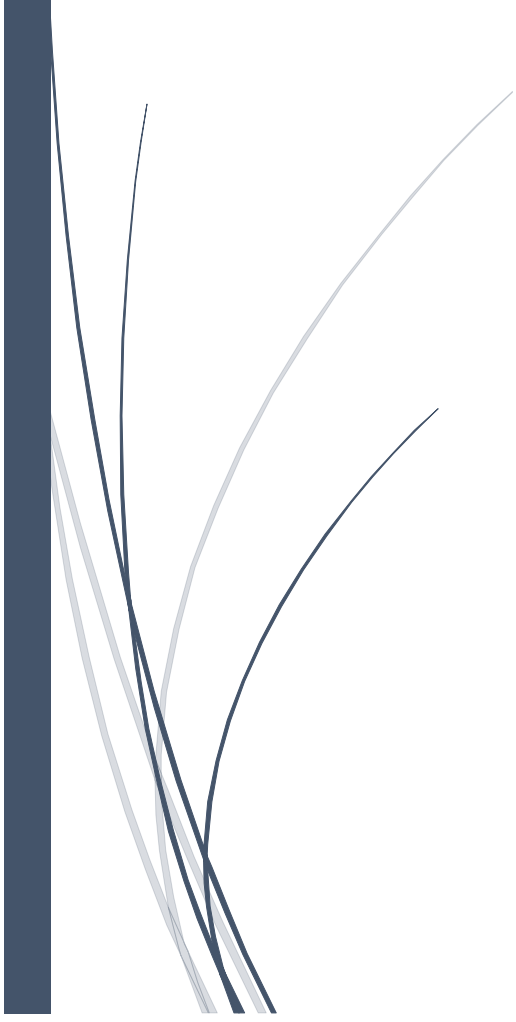
TP : Développer un service web avec REST en utilisant Spring BOOT et Spring Rest



Architecture des composants d’entreprise

Table des matières

I. Objectif du TP

- Comprendre l'architecture de Rest.
- Comprendre les 04 méthodes GET, POST, PUT et DELETE.
- Implémenter Rest avec Spring Boot et Spring Rest.
- Gérer les exceptions avec Spring (@ControllerAdvice) lors du traitement d'une requête Rest.
- Comprendre comment Spring Rest peut générer la ressource en format XML.
- Développer les tests unitaires et les tests d'intégration avec Spring, Junit et Mockito.
- Faire les tests avec postman.
- Comprendre le principe de « Spring Active Profile ».
- Configurer et administrer le conteneur web Apache Tomcat version 10.
- Comprendre comment avec Spring Boot créer le livrable WAR.
- Déployer le fichier WAR sur Tomcat.

NB : Le code source du TP est disponible sur GITHUB :

- ✓ Partie 1 : https://github.com/abbouformations/tprest.git
- ✓ Partie 2 : https://github.com/abbouformations/tprestwar.git

II. Prérequis

- IntelliJ IDEA ;
- JDK version 17 ;
- Une connexion Internet pour permettre à Maven de télécharger les librairies.
- Postman pour effectuer les tests.
- Apache Tomcat version 10.

NB : *Ce TP a été réalisé avec IntelliJ IDEA 2023.1.2 (Community Edition).*

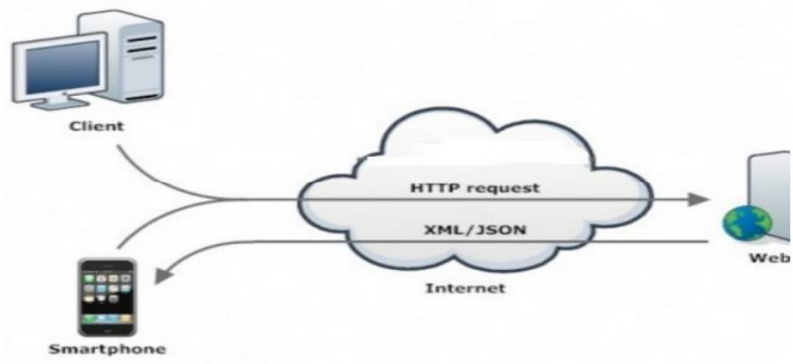
III. Le protocole de communication Rest

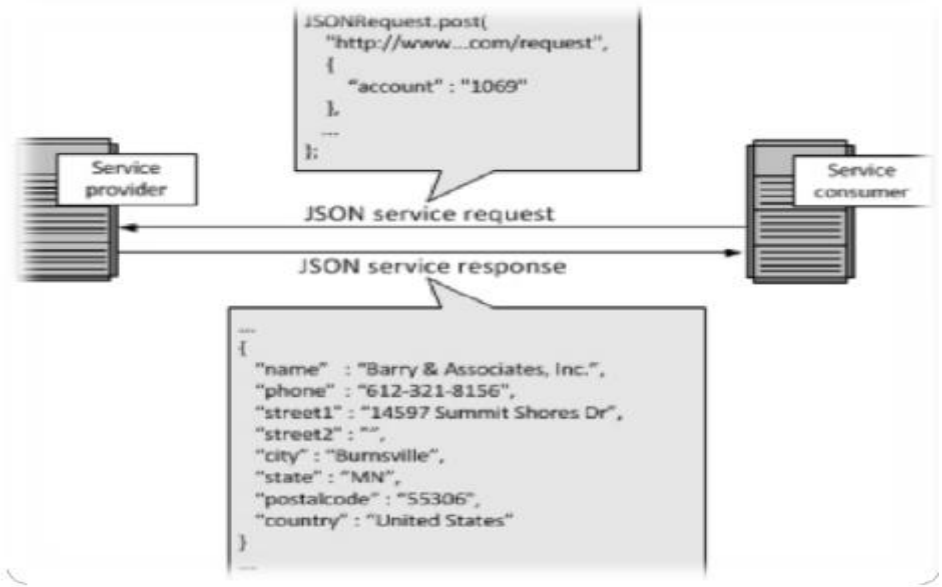
1. C'est quoi Rest

- ✓ REST (***RE****presentational* ***S****tate* ***T****ransfer*) c'est un style d'architecture (introduit par Roy Fielding en 2000) ;
- ✓ REST utilise HTTP dans le protocole de transport ;
- ✓ REST se base sur des ressources où chaque composant est une ressource et une ressource est accessible par une interface commune en utilisant des méthodes HTTP standards (GET, POST, PUT et DELETE) ;
- ✓ Le serveur REST fournit l'accès aux ressources et le client REST accède et présente les ressources ;
- ✓ Une ressource est identifiée par une URI (Uniform Resource Identifier) ;
- ✓ REST utilise diverses représentations pour représenter une ressource comme du texte, JSON et XML (JSON est le format le plus populaire utilisé dans les services.

2. L'architecture Rest

Les deux schémas suivants expliquent l'architecture de Rest :





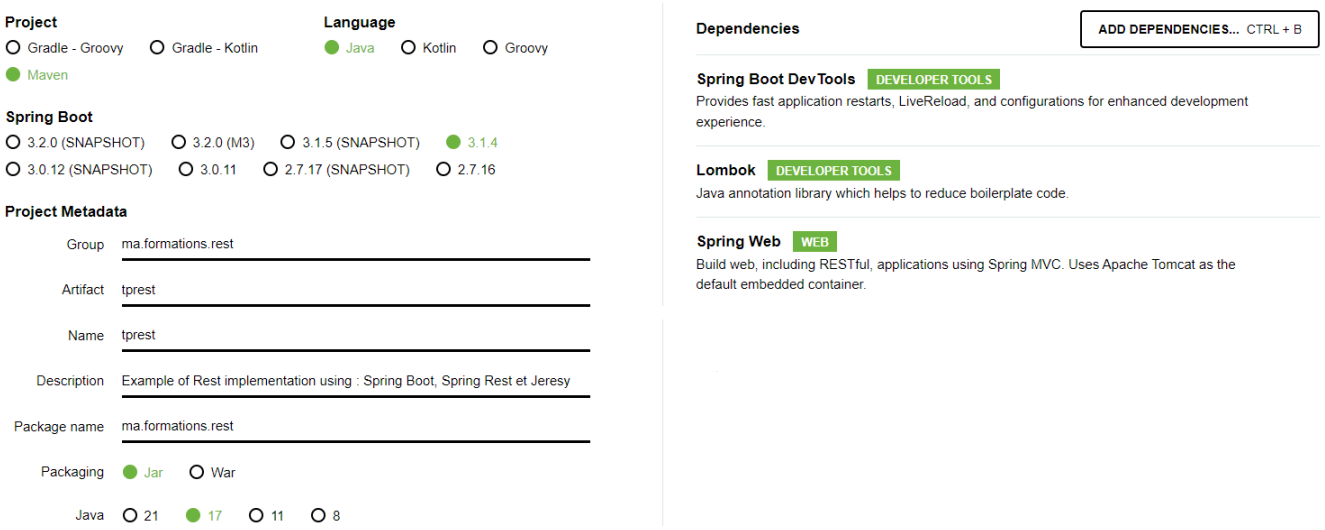
3. Les méthodes HTTP utilisées dans Rest :

- ✓ **GET** : Fournit un accès en lecture seule à une ressource.
- ✓ **PUT** : Utilisée pour modifier une ressource.
- ✓ **PATCH** : utilisée pour modifier un champ d'une ressource.
- ✓ **DELETE** : Utilisée pour supprimer une ressource.
- ✓ **POST** : Utilisée pour créer une nouvelle ressource.
- ✓ **OPTIONS** : Utilisée pour obtenir les opérations supportées par une ressource.

