Spring REST et Spring Data JPA

Démonstration 2 du module 6

L'objectif de cette démonstration est la mise en place de Spring Data JPA avec Spring Rest.

Ainsi l'application Web orientée REST aura sa couche DAL

Contexte

- Continuer le projet précédent.

Déroulement

Configuration de Spring Boot dans build.gradle

- Ajout du starter de Spring Data JPA
- Ajout du driver de la base de données MySQL

```
dependencies {
	//Starter Spring Data JPA
	implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
	//Driver de MySQL
	runtimeOnly 'mysql:mysql-connector-java'
```

- Faire « Refresh Gradle Project »

Configuration de la connexion à la base de données

- Intégrer dans application.properties :
  - La configuration de la base de données (URL, user, password, driver)
- Et des paramètres pour JPA et Hibernate :
  - La permission de créer les tables à partir des entités (`spring.jpa.hibernate.ddl-auto=update`)
  - L'affichage des requêtes et le format d'affichage

```
#Configuration DB
spring.datasource.url=jdbc:mysql://localhost:3306/db_spring_demo?useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=Pa$$w0rd

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

BO – mise en place de l'entité Article :

- Modification de classe Article pour qu'elle devienne une entité JPA:

```
package fr.eni.demo.bo;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Article {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private long id;
	@Column(nullable = false)
	private String mark;
	@Column(nullable = false)
	private String department;
	@Column(nullable = false)
	private String title;
	private String description;
	private float price;
	…

}
```

DAL :

- Création d'une interface JpaRepository pour Article

```
package fr.eni.demo.dal;

import org.springframework.data.jpa.repository.JpaRepository;

import fr.eni.demo.bo.Article;

public interface ArticleRepository extends JpaRepository<Article, Long> {

}
```

BLL :

- Placer l'annotation @Profile(«dev») sur la classe ArticleServiceMock.
- Création de la classe, ArticleServiceImpl qui implémente l'interface ArticleService et utilise le JpaRepository précédent :

```
package fr.eni.demo.bll;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import fr.eni.demo.bo.Article;
import fr.eni.demo.dal.ArticleRepository;

@Service
public class ArticleServiceImpl implements ArticleService {
	// injection du bean Repository
	private ArticleRepository articleRepository;
```

```
	@Autowired
	public ArticleServiceImpl(ArticleRepository articleRepository) {
		this.articleRepository = articleRepository;
	}

	@Override
	public List<Article> findAll() {
		return articleRepository.findAll();
	}

	@Override
	public Article findById(Long id) {
		Optional<Article> result = articleRepository.findById(id);
		//Difference between exist or no
		if(result.isPresent()) {
			return result.get();
		}else {
			throw new RuntimeException("Article id not found - " + id);
		}
	}

	@Override
	public void save(Article article) {
		boolean isValid = validateMark(article.getMark());
		if(isValid) {
			articleRepository.save(article);
		}
	}

	private boolean validateMark(String data) {
		if(data == null || data.isBlank()) {
			throw new RuntimeException("La marque est obligatoire");
		}
		return true;
	}

