github.io/jsonb-spec/users-guide.html).
Quand on y regarde de plus près, la sérialization d'un ensemble d'objets, n'est pas « optimisée ». On pourrait par exemple sérialiser chaque instance comme un tableau (voir https://www.baeldung.com/json-reduce-data-size#array). A partir de la documentation proposez une méthode qui permet de serialiser une liste de personnes avec un tableau de tableau où le premier élément représente les noms de propriétés et les suivantes les valeurs des instances pour ces propriétés.
Une fois cela réalisez, essayer en vous aidant de l'introspection sur les java beans (voir https://www.jmdoudoux.fr/java/dej/chap-javabean.htm#javabean-5) une méthode générale pour sérialiser en JSON de manière « optimisée » n'importe quelle collection d'objets.

Si vous avez terminé, vous pouvez réfléchir à comment séparer la partie de gestion des données (stockage ou autre), de la partie contrôleur (le service REST).