

République Tunisienne Ministère
De l'Enseignement Supérieur
Et de la Recherche Scientifique

Université De Sfax

École Nationale d'Électronique
Et des Télécommunications de Sfax



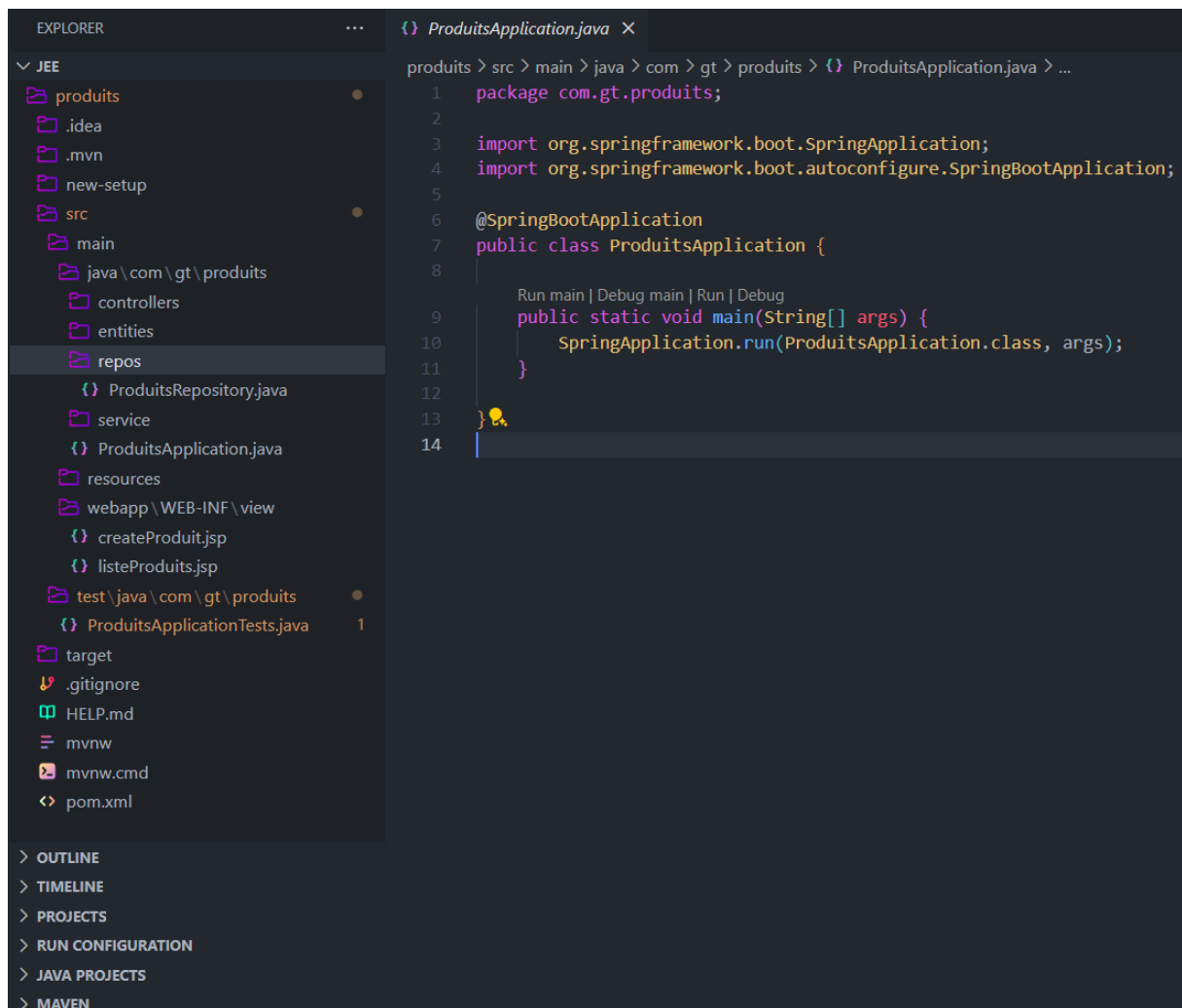
Atelier 1: SPRING BOOT

Réalisé par : Wissem Karous

Remis à : Mme.Kallel Emna

Atelier 1 :