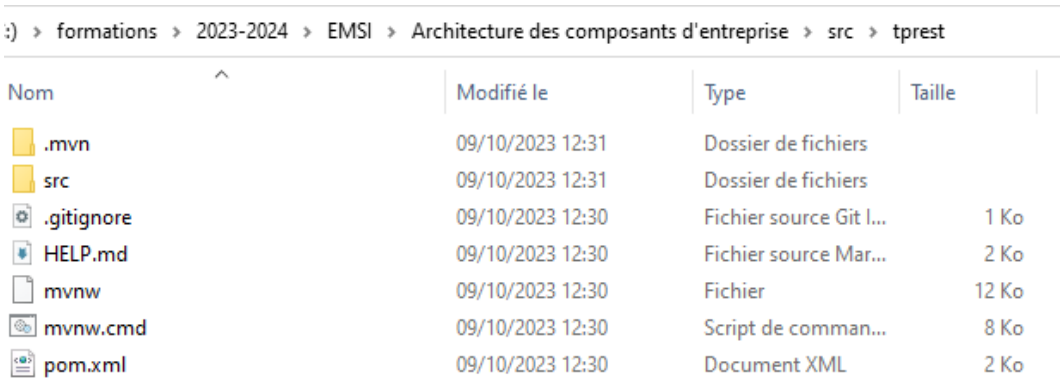
IV. Développement du SW avec Spring Rest

a. Création du projet Maven

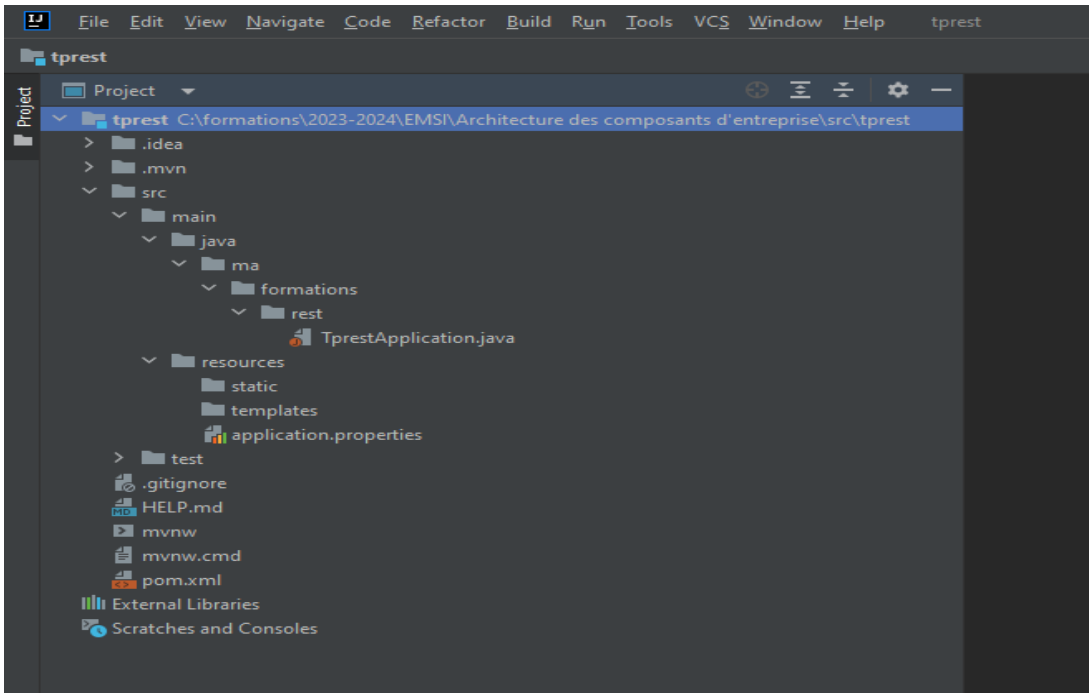
1. Aller au lien https://start.spring.io/, l'interface suivante s'affiche :



- ✓ Cocher java et Maven.
- ✓ Cocher la version 3.1.4 dans Spring Boot.
- ✓ Entrer le Group, l'artifact et le nom du package.
- ✓ Cocher jar dans Packaging.
- ✓ Cocher 17 dans Java.
- ✓ Ajouter les dépendances :
  - Spring Boot Dev Tools ;
  - Lombok ;
  - Spring Web ;
- ✓ Enfin, cliquer sur le bouton « GENERATE » pour générer le projet Maven.
- ✓ Décompresser le ZIP et copier le projet dans votre workspace :



2. Ouvrir le projet dans Intellig :



3. Ajouter la dépendance suivante dans pom.xml :

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

Le contenu global du fichier pom.xml est le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://maven.apache.org/POM/4.0.0"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>3.1.4</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>ma.formations.rest</groupId>
   <artifactId>tprest</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>tprest</name>
```

```
  <description>Example of Rest implementation using : Spring Boot et Spring
Rest</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>com.fasterxml.jackson.dataformat</groupId>
      <artifactId>jackson-dataformat-xml</artifactId>
    </dependency>

  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <excludes>
            <exclude>
              <groupId>org.projectlombok</groupId>
```

```
                <artifactId>lombok</artifactId>
            </exclude>
        </excludes>
    </configuration>
</plugin>
</plugins>
</build>

</project>
```

Explications :

- ❖ La dépendance :

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

- Contient *Tomcat* comme conteneur web embarqué, Spring MVC et Spring Rest.
- Spring propose également ***Undertow*** et ***Jettey*** comme conteneurs web embarqués.

- ❖ La dépendance :

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

- Pour la validation des données du DTO (***Data Transfert Object***). Par défaut, Spring utilise comme implémentation de l'API *Bean Validation*, le Framework *Hibernate Validator*. L'api propose plusieurs annotations comme par exemple : @NotNull, @NotEmpty, @Max, ….

- ❖ La dépendance :

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>
```

- Pour le démarrage automatique de *Tomcat* une fois le code source du projet a été modifié.

- ❖ La dépendance :

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

- Le Starter fourni par Spring Boot pour réaliser les tests unitaires et les tests d'intégrations. Ce starter utilise les Framework suivants : JUNIT, MOCKITO, HAMCREST, …

❖ La dépendance :

```
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

- C'est le Framework Jackson. Ce dernier permet de produire les ressources en format XML (Marshaling). Il est à préciser que Spring supporte uniquement le format JSON.

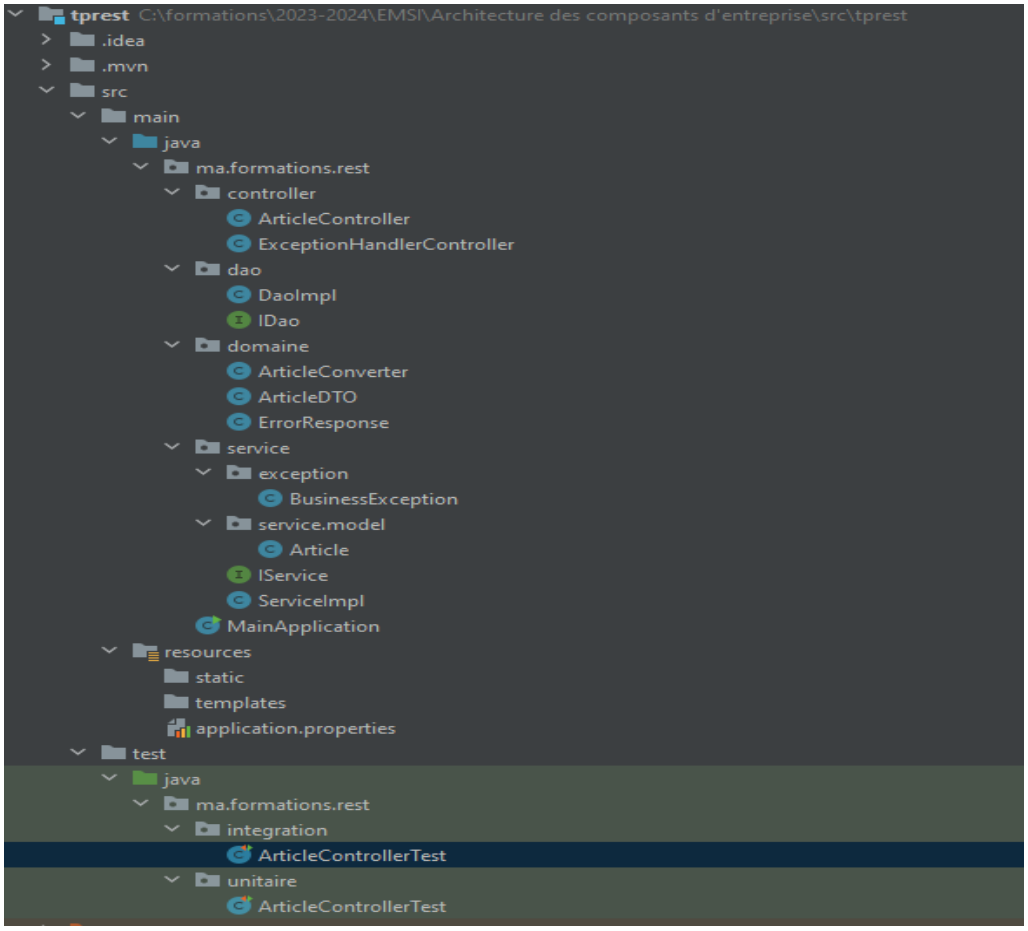
❖ Le Plugin :

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

- Ce plugin permet la compilation du projet et le build (la création de l'exécutable final : le JAR ou bien le WAR).

b. Implémenter les méthodes GET, POST, PUT et DELETE avec Spring Rest

- L'arborescence de votre projet devrait être :



- La classe **MainApplication** :

```
package ma.formations.rest;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MainApplication {

   public static void main(String[] args) {
      SpringApplication.run(MainApplication.class, args);
   }

