TP Spring n°6
Spring Data

On va, dans ce TP, afficher des données qui sont en base. On ne vas plus instancier nos bean en ajoutant des produits, stock, commandes dans des listes. Mais, lors de l'affichage de nos pages, on va récupérer les informations stockées dans les tables de la base de données. Les ArrayList dans les classes de service vont disparaître.

1° - Création de la base et lien dans Spring

1° - Créez sur pgAdmin (ou mySQL) une base de données, son nom : e_commerce. Ajoutez la table product qui doit avoir les colonnes suivantes :

- id : bigint NOT NULL - PRIMARY KEY
- description : text
- name : text
- price : double
- picture : text
- quantity : bigint

2° - Intégration de la base sous Spring. Faites le lien avec votre base de données dans le fichier application.properties :

```
spring.datasource.url=jdbc:postgresql://localhost:5432/e_commerce
spring.datasource.username=postgres
spring.datasource.password=******************
spring.datasource.driver-class-name=org.postgresql.Driver
# Keep the connection alive if idle for a long time (needed in production)
spring.datasource.testWhileIdle=true
spring.datasource.validationQuery=SELECT 1
```

3° - Récupération des données. Votre classe Product est l'image d'une ligne dans la table product. Afin de préciser cela, ajoutez l'annotation *@Entity* à votre classe Product. L'attribut représentant la clé primaire doit également être annoté par *@Id*.

On obtient :

```
@Entity
public class Product {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	// ....
}
```

Attention : si la classe n'a pas le même nom que la table en base, vous devez le préciser en ajoutant l'annotation *@Table(name = "nom_de_la_table")*, les majuscules ne sont pas prises en compte.

2° - Spring Data

L'interface principale de l'abstraction du référentiel Spring Data est Repository. Cette interface utilise la généricité et a besoin du type de l'entité ainsi que du type de l'id de cette entité. C'est pour cette raison que nous avons annoté la classe Product avec @Entity et l'entier id avec @Id. Nous allons utiliser l'interface *JpaRepository* qui fournit des fonctionnalités pour la classe d'entité gérée.

Cette interface hérite de l'interface CrudRepository qui offre les fonctionnalités de base. L'acronyme informatique anglais CRUD (pour create, read, update, delete) désigne les quatre opérations de base pour la persistance des données.

L'interface *CrudRepository* ressemble à ça :

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {
	<S extends T> S save(S entity);		// Pour sauvegarder l'entité
	Optional<T> findById(ID primaryKey); // Pour trouver l'entité depuis la pk
	Iterable<T> findAll();				// SELECT * FROM entité
	long count();						// Nb de lignes dans la table
	void delete(T entity);				// Supprime une ligne
	boolean existsById(ID primaryKey);	// On a compris
}
```

1° - Créez un package *repository*, ajoutez une interface avec le nom *ProductRepository*. Cette interface hérite de *CrudRepository* avec le couple *Product* (le type de l'entité) et *Long* (le type de l'id).

On obtient :

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

}
```

Rien de plus ici, c'est une interface, donc pas d'implémentation de méthodes, et pas de nouvelle méthode à ajouter, notre interface *ProductRepository* hérite déjà des méthodes de son parent comme findById qui lui retourne un Optional de Product.

On ajoutera par la suite des méthodes afin d'ajouter des fonctionnalités à cette couche permettant de manipuler la base de données.

2° - Nous allons utiliser cette interface dans notre service. Nous avons plus besoin de cette ArrayList, vous pouvez la supprimer. A la place, ajoutez un attribut de type *ProductRepository*, inscrivez l'annotation *@autowired* afin de récupérer l'instance par injection de dépendance.

Voici le début du code :

```
@Service("products")
public class ProductServiceImpl implements ProductService {

	@Autowired
	private ProductRepository productRepository;

	@Override
	public List<Product> getAllProducts() {
		// TODO
	}

	@Override
	public Product getProductById(Long id) {
		// TODO
	}

	@Override
	public Product save(Product product) {
		// TODO
	}
}
```

Testez votre code. Pensez à ajouter quelques produits dans la table.

Affichez les produits qui sont en base dans votre page */products*.
Maintenant toutes vos données sont en base, et vous devez voir votre application tourner et les pages s'afficher avec les informations.

Petit Bonus : Vous pouvez tester l'insertion en base avec ce que vous venez de créer. Modifiez la méthode *runner* qui est exécutée lors du lancement de l'application :

```
@Bean
CommandLineRunner runner(ProductService productService) {
	return args -> {
		Product product1 = new Product(null, "ALOHA SURFBOARDS", "ALOHA SURFBOARDS - FUN DIVISION 6'4 ECOSKIN - FCS2", 755.00 , "https://www.woodstockshop.com/13961-thickbox_default/aloha-surfboards-fun-division-6-4-ecoskin-fcs2.jpg", 3l);

		Product p = productService.save(product1);
	};
}
```

Attention : mettez ce code en commentaire après le lancement du serveur afin qu'il ne soit pas exécuté une seconde fois et qu'il ajoute à nouveau le produit en base. Et puis, si vous faites un copier/coller, vous allez ajouter une planche de surf…

3° - Faites la même chose pour le client. Ajoutez des clients en base de données, vous pouvez tester que Spring récupère bien les informations en créant à la manière d'une API (Application Programming Interface) un RestController qui retourne la liste des clients.