Architecture du projet :



The screenshot shows an IDE with a project named 'JEE'. The 'EXPLORER' view on the left displays the project structure:

- products
- .idea
- .mvn
- new-setup
- src
 - main
 - java\com\gt\produits
 - controllers
 - entities
 - repos
 - service
 - resources
 - webapp\WEB-INF\view
 - createProduit.jsp
 - listeProduits.jsp
 - test\java\com\gt\produits
 - ProduitsApplicationTests.java
- target
- .gitignore
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml

The 'ProduitsApplication.java' file is open in the editor, showing the following code:

```
1 package com.gt.produits;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class ProduitsApplication {
8
9     Run main | Debug main | Run | Debug
10    public static void main(String[] args) {
11        SpringApplication.run(ProduitsApplication.class, args);
12    }
13 }
14
```

➔ Ce code représente la classe principale d'une application Spring Boot, qui lance l'application en utilisant la méthode `run()` de la classe `SpringApplication`.

```
{ } ProduitsApplication.java X
produits > src > main > java > com > gt > produits > { } ProduitsApplication.java > ...
1  package com.gt.produits;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class ProduitsApplication {
8
9      Run main | Debug main | Run | Debug
10     public static void main(String[] args) {
11         SpringApplication.run(ProduitsApplication.class, args);
12     }
13 }
14
```

- ➔ Ce code crée une classe nommée Produit, qui fonctionne comme une entité JPA. Cette classe est équipée de quatre attributs : idProduit, nomProduit, prixProduit, et dateProduit. Elle inclut également les méthodes getters et setters nécessaires pour manipuler ces attributs.
- ➔ Un constructeur qui accepte les valeurs initiales pour ces attributs est aussi défini, de même qu'une méthode toString() qui permet représenter l'objet Produit sous forme de texte.

```

Produit.java X
produits > src > main > java > com > gt > produits > entities > {} Produit.java > Language Support for Java(TM) by Red I
1 package com.gt.produits.entities;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 import java.util.Date;
9 @Entity
10 public class Produit {
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private Long idProduit;
14     private String nomProduit;
15     private Double prixProduit;
16     private Date dateProduits ;
17
18
19
20     public Produit() {
21         super();
22     }
23
24     public Produit(String nomProduit, Double prixProduit, Date dateProduits) {
25         super();
26         this.nomProduit = nomProduit;
27         this.prixProduit = prixProduit;
28         this.dateProduits = dateProduits;
29     }
30
31     public Long getIdProduit() {
32         return idProduit;
33     }
34
35     public String getNomProduit() {
36         return nomProduit;
37     }

```

➔ Cette déclaration concerne une interface de repository appelée "ProduitRepository" pour Spring Data JPA. Elle hérite de l'interface "JpaRepository", spécifiant "Produit" comme type d'entité et "Long" comme type d'identifiant. Cette interface offre des méthodes CRUD pour manipuler l'entité "Produit".

```

{ } ProduitsRepository.java X
produits > src > main > java > com > gt > produits > repos > { } ProduitsRepository.java > ...
1  package com.gt.produits.repos;
2
3  import com.gt.produits.entities.Produit;
4  import org.springframework.data.jpa.repository.JpaRepository;
5
6  public interface ProduitsRepository extends JpaRepository<Produit,Long> {
7
8  }
9