}
```

Explications :

- C'est la classe de démarrage de *Spring Boot*. Cette classe doit avoir la méthode main pour exécuter l'application *Spring Boot*.

- Elle est annotée par *@SpringBootApplication.* Cette dernière inclut la configuration automatique (Auto-Configuration), l'analyse des composants (component Scan) et le démarrage de *Spring Boot*.

- Si vous ajoutez l'annotation *@SpringBootApplication* à la classe, vous n'avez pas besoin d'ajouter les annotations *@EnableAutoConfiguration*, *@ComponentScan* et *@SpringBootConfiguration*. L'annotation *@SpringBootApplication* inclut toutes les autres annotations.

- Seulement une classe qui doit être annotée par *@SpringBootApplication.*

- Remarquez que le package racine de votre projet est bien *ma.formations.rest*. Spring gère uniquement les Bean qui se trouvent dans ce package ou bien les sous packages de ce dernier.

- Modifier le fichier **application.properties** comme suit :

```
server.port=7777
```

Explication :

- Dans cet exemple, le conteneur web *Tomcat* sera démarré dans le port 7777. Par défaut, le port est 8080.

- Créer la classe **ArticleDTO** suivante :

```
package ma.formations.rest.domaine;

import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.Size;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class ArticleDTO implements Serializable {
   private Long id;
   @Size(min = 1, max = 30, message = "description size must be between 1 and 30")
   private String description;
```

```
    private Double price;
    @Min(value = 1, message = "The quantity value must be greeter than 1")
    private Double quantity;
}
```

Explications :

- Cette classe joue le rôle de DTO (Data Object Transfert). Ce sont les attributs de cette classe qui seront échangés entre la couche Front et notre application.
    - Le DTO permet de réduire le problème de couplage fort. On peut changer les modèles de la couche Backend sans impacter la couche Front.
    - Le DTO permet de ne pas utiliser les objets externes à une couche donnée.

- @Size, @Min sont des annotations de l'api *Bean validation*. Cette dernière fait partie des spécifications de Java EE. Il s'agit ici des règles de gestion concernant les champs « *description* » et « *quantity* ». Il est à préciser que vous pouvez développer vos propres annotations pour implémenter des règles de gestion personnalisées.

- Remarquez le package *jakarta.validation.*.* En effet, les spécifications javax.* ont été supprimées de la JDK 17.

- @Builder est une annotation de Lombok, elle permet de créer un builder afin de créer une instance du Bean en question.

- Créer la classe **ArticleConverter** :

```
package ma.formations.rest.domaine;

import ma.formations.rest.service.service.model.Article;

import java.util.ArrayList;
import java.util.List;

public class ArticleConverter {

    public static Article toBO(ArticleDTO dto) {
        if (dto == null) return null;
        return Article.builder().
                id(dto.getId()).
                description(dto.getDescription()).
                price(dto.getPrice()).
                quantity(dto.getQuantity()).
```

```
            build();
    }

    public static ArticleDTO toDTO(Article bo) {
        if (bo == null) return null;
        return ArticleDTO.builder().
                id(bo.getId()).
                description(bo.getDescription()).
                price(bo.getPrice()).
                quantity(bo.getQuantity()).
                build();
    }

    public static List<Article> toBOs(List<ArticleDTO> dtoList) {
        List<Article> boList = new ArrayList<>();
        dtoList.forEach(a -> boList.add(toBO(a)));
        return boList;
    }

    public static List<ArticleDTO> toDTOs(List<Article> boList) {
        List<ArticleDTO> dtoList = new ArrayList<>();
        boList.forEach(a -> dtoList.add(toDTO(a)));
        return dtoList;
    }
}
```

Explications :

- Cette classe permet d'implémenter le pont DTO<-> BO (Business Object).

- Si vous modifiez le BO, vous allez modifier uniquement les méthodes de cette classe, les méthodes de la classe DTO resteront les mêmes.

- Créer la classe **ErrorResponse** :

```
package ma.formations.rest.domaine;


import lombok.Getter;
import lombok.Setter;

import java.util.List;

@Getter
@Setter
public class ErrorResponse {
```

```
    private String message;
    private List<String> details;

    public ErrorResponse(String message, List<String> details) {
        super();
        this.message = message;
        this.details = details;
    }

}
```

Explications :

- Cette classe permet de personnaliser les messages d'erreurs que nous allons transmettre comme réponse au client au cas où une exception a été levée ou bien au cas où une règle de gestion n'a pas été respectée.

- Créer la classe **Article** :

```
package ma.formations.rest.service.service.model;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Article implements Serializable {
    private Long id;
    private String description;
    private Double price;
    private Double quantity;
}
```

- L'interface **IDao** :

```
package ma.formations.rest.dao;

import ma.formations.rest.service.service.model.Article;

import java.util.List;
```

```
public interface IDao {
   Article findById(Long id);
   List<Article> findAll();
   void save(Article article);
   void deleteById(Long id);
}
```

- La classe **DaoImpl** :

```
package ma.formations.rest.dao;

import ma.formations.rest.service.service.model.Article;
import org.springframework.stereotype.Repository;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

@Repository
public class DaoImpl implements IDao {
   private static final List<Article> database = new ArrayList<>
         (Arrays.asList(
               new Article(1L, "PC PORTABLE HP I7", 15000d, 10d),
               new Article(2L, "ECRAN", 1500d, 10d),
               new Article(3L, "CAMERA LG", 3000d, 10d),
               new Article(4L, "SOURIS", 200d, 10d)));

   @Override
   public Article findById(Long id) {
      return database.stream().filter(a -> a.getId().equals(id)).findFirst().orElse(null);
   }

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

   @Override
   public void save(Article article) {
      database.add(article);
   }

   @Override
   public void deleteById(Long id) {
```

```
        database.remove(database.stream().filter(a -> id.equals(a.getId())).findFirst());
    }
}
```

- Créer la classe **BusinessException** :

```
package ma.formations.rest.service.exception;

public class BusinessException extends RuntimeException {
    public BusinessException(String s) {
        super(s);
    }
}
```

- L'interface **IService** :

```
package ma.formations.rest.service;

import ma.formations.rest.domaine.ArticleDTO;

import java.util.List;

public interface IService {
    ArticleDTO getById(Long id);

    List<ArticleDTO> getAll();

    void create(ArticleDTO article);

    void update(Long id, ArticleDTO article);

    void deleteById(Long id);
}
```

- La classe **ServiceImpl** :

```
package ma.formations.rest.service;

import lombok.AllArgsConstructor;
import ma.formations.rest.dao.IDao;
import ma.formations.rest.domaine.ArticleConverter;
import ma.formations.rest.domaine.ArticleDTO;
import ma.formations.rest.service.exception.BusinessException;
import ma.formations.rest.service.service.model.Article;
```

```java
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@AllArgsConstructor
public class ServiceImpl implements IService {
   private IDao dao;

   @Override
   public ArticleDTO getById(Long id) {
      Article articleFound = dao.findById(id);
      if (articleFound == null) {
         throw new BusinessException(String.format("no article with id= %s exist", id));
      }
      return ArticleConverter.toDTO(articleFound);
   }

   @Override
   public List<ArticleDTO> getAll() {
      return ArticleConverter.toDTOs(dao.findAll());
   }

   @Override
   public void create(ArticleDTO article) {
      if (article.getId() == null) {
         dao.save(ArticleConverter.toBO(article));
         return;
      }
      Article articleFound = dao.findAll().stream().
            filter(a -> article.getId().equals(a.getId())).
            findFirst().orElse(null);

      if (articleFound != null)
         throw new BusinessException(String.format("Article with the same Id=%s exist in
database", article.getId()));

      dao.save(ArticleConverter.toBO(article));
   }

   @Override
   public void update(Long id, ArticleDTO article) {
      if (id == null)
         throw new BusinessException(String.format("Article id=%s should not be null", id));

      dao.findAll().stream().
            filter(a -> a.getId().equals(id)).
```

```
            findFirst().orElseThrow(() -> new BusinessException(String.format("No article with the
id=%s exist in database", id)));

        article.setId(id);
        dao.save(ArticleConverter.toBO(article));

    }

    @Override
    public void deleteById(Long id) {
        Article articleFound = dao.findById(id);
        if (articleFound == null) {
            throw new BusinessException(String.format("no article with id= %s exist", id));
        }
        dao.deleteById(id);
    }
}
```

- Créer la classe **ArticleController** :

```
package ma.formations.rest.controller;

import jakarta.validation.Valid;
import lombok.AllArgsConstructor;
import ma.formations.rest.domaine.ArticleDTO;
import ma.formations.rest.service.IService;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController()
@RequestMapping("/api/articles")
@AllArgsConstructor
public class ArticleController {

    private IService service;

    @GetMapping(value = "/all", produces = {MediaType.APPLICATION_XML_VALUE,
MediaType.APPLICATION_JSON_VALUE})
    public List<ArticleDTO> getAll() {
        return service.getAll();
    }
```

```
    //answer to this  url : http://localhost:7777/api/articles/id/1
    @GetMapping(value = "/id/{id}")
    public ResponseEntity<Object> getArticleById(@PathVariable(value = "id") Long id) {
        ArticleDTO articleFound = service.getById(id);
        return new ResponseEntity<>(articleFound, HttpStatus.OK);
    }

    //answer to this  url : http://localhost:7777/api/articles?id=1
    @GetMapping
    public ResponseEntity<Object> getArticleByIdUsingParam(@RequestParam(value = "id")
Long id) {
        ArticleDTO articleFound = service.getById(id);
        return new ResponseEntity<>(articleFound, HttpStatus.OK);
    }

    @PostMapping(value = "/create")
    public ResponseEntity<Object> createArticle(@Valid @RequestBody ArticleDTO dto) {
        service.create(dto);
        return new ResponseEntity<>("Article is created successfully", HttpStatus.CREATED);
    }

