Le Pattern DTO

Quand une API dépasse le simple CRUD sur une entité, on commence à avoir besoin formaliser les entrées et les sorties de notre API. Par exemple losqu'on a des entités JPA et qu'on ne veut pas les exposer intergralement au niveau du client pour différentes raisons. Par exemple, si on essaye de sérialiser en JSON une entité membre d'une relation à double navigation (le parent voit ses enfants et les enfants voient le parent) on aura une boucle finie.

Les Data Transfer Objects

Les DTO sont des classes toutes simples qui ne contiennent que des propriétés (champs privé avec getter & setters) et sont utilisées comme paramètre et valeur de retour des controleurs. Ils voyagent également jusque dans les services une fois validés. On va donc avoir tendance à créer pour chaque forme de requête et de réponse un nouveau DTO.

Exemple :

```
public class CreateTodoDTO {

  private String title;
  private String description;

  ... getters & setters ...
}
```

Pour créer un Todo, l'Id n'est pas spécifié par l'utilisateur, on va donc créer un DTO de Todo sans Id, et le prendre en paramètre lors de la création des Todos.

Aussi :

```
public class TodoDTO {
  private String id;
  private String title;
  private String description;
  private String ownerId;
  ... getters & setters ...
}
```

Pour notre DTO qui permet d'envoyer un Todo au client, on rajoute l'Id, et on remplace la référence à l'utilisateur par uniquement son Id, afin d'éviter la boucle infinie étant donné que la référence à l'user liste elle même les Todos.

Validation Automatique

Spring Boot implémente un standard de validation Java (javax.validation) des DTO basé sur des annotations, qui permet de façon déclarative d'appliquer des contraintes sur les propriétés des DTO.

Installation :

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Les principales annotations disponibles :

- `@NotNull` : pas de valeur `null`
- `@NotEmpty` : pour les `String` et les `Collection`, la taille doit être supérieure à 0
- `@NotBlank` : pour les `String`, imposer que la chaine ne doit pas être vide
- `@AssertTrue` / `@AssertFalse` : pour les `boolean`
- `@Min` / `@Max` : pour les valeur numérques, pour imposer in minimum / maximum
- `@Positive` / `@PositiveOrZero` / `@Negative` / `@NegativeOrZero` : permet de contraindres les valeurs numériques selon le signe
- `@Email` : pour vérifier qu'une `String` a la forme d'une email
- `@Future` / `@Past` / `@FuturOrPresent` / `@PastOrPresent` : pour contraindre les dates par rapport à la date courante

Les annotations peuvent aussi être utilisées sur les objets dans les collections : `List<@NotBlank String>`. En utilisant ces annotations, les DTO sont automatiquement validés par le framework lorsqu'ils passent pas le controleur lorsqu'il sont marqués par l'annotation `@Valid`.

Exemple pour nos DTO de Todos :

```
public class CreateTodoDTO {
  @Size(min=6, max=255)
  private String title;

  @NotEmpty
  private String description;
```

```
  ... getters & setters ...
}
```

Et dans notre méthode endpoitn :

```
@PostMapping
public void createTodo(@Valid @RequestBody CreateTodoDTO dto){
...
}
```

Revision #7
Created 2021-01-25 20:13:01 UTC by Arsène Lapostolet
Updated 2022-09-28 12:49:42 UTC by Arsène Lapostolet