```

➔ Cette interface, appelée "ProduitService", décrit les méthodes nécessaires pour interagir avec des objets de type "Produit". Elle comprend des opérations telles que la création, la mise à jour et la suppression d'un produit. En outre, elle permet de récupérer tous les produits et d'obtenir un produit spécifique à partir de son identifiant.

```

{ } ProduitService.java X
produits > src > main > java > com > gt > produits > service > { } ProduitService.java > ...
1  package com.gt.produits.service;
2
3  import java.util.List;
4  import com.gt.produits.entities.Produit;
5  public interface ProduitService {
6      Produit saveProduit(Produit p);
7      Produit updateProduit(Produit p);
8      void deleteProduit(Produit p);
9      void deleteProduitById(Long id);
10     Produit getProduit(Long id);
11
12     List<Produit> getAllProduits();
13 }

```

➔ Ce code présente une implémentation de l'interface ProduitService. Il utilise un objet de type ProduitRepository pour réaliser des opérations CRUD sur l'entité Produit. La classe porte l'annotation @Service pour indiquer au conteneur Spring qu'il s'agit d'un bean Spring qui doit être géré par le framework.

```
{ } ProduitServiceImpl.java X
produits > src > main > java > com > gt > produits > service > { } ProduitServiceImpl.java > Language Support for Java(TM) by Red Hat > { } com
1 package com.gt.produits.service;
2
3 import com.gt.produits.repos.ProduitsRepository;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.stereotype.Service;
6 import com.gt.produits.entities.Produit;
7
8 import java.util.List;
9
10 @Service
11 public class ProduitServiceImpl implements ProduitService {
12     @Autowired
13     ProduitsRepository produitRepository;
14
15     @Override
16     public Produit saveProduit(Produit p) {
17         return produitRepository.save(p);
18     }
19
20     @Override
21     public Produit updateProduit(Produit p) {
22         return produitRepository.save(p);
23     }
24
25     @Override
26     public void deleteProduit(Produit p) {
27         produitRepository.delete(p);
28     }
29     @Override
30     public void deleteProduitById(Long id) {
31         produitRepository.deleteById(id);
32     }
33     @Override
34     public Produit getProduit(Long id) {
35         return produitRepository.findById(id).get();
36     }
37     @Override
```

ed. Java: Ready Ln 1, Col 1 Spaces: 4

➔ la création, la recherche, la mise à jour, la suppression et la liste de tous les produits dans le système en utilisant le framework Spring Boot :

```
{ } ProduitsApplication.java X
produits > src > main > java > com > gt > produits > { } ProduitsApplication.java > ...
1  package com.gt.produits;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class ProduitsApplication {
8
9      Run main | Debug main | Run | Debug
10     public static void main(String[] args) {
11         SpringApplication.run(ProduitsApplication.class, args);
12     }
13 }
14
```

Test :

```
{ } ProduitsApplicationTests.java 1 X
produits > src > test > java > com > gt > produits > { } ProduitsApplicationTests.java > Language Support for Java(TM) by Red Hat > ProduitsApplicationTests.java
1  package com.gt.produits;
2
3  import java.util.Date;
4  import java.util.List;
5
6  import org.junit.jupiter.api.Test;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.boot.test.context.SpringBootTest;
9
10 import com.gt.produits.entities.Produit;
11 import com.gt.produits.repos.ProduitsRepository;
12
13 @SpringBootTest
14 class ProduitsApplicationTests {
15     @Autowired
16     private ProduitsRepository produitRepository;
17
18     @Test
19     public void testCreateProduit() {
20         Produit prod = new Produit(nomProduit:"PC Dell", prixProduit:2950.500, new Date());
21         produitRepository.save(prod);
22     }
23
24     @Test
25     public void testFindProduit() {
26         Produit p = produitRepository.findById(id:1L).get();
27         System.out.println(p);
28     }
29 }
```