    @PutMapping(value = "/update/{id}")
    public ResponseEntity<Object> updateArticle(@PathVariable(name = "id") Long id,
@RequestBody ArticleDTO dto) {
        service.update(id, dto);
        return new ResponseEntity<>("Article is updated successfully", HttpStatus.OK);
    }


    @DeleteMapping(value = "/delete/{id}")
    public ResponseEntity<Object> deleteArticle(@PathVariable(name = "id") Long id) {
        service.deleteById(id);
        return new ResponseEntity<>("Article is deleted successfully", HttpStatus.OK);
    }
}
```

Explications :

- L'annotation *@Valid* est fournie par le starter *spring-boot-starter-validation* qui utilise *Hibernate Validator* comme implémentation de l'api Bean Validation.

- @Valid permet à Spring de vérifier si les données envoyées dans le DTO respectent les règles de gestion configurées moyennant les annotations @NotNul, @Size, …

- Créer la classe **ExceptionHandlerController :**

```java
package ma.formations.rest.controller;


import ma.formations.rest.domaine.ErrorResponse;
import ma.formations.rest.service.exception.BusinessException;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.HttpStatusCode;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.ObjectError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

import java.util.ArrayList;
import java.util.List;

@ControllerAdvice
public class ExceptionHandlerController extends ResponseEntityExceptionHandler {

   @Override
   protected ResponseEntity<Object>
handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders
headers, HttpStatusCode status, WebRequest request) {
      List<String> details = new ArrayList<>();
      for (ObjectError error : ex.getBindingResult().getAllErrors()) {
         details.add(error.getDefaultMessage());
      }
      ErrorResponse error = new ErrorResponse("Validation Failed", details);
      return new ResponseEntity(error, HttpStatus.BAD_REQUEST);
   }

   @ExceptionHandler(value = BusinessException.class)
   public final ResponseEntity<Object> handleBusinessException(BusinessException ex,
WebRequest request) {
      List<String> details = new ArrayList<>();
      details.add(ex.getLocalizedMessage());
      ErrorResponse error = new ErrorResponse("Functional errors", details);
      return new ResponseEntity(error, HttpStatus.BAD_REQUEST);
   }

   @ExceptionHandler(Exception.class)
```

```
    public final ResponseEntity<Object> handleOtherExceptions(Exception ex, WebRequest
request) {
        List<String> details = new ArrayList<>();
        details.add(ex.getMessage());
        ErrorResponse error = new ErrorResponse("Technical error, please consult your
administrator", details);
        return new ResponseEntity(error, HttpStatus.INTERNAL_SERVER_ERROR);
    }

}
```

Explications :

- L'annotation *@ControllerAdvice* permet à Spring de créer un contrôleur qui va intercepter les exceptions levées par l'application. Le type d'exception à traiter est précisé en paramètre de l'annotation *@ExceptionHandler*.

- Dans cet exemple, la méthode *handleBusinessException(..)* traite les exceptions de type *BusinessException* et la méthode *handleOtherExceptions(..)* traite les autres exceptions.

- Remarquez que nous avons hérité de la classe *ResponseEntityExceptionHandler* et ceci afin de redéfinir la méthode *handleMethodArgumentNotValid.* Cette dernière permet de traiter les exceptions levées par l'api Bean Validation.

- Avec l'annotation *@ControllerAdvice*, *Spring* implémente le Design Pattern AOP. Il permet de séparer le traitement technique relatif à la gestion des exceptions du code métier de l'application.

V. Les tests

a. Développement des cas de test

1. Les tests unitaires

- Créer la classe de test suivante :

```
package ma.formations.rest.unitaire;

import ma.formations.rest.domaine.ArticleDTO;
import ma.formations.rest.service.IService;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
```

```
import org.springframework.test.web.servlet.MockMvc;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@AutoConfigureMockMvc
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class ArticleControllerTest {
    @Autowired
    private MockMvc mvc;

    @MockBean
    private IService service;

    @Test
    void testGetAll() throws Exception {
        List<ArticleDTO> articles = new ArrayList<>
                (Arrays.asList(
                        new ArticleDTO(1L, "PC PORTABLE HP I7", 15000d, 10d),
                        new ArticleDTO(2L, "ECRAN", 1500d, 10d),
                        new ArticleDTO(3L, "CAMERA LG", 3000d, 10d),
                        new ArticleDTO(4L, "SOURIS", 200d, 10d)));


        when(service.getAll()).thenReturn(articles);

mvc.perform(get("/api/articles/all").contentType(MediaType.APPLICATION_JSON).accept(Med
iaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$[0].id").value(1L))
                .andExpect(jsonPath("$[1].description").value("ECRAN"))
                .andExpect(jsonPath("$[2].price").value(3000d))
                .andExpect(jsonPath("$[3].quantity").value(10d));
    }
}
```

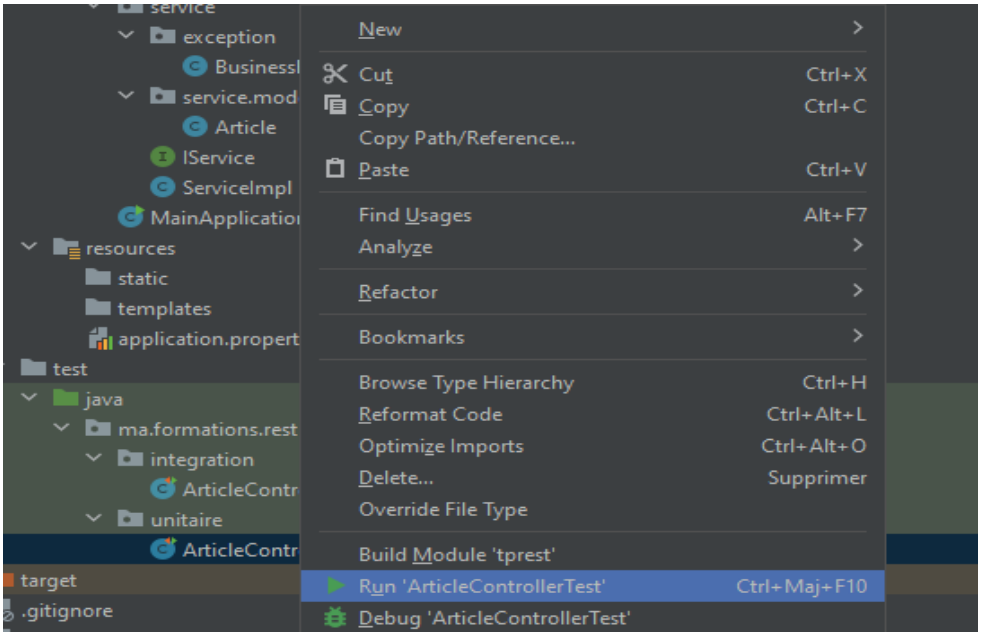
Explications :

- L'annotation ***@SpringBootTest*** permet de créer le contexte de votre application pour pouvoir effectuer les tests unitaires et les tests d'intégration. Cette dernière permet d'activer des fonctionnalités supplémentaires telles que des propriétés d'environnement personnalisées, différents modes d'environnement Web, des ports aléatoires, des Bean TestRestTemplate et WebTestClient.

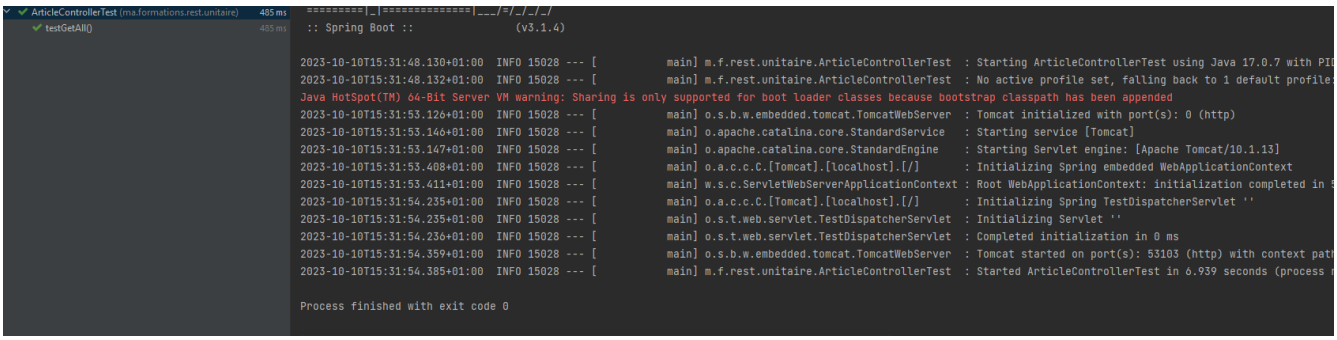
- L'annotation *@AutoConfigureMockMvc* permet de configurer l'objet MockMvc.

- Remarquer que la variable service est annotée par @MockBean. Il s'agit d'un objet mock (un objet fictif). Le Framework Mockito offre certains services très pratiques dans les tests unitaires. Par exemple, Mockito permet de simuler le résultat de retour d'une méthode.

- Exécuter ce test comme illustré ci-après :



- Vérifier que le test est concluant :



2. Les tests d'intégration

- Créer la classe de test suivante :

```
package ma.formations.rest.integration;

import ma.formations.rest.domaine.ArticleDTO;
import ma.formations.rest.service.IService;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.boot.test.web.server.LocalServerPort;
import org.springframework.http.*;

import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;

@AutoConfigureMockMvc
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class ArticleControllerTest {
    @Autowired
    private IService service;

    @Autowired
    private TestRestTemplate restTemplate;

    @LocalServerPort
    private int port;

    @Test
    void testGetById() throws Exception {

        ArticleDTO article = new ArticleDTO(1L, "PC PORTABLE HP I7", 15000d, 10d);