	@Override
	public void delete(Long id) {
		articleRepository.deleteById(id);
	}
}
```

Contrôleur de l'application Web REST :

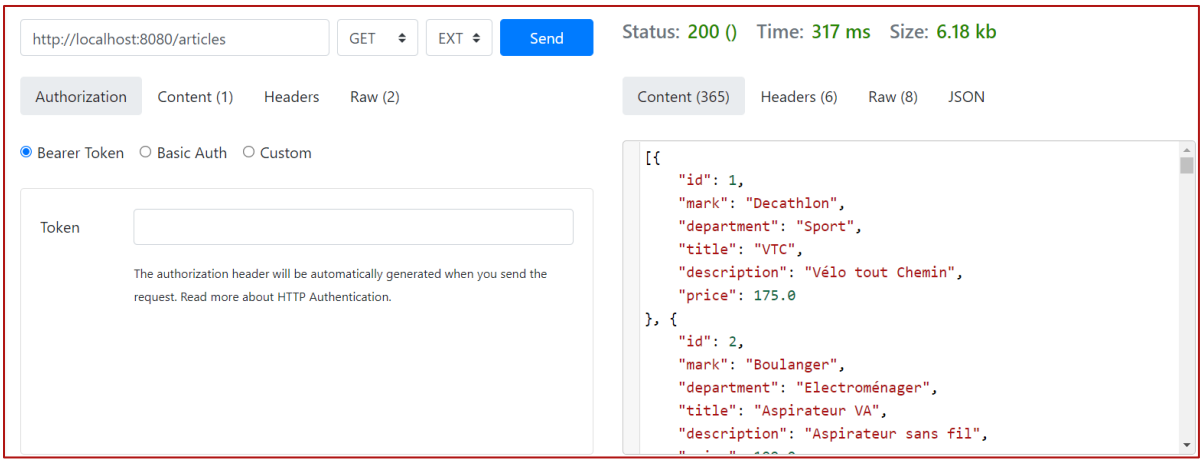
- Rien à faire car nous injection par l'interface ArticleService et les méthodes n'ont pas changé.

Utilisation de ReqBin pour tester les requêtes :

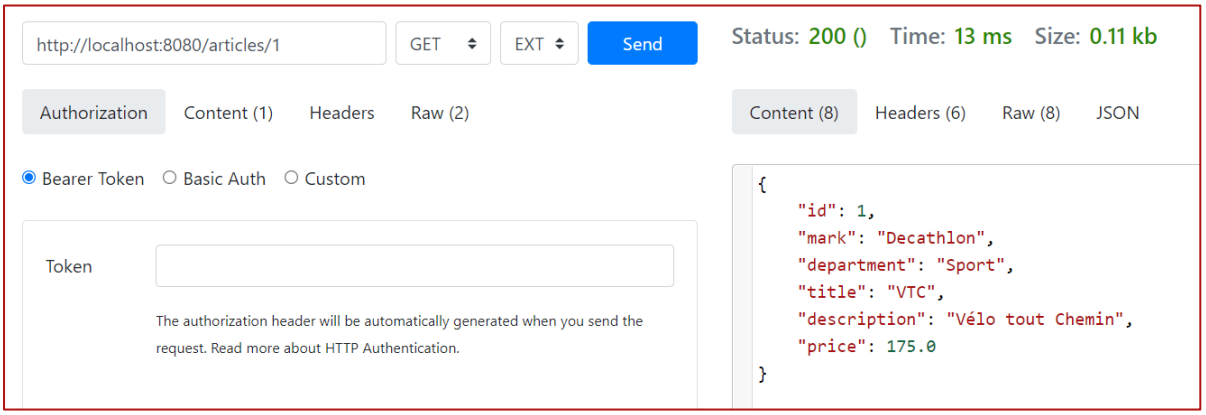
Exécution :

- Lancer l'application de votre serveur.
- Il va créer la table Article en base de données
  - Ajouter des articles avec le fichier data.sql fourni dans les ressources
- Puis dans ReqBin tester l'ensemble des méthodes vous retrouverez le même comportement que précédemment.
- Vous pouvez vérifier aussi dans MySQL Server les résultats.

- Get : http://localhost:8080/articles



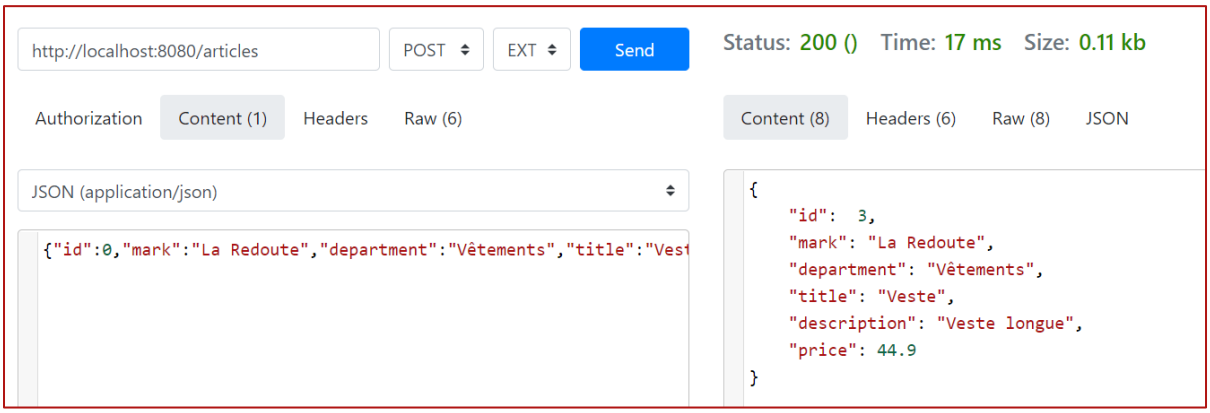
- Get by id : http://localhost:8080/articles/1



- Post : http://localhost:8080/articles
  - Utiliser le JSON suivant pour créer un nouvel article :