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

	@Autowired
	private ClientService clientService;

	@GetMapping(value = "/clients")
	public List<Client> getClients(Model model) {
		System.out.println("/clients : get all clients");

		return clientService.getAllClients();
	}
}
```

Url : http://localhost:8080/api/clients
Résultats :

```
[
	{
		"id": 1,
		"username": "peter",
		"password": "1234"
	},
	{
		"id": 2,
		"username": "Bobby",
		"password": "pass"
	}
]
```

4° - On souhaite maintenant trouver un utilisateur depuis son username.
Url : http://localhost:8080/api/clients?username=peter

Écrivez la nouvelle méthode dans le controller, récupérez le username grâce à l'annotation RequestParam. Pour réaliser celà, ajoutez une nouvelle méthode dans votre interface Repository de votre client :

```
Optional<Client> findByUsername(String username);
```

Il "suffit" de bien nommer cette méthode, avec "findBy" et le nom de l'attribut, plus le paramètre pour qu'il traduire cette méthode en une requête :
SELECT * FROM table_client WHERE table_client.username = "paramValue";

Pour en savoir plus sur cette magie :
https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories

Résultats :

```
{
	"id": 1,
	"username": "peter",
	"password": "1234"
}
```

4° - Faites une nouvelle fois la même chose pour votre classe Order. Elle devient une *@Entity*, ajoutez l'annotation *@Id* et ajoutez sur l'attribut client l'annotation *@ManyToOne.*

L'annotation @ManyToOne définit une relation n:1 entre deux entités.

L'annotation @ManyToOne implique que la table *Order* contient une colonne qui est une clé étrangère contenant la clé d'un Client. Par défaut, JPA s'attend à ce que cette colonne se nomme client_id, mais il est possible de changer ce nom grâce à l'annotation @JoinColumn.

On respecte ainsi notre modélisation UML :

```
@ManyToOne
private Client client;
```

Ou alors, si la colonne ne se nomme pas client_id en base :

```
@ManyToOne
@JoinColumn(name = "fk_id_client")
private Client client;
```

Ajoutez en base une commande afin d'afficher celle-ci dans le navigateur grâce à votre RestController. Vous devez voir apparaître la commande avec les informations du client.

Trouvez une solution afin de ne pas afficher le mot de passe du client.

On obtient :

```
[
	{
		"id": 1,
		"dateCreated": "2022-04-24",
		"status": "Payée",
		"client": {
			"id": 1,
			"username": "peter"
			},
		"orderProducts": [],
		"totalOrderPrice": 0,
		"numberOfProducts": 0,
		"totalNumberOfProducts": 0
	}
]
```

5° - Pour la classe OrderProduct, la clé primaire est une composition des colonnes order_id et product_id.

Pour réaliser cela, vous avez besoin de créer une nouvelle classe *OrderProductId* qui représente cette clé composite. Cette classe sera marquée avec l'annotation *@Embeddable.*

```
@Embeddable
public class OrderProductId implements Serializable {

	@Column(name = "product_id")
	private Long productId;

	@Column(name = "order_id")
	private Long orderId;
}
```

Ajoutez des méthodes de type set ou alors un constructeur avec les deux arguments.

Ajoutez un attribut de ce type à votre classe OrderProduct avec l'annotation *@EmbeddedId*.

Pour faire le lien avec le *product*, l'attribut doit être annoté de *@ManyToOne*

```
@Entity
@Table(name = "table_order_product")
public class OrderProduct {

	@EmbeddedId
	private OrderProductId id;

	@ManyToOne
	@JoinColumn(name = "product_id", insertable = false, updatable = false)
	private Product product;

	// ...
}
```

6° - Il vous reste à faire le lien dans l'entité Order avec l'annotation OneToMany sur la liste des OrderProduct pour récupérer les lignes de commandes de cette commande en base.

Ajoutez en base des lignes de commande à votre commande afin d'afficher celle-ci dans le navigateur grâce à votre RestController. Vous devez voir apparaître la commande avec les informations du client.

On obtient :


```
[
	{
		"id": 1,
		"dateCreated": "2022-04-24",
		"status": "Payée",
		"client": {
			"id": 1,
			"username": "peter"
			},
		"orderProducts": [
			{
				"product": {
					"id": 3,
					"name": "CLAYTON SURFBOARDS",
					"description": "La… ",
					"price": 599,
					"picture": "https://...jpg",
					"quantity": 0
				},
				"quantity": 2,
				"totalPrice": 1198
			},
			{
				"product": {
					"id": 1,
					"name": "Mick Fanning SOFTBOARDS",
					"description": "Coll… ",
					"price": 450,
					"picture": "https://...jpg",
					"quantity": 0
				},
				"quantity": 1,
				"totalPrice": 450
			}
		],
		"totalOrderPrice": 1648,
		"numberOfProducts": 2,
		"totalNumberOfProducts": 3
	}
]
```