        Long id = 1L;
        HttpHeaders headers = new HttpHeaders();
        headers.setAccept(List.of(new MediaType[]{MediaType.APPLICATION_JSON}));

        HttpEntity<ArticleDTO> entity = new HttpEntity<ArticleDTO>(headers);
        ResponseEntity<ArticleDTO> result = this.restTemplate.exchange("http://localhost:" +
port + "/api/articles/id/" + article.getId(), HttpMethod.GET,
                entity, ArticleDTO.class);
        assertThat(result).isNotNull();
        ArticleDTO dto = result.getBody();
        assertThat(dto.getId()).isEqualTo(article.getId());
        assertThat(dto.getDescription()).isEqualTo(article.getDescription());
```

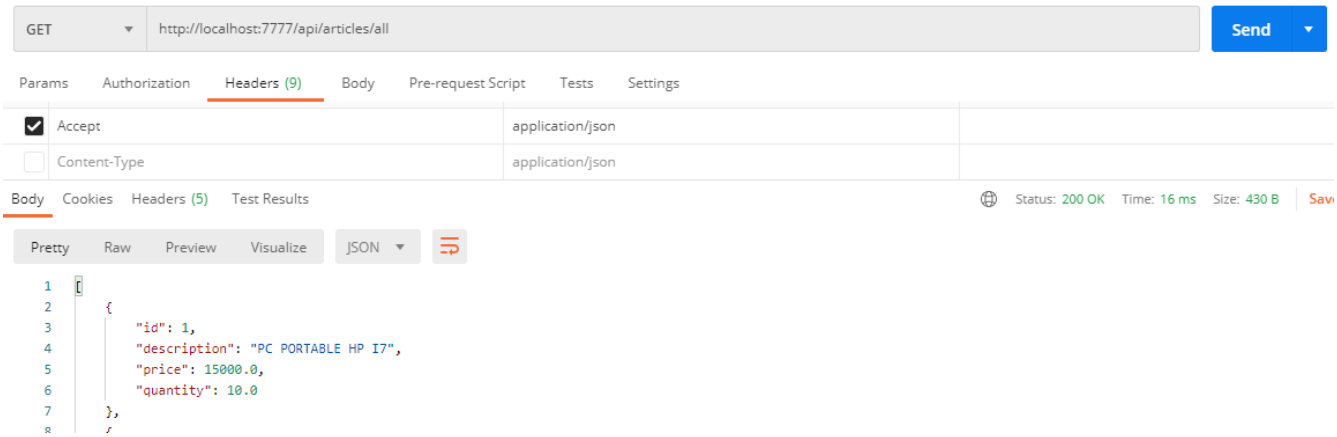
```
        assertThat(dto.getPrice()).isEqualTo(article.getPrice());
        assertThat(dto.getQuantity()).isEqualTo(article.getQuantity());
    }

}
```

Explications :

- Remarquez que nous avons annoté les variables d'instance de la classe par @Autowired au lieu de @MockBean. En effet, l'objectif est d'effectuer les test d'intégration de bout en bout.
- La classe *TestRestTemplate* offre la méthode *exchange(..)* pour tester les méthodes Rest exposées par notre service web.

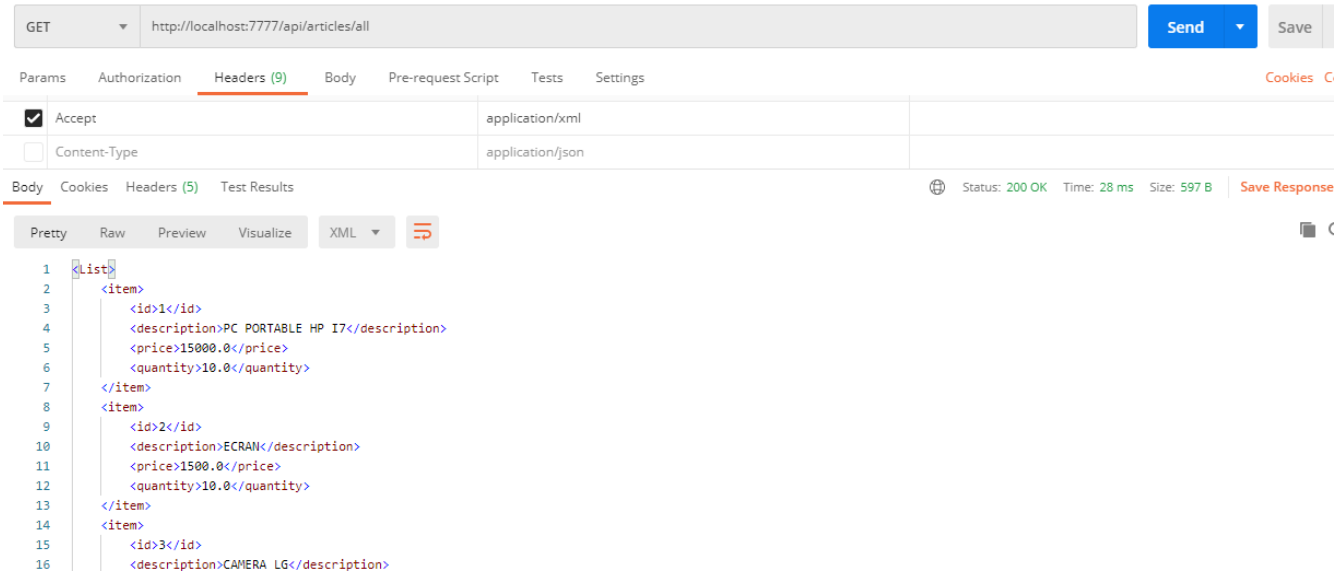
b. Les tests avec Postman

- Démarrer votre application en exécutant la méthode main de la classe MainApplication.
- Lancer postman.
  - ❖ Tester l'URI http://localhost:7777/api/articles/all et GET :
- Tester l'URI http://localhost:7777/api/articles/all avec la méthode GET comme le montre la fenêtre suivante :

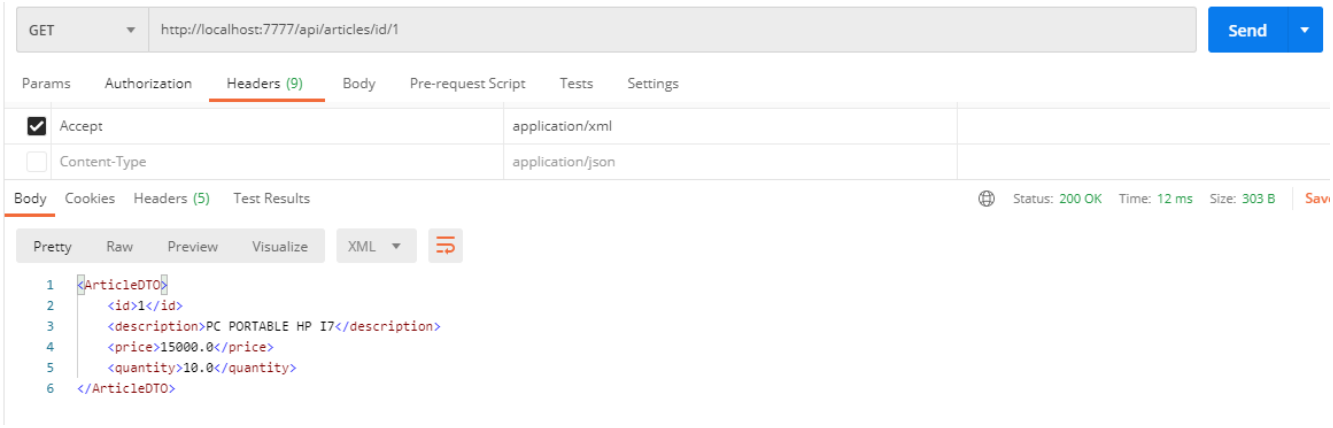


  - ✓ Entrer l'uri http://localhost:7777/api/articles/all.
  - ✓ Dans Headers, préciser *application/json* dans *Accept*.
  - ✓ Cliquer sur *Send* et vérifier que le status est 200 et que le résultat a été communiqué en format JSON.

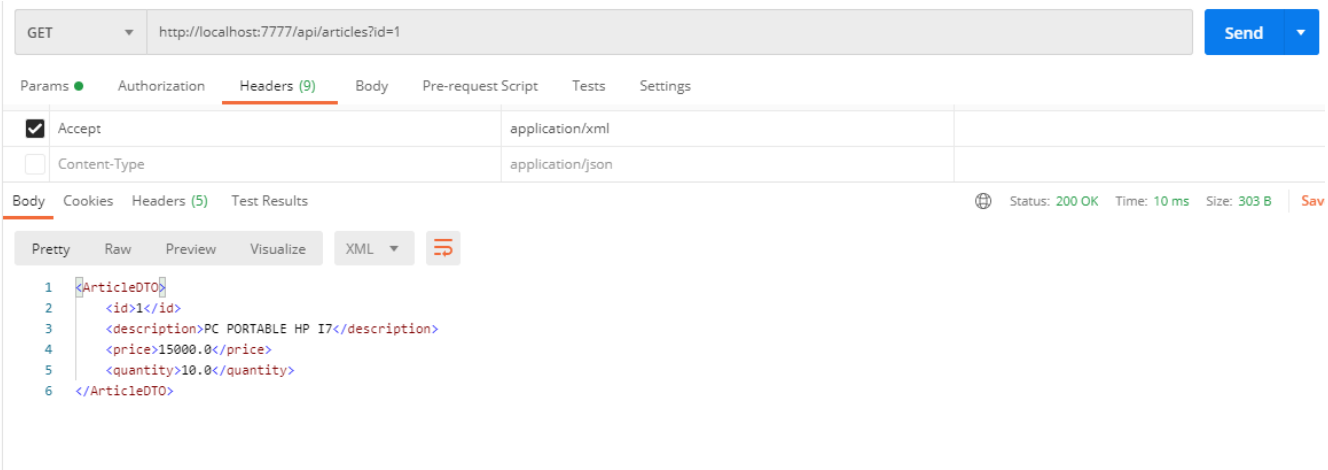
- Refaire le même test en précisant *application/xml* dans *Accept*. Cliquer sur *Send* et vérifier que le statut est 200 et que le résultat a été communiqué en format XML comme le montre l'écran suivant :



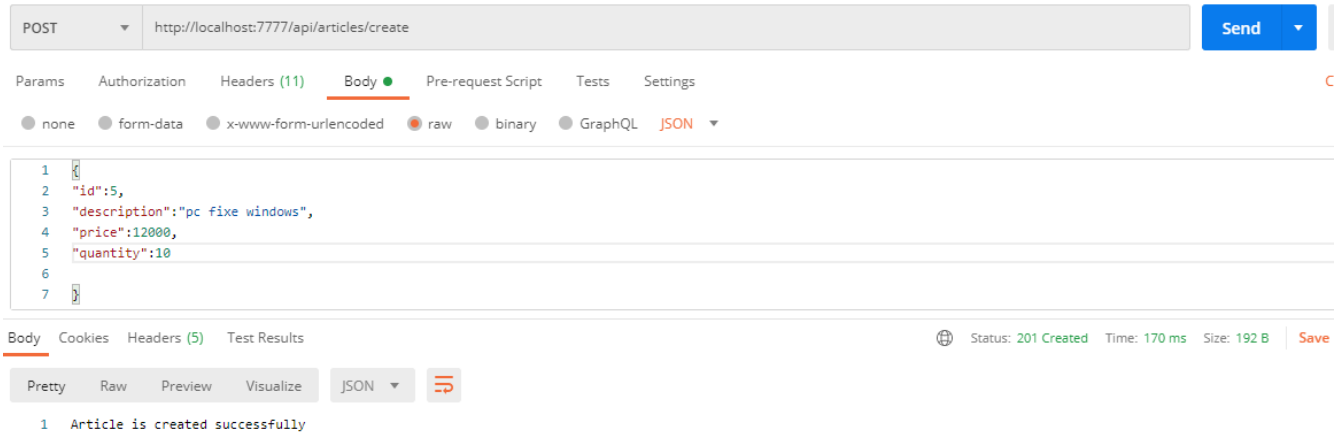
❖ Refaire le même test avec l'URI http://localhost:7777/api/articles/id/1 et GET. Vous devriez avoir le résultat suivant :



❖ Refaire le même test avec l'URI http://localhost:7777/api/articles?id=1 et GET. Vous devriez avoir le résultat suivant :



❖ Tester l'URI http://localhost:7777/api/articles/create et POST avec des données correctes :

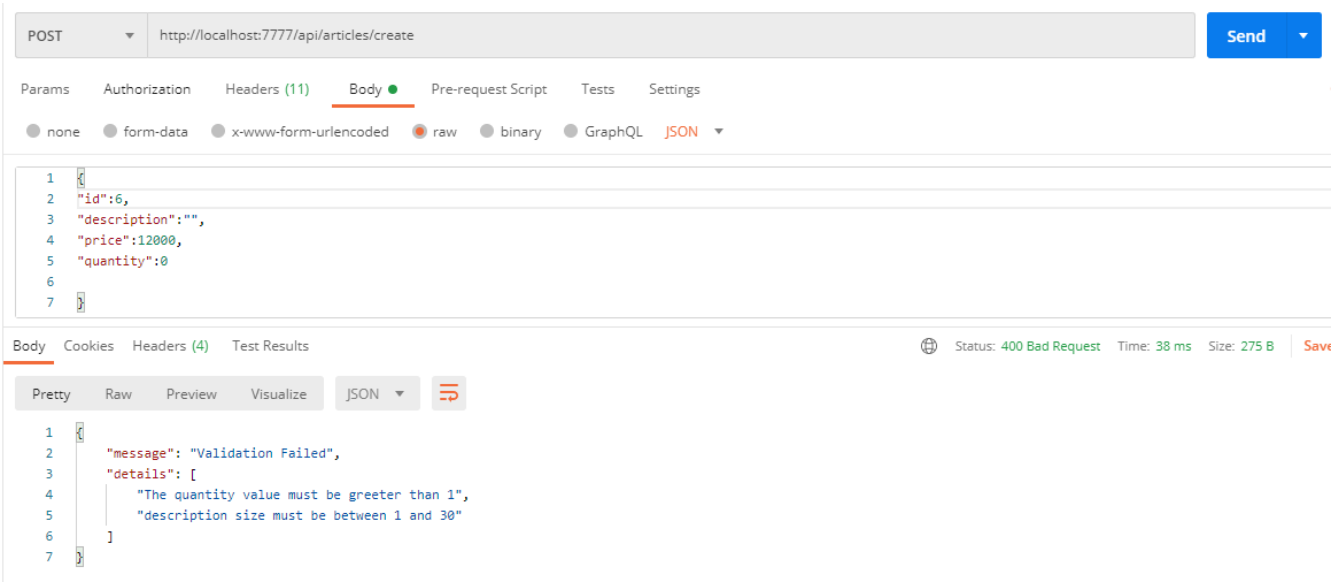


✓ Entrer l'uri http://localhost:7777/api/articles/create.
✓ Dans Headers, préciser *application/json* dans *Accept*.
✓ Dans Body, saisir le message json de l'article à créer :