```
{"id":0,"mark":"La Redoute","department":"Vêtements","title":"Veste","description":"Veste longue","price":44.9}
```

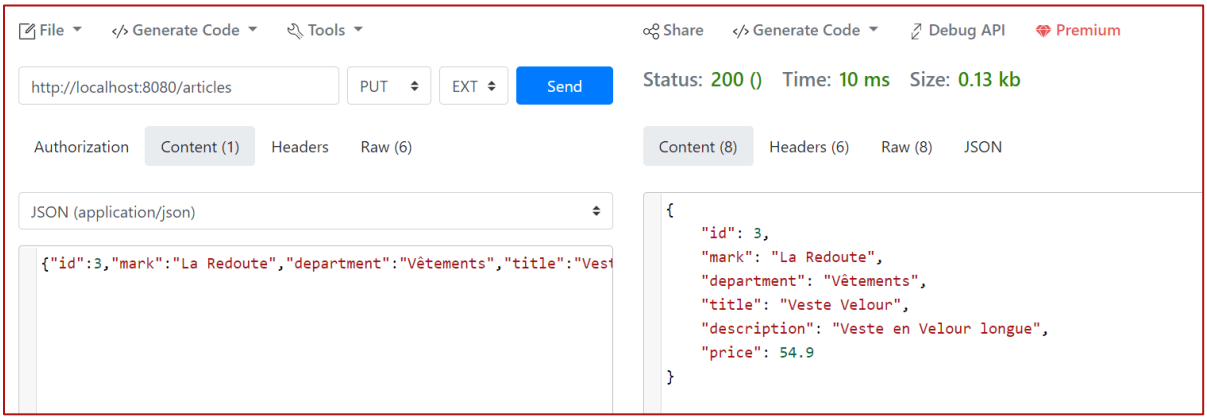
  - Mettez bien en POST
  - Et sélectionnez bien JSON (application/json)



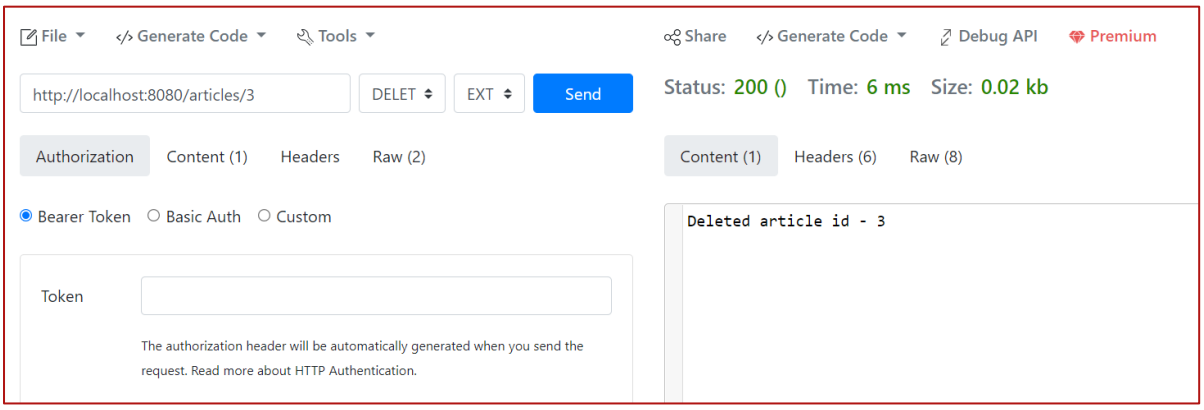
- Put : http://localhost:8080/articles
  - Fonctionnement similaire à POST
  - Utiliser le JSON suivant pour mettre à jour l'article précédemment enregistré :

```
{"id":3,"mark":"La Redoute","department":"Vêtements","title":"Veste Velour","description":"Veste en Velour longue","price":54.9}
```

  - Mettez bien en PUT
  - Et sélectionnez bien JSON (application/json)



- Delete : http://localhost:8080/articles/3
  - Sélectionnez DELETE



Conclusion :

- Avec notre application Web sur l'architecture REST;
- Il est possible de transmettre des données au format JSON, XML, …
- Et de l'interfacer avec une couche présentation orientée JS.
- Nous avons moyen de sécuriser les accès entre les :
  - URL des contrôleurs
  - La couche Service
  - Protection par injection de SQL au travers de Spring Data JPA.