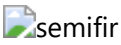

Spring Dto

C'est quoi un Dto ?

DTO ou Data Transfert Object

- C'est un objet qui permet de transférer des données entre les couches de l'application
- C'est a dire qu'il permet de séparer les données de la couche de présentation de la couche métier.

Il permet aussi de réduire la taille des données à transférer et de ne pas envoyer des données inutiles.

Comment mettre en place les Dto ?

Dans un premier temps nous allons regarder comment mettre en place les Dto dans une application Spring Boot.

Puis nous allons avoir besoin d'utiliser un objet nommé `ModelMapper` qui va nous permettre de mapper les données de l'entité vers le Dto et inversement.

`ModelMapper` provient d'une librairie nommé MapStruct qui permet de mapper les données d'un objet vers un autre objet.

Nous allons partir d'un exemple basique de Spring Boot avec une entité `User`, une entité `Location` et un CRUD classique sur ces deux entités.

Mais Avant ca nous allons mettre en place le projet Spring Boot.

Nous allons avoir besoin de plusieurs dépendances pour mettre en place le projet.

- Spring Boot Web (Pour faire une API Rest)
- Lombok (Pour nous économiser du code)
- H2 Database (Pour avoir une base de données en mémoire)
- MapStruct (Pour mapper les données d'un objet vers un autre objet)

User

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String email;
    private String nom;
    private String prenom;
    private String password;

    @ManyToOne()
    @JoinColumn(name = "location_id")
    private Location location;
}
```

Location

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "locations")
public class Location {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private String place;
    private String description;
    private double latitude;
    private double longitude;
}
```

> Ici `User` & `Location` sont liés par une relation ManyToOne, c'est à dire qu'un User peut avoir une seule Location mais une Location peut être liée à plusieurs User.

De plus nous utilisons Lombok pour nous économiser du code.

Il faut ensuite mettre en place les Repository.

UserRepository & LocationRepository

```
@Repository
public interface LocationRepository extends JpaRepository<Location, Long> {
}
```

```
@Repository
public interface UserRepository  extends JpaRepository<User, Long> {
}
```

> Ici nous avons juste besoin d'implémenter les interfaces `JpaRepository` qui nous permettent d'avoir les méthodes de base pour faire un CRUD.

> Avant la mise en place du service nous allons implémenter notre mapper ainsi que le DTO pour pouvoir filtrer notre donnée.

UserLocationDTO

```
@Data
public class UserLocationDTO {

    private long userId;
    @Email @NotNull
    private String email;
    @NotNull
    private String place;
    @NotNull
    private double latitude;
    @NotNull
    private double longitude;
}
```

- Le DTO reprend ainsi certain champs de l'entité `User` et de l'entité `Location`.

- On remarque donc que le DTO va nous permettre de ne pas envoyer des données inutiles, c'est une sorte de filtre de données.

- De plus nous utilisons les annotations `@NotNull` et `@Email` pour valider les données.
- Cela permet d'empêcher l'envoi de données vides ou de données qui ne correspondent pas au format attendu.

> Maintenant que nous avons notre DTO nous allons pouvoir mettre en place le mapper.

UserMapper

```java
@Mapper(componentModel = "spring")
public interface UserMapper {

    @Mapping(source = "id", target = "userId")
    @Mapping(source = "email", target = "email")
    @Mapping(source = "location.place", target = "place")
    @Mapping(source = "location.latitude", target = "latitude")
    @Mapping(source = "location.longitude", target = "longitude")
    UserLocationDTO userToUserLocationDTO(User user);

    List<UserLocationDTO> usersToUserLocationDTOs(List<User> users);

}
```

> `@Mapper(componentModel = "spring")`

Cette annotation configure le comportement de MapStruct pour cette interface.

Le paramètre `componentModel = "spring"` indique que les instances de cette interface seront gérées par Spring et que nous pourrons les injecter dans d'autres composants.

> `UserLocationDTO userToUserLocationDTO(User user)`

Cette méthode définit le mappage d'un objet User vers un objet UserLocationDTO.