```
{
"id":5,
"description":"pc fixe windows",
"price":12000,
"quantity":10

}
```

✓ Cliquer sur *Send* et vérifier que le statut est 201 (Created).

❖ Tester l'URI http://localhost:7777/api/articles/create et POST avec des données incorrectes :

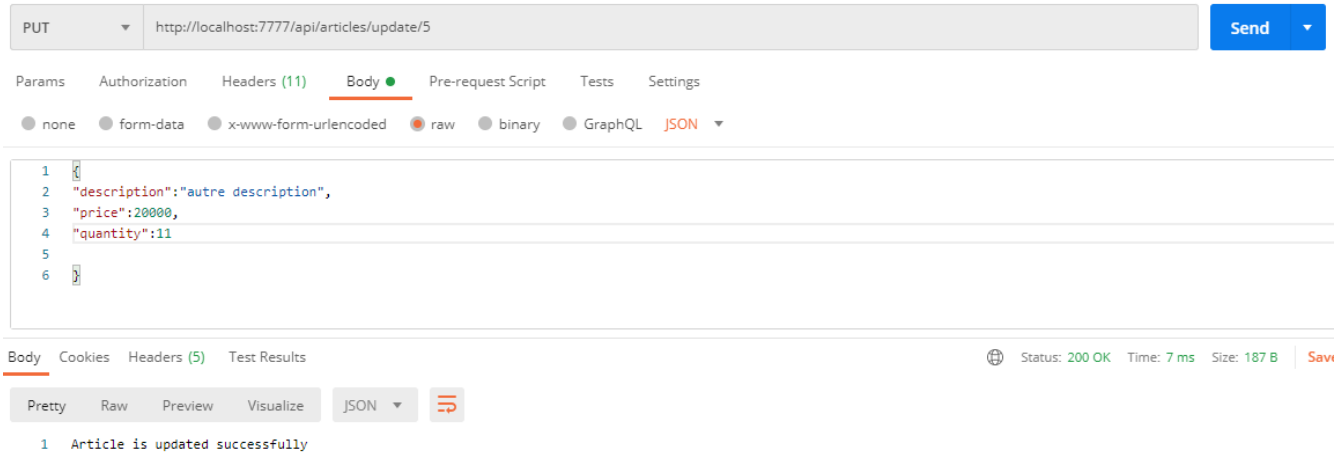


- ✓ Entrer l'uri http://localhost:7777/api/articles/create.
- ✓ Dans Headers, préciser *application/json* dans *Accept*.
- ✓ Dans Body, saisir le message json de l'article à créer :

```
{
"id":6,
"description":"",
"price":12000,
"quantity":0
}
```

- ✓ Cliquer sur *Send* et vérifier que le status est 400 (Bad Request). Vérifier le message envoyé par le serveur. Il s'agit ici de la classe ErrorResponse qui est traité par le contrôleur *ExceptionHandlerController*.

❖ Tester l'URI http://localhost:7777/api/articles/update/1 et PUT :



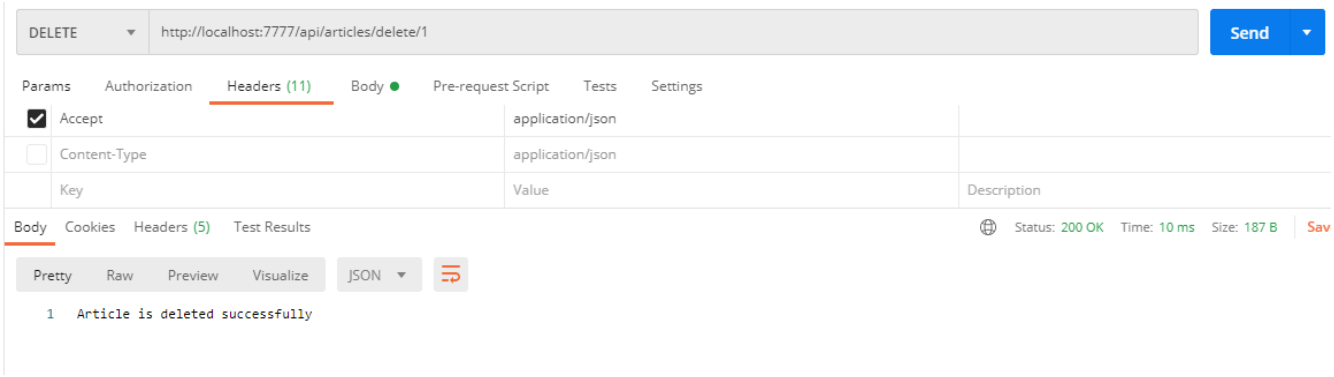
- ✓ Entrer l'uri http://localhost:7777/api/articles/update/5.

- ✓ Dans Headers, préciser *application/json* dans *Accept*.
- ✓ Dans Body, saisir le message json de l'article à créer :