Tous les tests sont finis avec succès :

```

    ✓ Test run at 5/5/2024, 10:49:18 AM
    ✓ testUpdateProduit
    ✓ Test run at 5/5/2024, 10:48:33 AM
    ✓ testFindProduit()
    ✓ Test run at 5/5/2024, 10:48:17 AM
    ✓ testCreateProduit
    ✓ Test run at 5/5/2024, 10:47:37 AM
    ✓ testListerTousProduits

Hibernate: select p1_0.id_produit,p1_0.date_produits,p1_0.nom_produit,p1_0.prix_produit from produit
p1_0 where p1_0.id_produit=?
Produit{idProduit=9, nomProduit='PC Dell', prixProduit=2950.5, dateProduits=2024-05-05 10:46:43.0}
Hibernate: select p1_0.id_produit,p1_0.date_produits,p1_0.nom_produit,p1_0.prix_produit from produit
p1_0 where p1_0.id_produit=?
Hibernate: select p1_0.id_produit,p1_0.date_produits,p1_0.nom_produit,p1_0.prix_produit from produit
p1_0 where p1_0.id_produit=?
Hibernate: update produit set date_produits=?,nom_produit=?,prix_produit=? where id_produit=?
Hibernate: select p1_0.id_produit,p1_0.date_produits,p1_0.nom_produit,p1_0.prix_produit from produit
p1_0
Produit{idProduit=8, nomProduit='PC Dell', prixProduit=1000.0, dateProduits=2024-05-05 10:45:56.0}
Produit{idProduit=9, nomProduit='PC Dell', prixProduit=2950.5, dateProduits=2024-05-05 10:46:43.0}
Produit{idProduit=10, nomProduit='PC Dell', prixProduit=2950.5, dateProduits=2024-05-05 10:47:46.0}
Produit{idProduit=11, nomProduit='PC Dell', prixProduit=2950.5, dateProduits=2024-05-05 10:48:26.0}
Produit{idProduit=12, nomProduit='PC Dell', prixProduit=2950.5, dateProduits=2024-05-05 10:49:27.0}
Hibernate: select p1_0.id_produit,p1_0.date_produits,p1_0.nom_produit,p1_0.prix_produit from produit
p1_0 where p1_0.id_produit=?
-----
BUILD SUCCESS
-----
```

Lors de l'exécution des tests, les requêtes Hibernate s'affichent dans la console

- La première requête est de type INSERT et sert à ajouter un nouveau produit dans la table "produit", en spécifiant le nom, le prix et la date du produit.
- Les deuxième et troisième requêtes, de type SELECT, permettent de récupérer le produit tout juste ajouté.
- La quatrième requête, également une requête SELECT, est utilisée pour obtenir la liste de tous les produits présents dans la table "produit".
- La dernière ligne de la console montre les détails du produit récupéré par la deuxième requête SELECT, y compris les attributs idProduit, nomProduit, prixProduit, et dateProduit.

- J'ai également confirmé via PhpMyAdmin que toutes les modifications ont été correctement appliquées.

PhpMyAdmin :

The screenshot shows the phpMyAdmin web interface. On the left is a sidebar with a tree view of databases and tables. The main area displays the 'produit' table from the 'spring_db' database. A green status bar at the top indicates 'Affichage des lignes 0 - 3 (total de 4, traitement en 0.0003 seconde(s))'. Below this, the SQL query 'SELECT * FROM `produit`' is shown. A toolbar with various actions like 'Parcourir', 'Structure', 'SQL', 'Rechercher', 'Insérer', 'Exporter', 'Importer', 'Privileges', 'Opérations', and 'Déclencheurs' is visible. The table data is presented in a grid with columns: id_produit, date_produits, nom_produit, and prix_produit. Two rows are visible, each with edit, copy, and delete icons. At the bottom, there are buttons for 'Imprimer', 'Copier dans le presse-papiers', 'Exporter', 'Afficher le graphique', and 'Créer une vue'.

	id_produit	date_produits	nom_produit	prix_produit
<input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer	8	2024-05-05 09:45:56.000000	PC Dell	1000
<input type="checkbox"/> Éditer <input type="checkbox"/> Copier <input type="checkbox"/> Supprimer	9	2024-05-05 09:46:43.000000	PC Dell	2950.5

Merci de votre Attention !