Chaque annotation `@Mapping` spécifie comment les attributs de l'objet source (User) sont mappés sur les attributs de l'objet cible `(UserLocationDTO)`.

Par exemple, `source = "id"` indique que l'attribut id de User est mappé vers l'attribut userId de UserLocationDTO.

> `List<UserLocationDTO> usersToUserLocationDTOs(List<User> users)`

Cette méthode définit le mappage d'une liste d'objets `User` vers une liste d'objets `UserLocationDTO`.

Cette méthode peut être utilisée pour mappage de collections.

- Le mapper va nous permettre de mapper les données de l'entité `User` vers le DTO `UserLocationDTO`.
- On remarque que l'on peut mapper les données de l'entité `Location` qui est liée à l'entité `User`.

- On peut aussi mapper une liste d'entité `User` vers une liste de DTO `UserLocationDTO`.
- Cela va nous permettre de récupérer une liste de DTO pour l'envoyer à la couche de présentation.

> Maintenant que nous avons notre mapper nous allons pouvoir mettre en place le service.

UserService

```
@Service
public class UserService {

    private final UserRepository userRepository;

    private final UserMapper userMapper;

    public UserService(UserRepository userRepository , UserMapper userMapper) {
        this.userRepository = userRepository;
        this.userMapper = userMapper;
    }

    public List<UserLocationDTO> findAllUsers() {
        return userMapper.usersToUserLocationDTOs(userRepository.findAll());
    }

    public UserLocationDTO findByIdUsers(long id) {
        return
userMapper.userToUserLocationDTO(userRepository.findById(id).get());
    }
}
```

> Ici nous avons besoin d'injecter le `UserRepository` et le `UserMapper` dans le constructeur.

> `userMapper.usersToUserLocationDTOs(userRepository.findAll())`

Cette ligne permet de mapper les données de l'entité `User` vers le DTO `UserLocationDTO`.

> `userMapper.userToUserLocationDTO(userRepository.findById(id).get())`

Cette ligne permet de mapper les données de l'entité `User` vers le DTO `UserLocationDTO` en fonction de l'id.

> Maintenant que nous avons notre service nous allons pouvoir mettre en place le controller.

UserController

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping("/findAll")
    public ResponseEntity<List<UserLocationDTO>> findAllUsersMapper() {
       return new ResponseEntity<>(userService.findAllUsers(), HttpStatus.OK);
    }

    @GetMapping("/findById/{id}")
    public ResponseEntity<UserLocationDTO> findByIdUsersMapper(@PathVariable(value
= "id") long id) {
        return new ResponseEntity<>(userService.findByIdUsers(id), HttpStatus.OK);
    }
}
```

> Ici nous avons besoin d'injecter le `UserService` dans le constructeur.

> `@GetMapping("/findAll")` `@GetMapping("/findById/{id}")`

Ces deux annotations nous permettent de définir les routes de l'API.

> Maintenant que nous avons notre controller nous allons pouvoir tester notre API.

Test de l'API

- Pour tester l'API nous allons utiliser Postman.

- Nous allons donc envoyer une requête GET sur l'URL `http://localhost:8080/api/users/findAll`

```
[
  {
    "userId": 1,
    "email": "pierre.gaillard@live.fr",
```

```
    "place": "Lille",
    "latitude": 50.62925,
    "longitude": 3.057256
  },
  {
    "userId": 2,
    "email": "emilie.gaillard@live.fr",
    "place": "Lille",
    "latitude": 50.62925,
    "longitude": 3.057256
  }
]
```

- On remarque que l'on reçoit bien une liste de DTO.

- Nous allons ensuite envoyer une requête GET sur l'URL `http://localhost:8080/api/users/findById/1`

```
{
  "userId": 1,
  "email": "pierre.gaillard@live.fr",
  "place": "Lille",
  "latitude": 50.62925,
  "longitude": 3.057256
}
```

- On remarque que l'on reçoit bien un DTO en fonction de l'id.

Suite du cours