```
{
"description":" autre description",
"price":20000,
"quantity":11
}
```

- ✓ Cliquer sur *Send* et vérifier que le status est 200. Vérifier le message envoyé par le serveur : « Article is updated successfully ».

❖ Tester l'URI http://localhost:7777/api/articles/dekete/1 et DELETE :

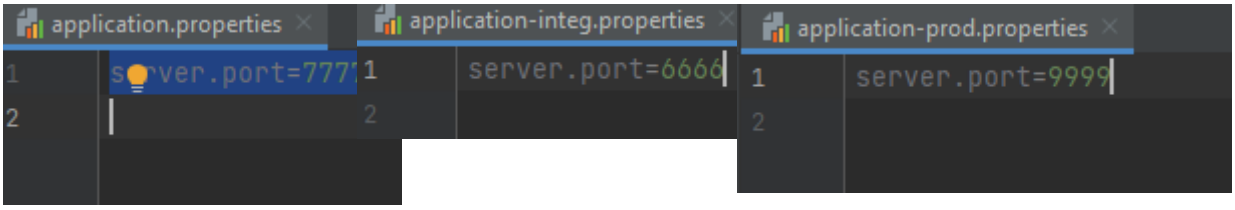


- ✓ Entrer l'uri http://localhost:7777/api/articles/delete/1.
- ✓ Dans Headers, préciser *application/json* dans *Accept*.
- ✓ Cliquer sur *Send* et vérifier que le status est 200. Vérifier le message envoyé par le serveur : « Article is deleted successfully ».

VI. Déploiement de l'application

1. Spring Active Profile

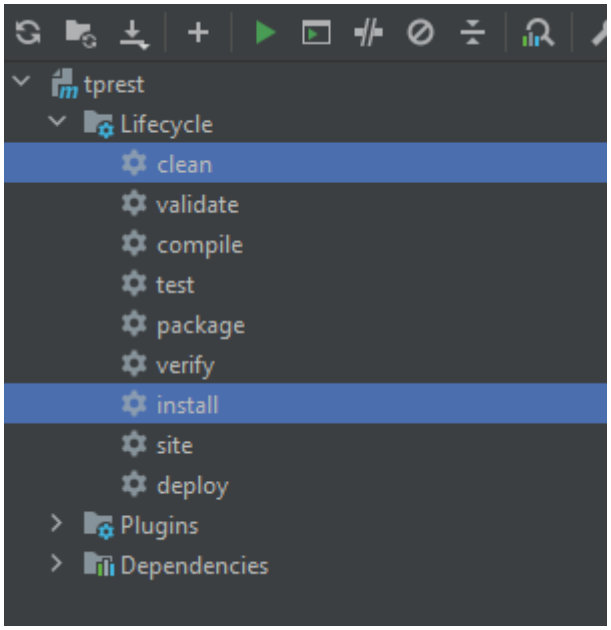
- Vous pouvez créer plusieurs profiles au niveau de votre application. Chaque profile est identifié par un fichier ***application_XXX.properties***. Par exemple :
  - application-integ.properties pour exécuter l'application en environnement d'intégration.
  - application-prod.properties pour exécuter l'application en environnement de production.
  - application.properties pour exécuter l'application en environnement de développement (l'exécution par défaut) :



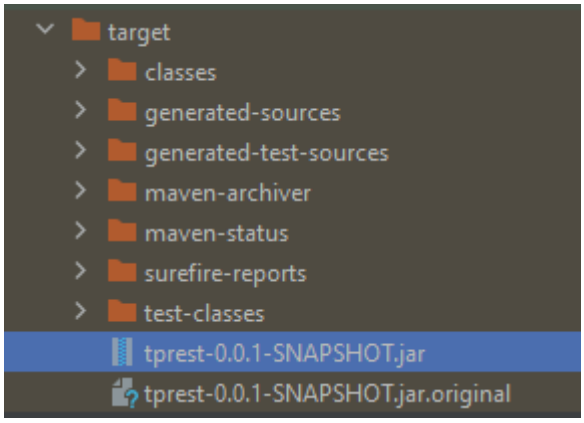
- Dans l'exemble ci-dessous, nous avons paramétré uniquement le numéro de port.

2. Création du JAR

- Suivre les étapes suivantes pour créer le JAR :
  - ✓ Lancer la commande ***clean install*** *de Maven* comme illustré ci-après :



Vérifier que le fichier JAR a été bien généré :



- Pour exécuter l'application en utilisant le conteneur web Tomcat embarqué au niveau du JAR, lancer la commande suivante :

```
C:\Users\abbou>java -jar "C:\formations\2023-2024\EMSI\Architecture des composants d'entreprise\src\tprest\target\tprest-0.0.1-SNAPSHOT.jar"
```

- Vous pouvez exécuter l'application en démarrant Tomcat sur un autre port :

```
C:\Users\abbou>java -jar "C:\formations\2023-2024\EMSI\Architecture des composants d'entreprise\src\tprest\target\tprest-0.0.1-SNAPSHOT.jar" --server.port=8888
```

- Pour exécuter l'application par profile (ici le profile PROD), lancez la commande suivante :

```
C:\Users\abbou>java -jar "C:\formations\2023-2024\EMSI\Architecture des composants d'entreprise\src\tprest\target\tprest-0.0.1-SNAPSHOT.jar" --spring.profiles.active=prod
```

Remarquez que Tomcat a été démarré sur le port 9999.

3. Utiliser un fichier de configuration externe

- Créer un nouveau fichier application.properties.
- Lancer la commande suivante :

```
C:\Users\abbou>java -jar -Dspring.config.location=C:\Users\abbou\OneDrive\Bureau\application.properties -jar "C:\formations\2023-2024\EMSI\Architecture des composants d'entreprise\src\tprest\target\tprest-0.0.1-SNAPSHOT.jar"
```

Observer le flag ***-Dspring.config.location***.

4. Création du WAR

- Pour déployer votre application sur un serveur d'application existant ou sur un conteneur web existant, il faut tout d'abord créer le WAR. Pour ceci, suivre les étapes suivantes :
  - ✓ Modifier le fichier *pom.xml* comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://maven.apache.org/POM/4.0.0"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.1.4</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>ma.formations.rest</groupId>
  <artifactId>tprest</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>tprest</name>
  <packaging>war</packaging>
  <description>Example of Rest implementation using : Spring Boot, Spring Rest et
```

```
Jeresy</description>
  <properties>
    <java.version>17</java.version>
    <start-class>ma.formations.rest.MainApplication</start-class>
  </properties>


  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-jersey</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>com.fasterxml.jackson.dataformat</groupId>
      <artifactId>jackson-dataformat-xml</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-tomcat</artifactId>
      <scope>provided</scope>
    </dependency>
```

```
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <excludes>
            <exclude>
              <groupId>org.projectlombok</groupId>
              <artifactId>lombok</artifactId>
            </exclude>
          </excludes>
        </configuration>
      </plugin>
    </plugins>
  </build>

</project>
```

Explications :

- ✓ Les modifications sont en surbrillance.

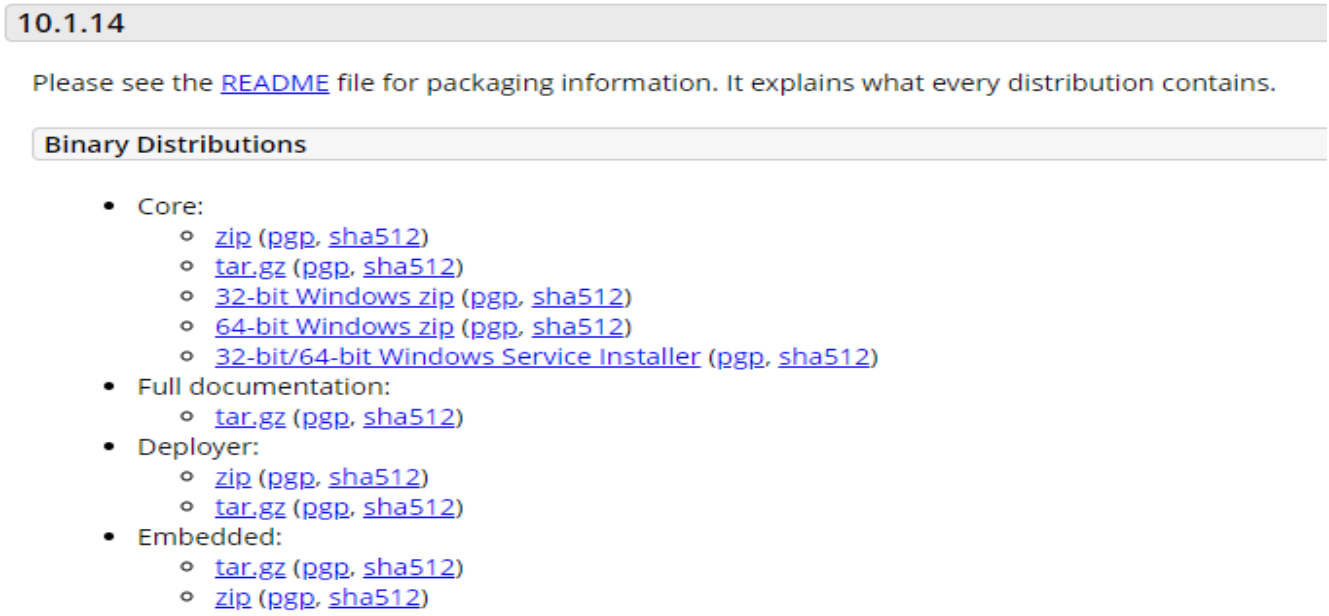
- Modifier la classe *MainApplication* comme suit :

```
package ma.formations.rest;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication
public class MainApplication extends SpringBootServletInitializer {
   @Override
   protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
      return application.sources(MainApplication.class);
   }
}
```

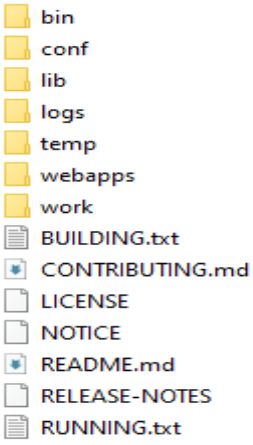
- Lancer la commande Maven clean install et vérifier que le fichier WAR a été bien généré.

5. Configuration d'Apache Tomcat 10

❖ ***Apache Tomcat version 10.1.14*** est téléchargeable sur le site :
https://tomcat.apache.org/download-10.cgi :



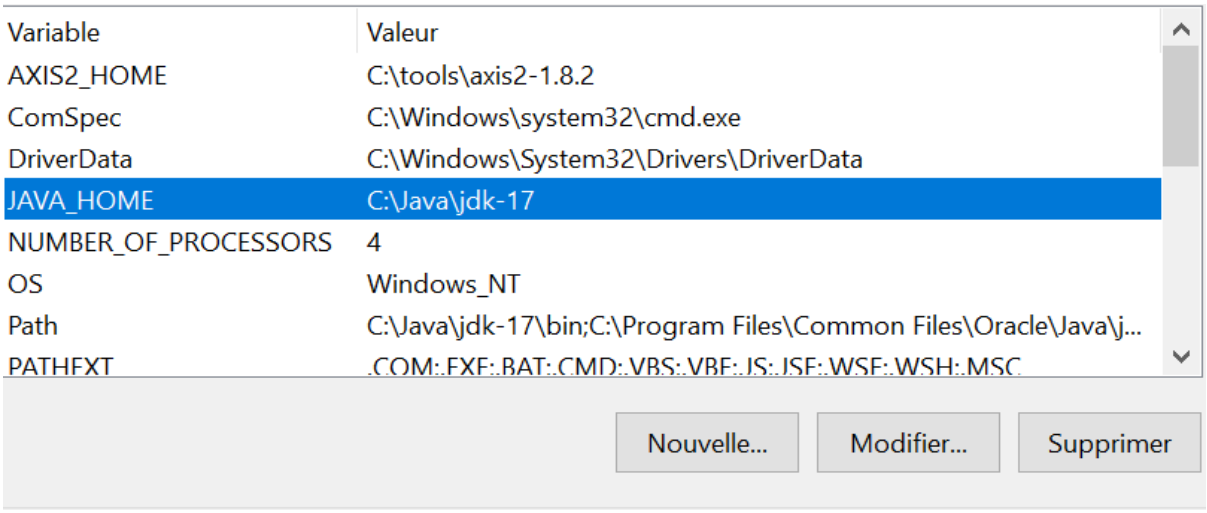
Vous pouvez télécharger la version ZIP comme illustré dans l'écran ci-dessus.
Une fois téléchargé, décompresser le fichier et copier le dossier sur par exemple ***c:\serveurs\apache-tomcat-10.1.14*** :



NB :

- La version 10 de Tomcat est compatible avec JDK 17.
- Le serveur Tomcat doit être mis dans un dossier ne contenant pas des espaces.

❖ Ajouter la variable d'environnement ***JAVA_HOME*** qui doit avoir comme valeur **le chemin racine de votre JDK**. Voir écran suivant pour plus de précisions :



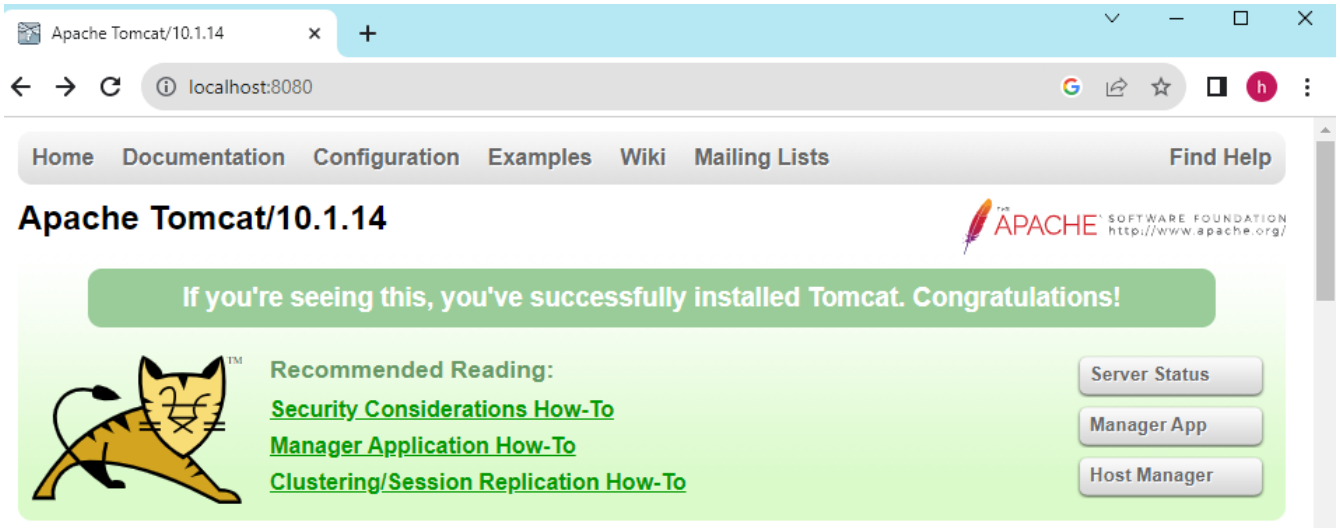
Cette variable est obligatoire pour pouvoir démarrer Tomcat via le lancement du script ci-après : ***[TOMCAT_PATH]\bin\startup.bat***

❖ Ajouter un compte administrateur. Pour ceci, ouvrir le fichier ***[TOMCAT_PATH]\conf\tomcat-users.xml*** et ajouter les lignes suivantes :

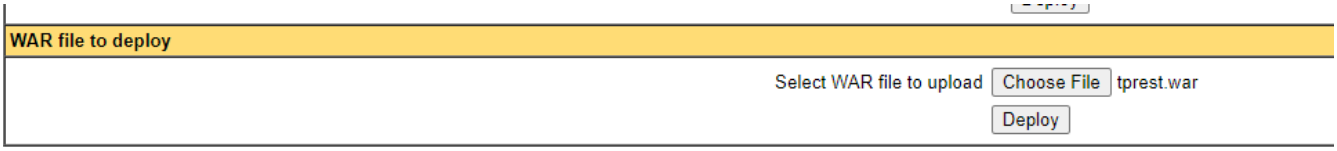
```
<role rolename="manager-gui"/>
<user password="admin" roles="manager-gui" username="admin"/>
```

6. Déploiement du fichier WAR dans Apache Tomcat

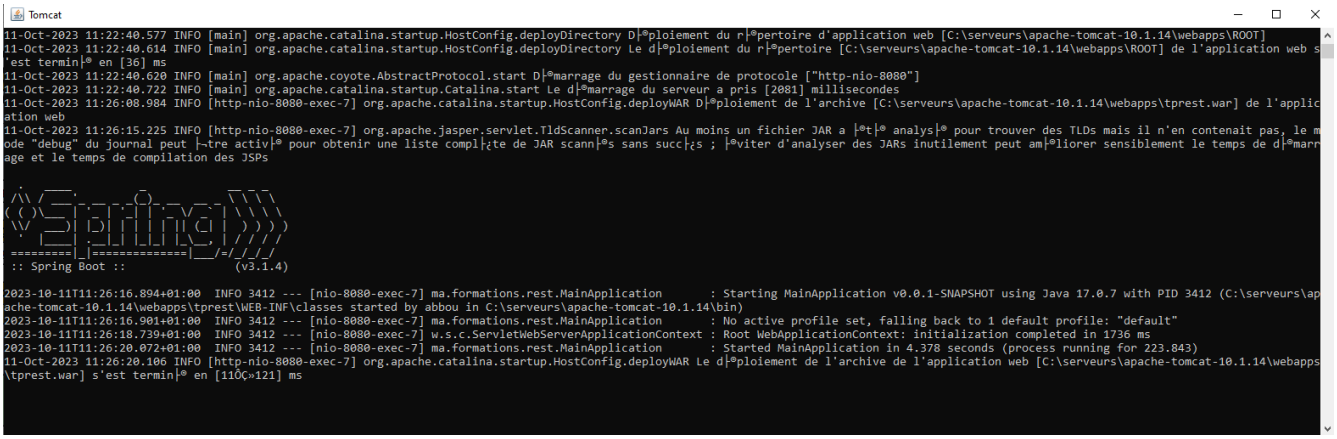
- Lancer le script : [TOMCAT_PATH]\\bin\startup.bat.
- Accéder ensuite au lien http://localhost:8080/:



- Cliquer sur le bouton ***« Manager App »*** ci-dessus :



- Dans la section « *WAR file to deploy* », cliquer sur « *Choose File* », sélectionner le fichier WAR à dépolyer et enfin cliquer sur *Deploy*.
- Au niveau de la console, vérifier que la classe de démarrage de SPRING boot a été bien exécutée :



- Tester l'URI : http://localhost:8080/tprest/api/articles/all et vérrifier que la ressource a été bien envoyée par le serveur :