



Sagesse Devoir  
Ecole Polytechnique Thiès



Sagesse Devoir  
Ecole Polytechnique Thiès



République du Sénégal  
Un Peuple – Un But – Une Foi  
Ministère l'Enseignement Supérieur de la Recherche et  
de l'Innovation  
École Polytechnique de Thiès  
B.P.A 10 Thiès  
Tél: (221) 76 223 61 74 – Fax: (221) 33 951 14 67

## RAPPORT SUR SERVICE WEB RESTFUL

---

Présenté par :

Mohamed Massamba SENE

Professeur  
Dr. Samba Sidibé

Matière  
Développement Web 3

## Table des matières

I.	Définir les ressources .....	3
1.	Identification des ressources .....	3
2.	Concevoir les endpoints .....	3
II.	Conception des services web.....	6
1.	Développement des services web.....	6
2.	Documentation des services web.....	16
III.	Test des services web .....	23
IV.	Client Angular+bootstrap.....	29

## I. Définir les ressources

### 1. Identification des ressources

Nous avons décidé pour les clients mobiles et web de créer des interfaces pour la vente en ligne. L'objectif est de permettre à un client en créant un compte de parcourir l'ensemble des produits disponibles avec la possibilité de les regrouper par catégorie ou par marque. Le client aura ensuite la possibilité lorsqu'il trouve le produit désiré de voir tous les détails de ce produit ainsi que les magasins où il est encore en stock et de l'ajouter à son panier. Après cela il pourra alors passer une commande au niveau du magasin pour le produit spécifié. Les stocks seront ensuite mis à jour pour refléter la diminution de la quantité de ce produit disponible.

Le client pourra également consulter la liste de ses commandes, modifier ses informations et supprimer son compte. A noter que la suppression du compte se limite à son accès au client web ou mobile et ne sera pas reporté au niveau de la base de données pour des raisons de traçabilité.

Ainsi les ressources que nous jugeons nécessaires aux fonctionnement des applications sont :

- ArticleCommandeResource pour la gestion des articles commandes
- CategorieResource pour la gestion des catégories
- ClientResource pour la gestion des clients
- CommandeResource pour la gestion des commandes
- MarqueResource pour la gestion des marques
- ProduitResource pour la gestion des produits
- StockResource pour la gestion des stocks

### 2. Concevoir les endpoints

#### Endpoint pour ArticleCommandeResource

Pour cette ressource nous utilisons :

- L'URI « /article » avec comme action possible l'enregistrement des articles commandes appartenant à une commande passée par un client. Pour cela nous utiliserons donc la méthode PUT afin de pouvoir enregistrer et modifier en cas d'erreur les articles commandes.

#### Endpoint pour CategorieResource

Pour cette ressource nous utilisons :

- L'URI « /categorie » avec comme action possible de trouver l'ensemble des catégories disponibles. Pour cela nous utiliserons donc la méthode GET.
- L'URI « /categorie/{id} » avec comme action possible de trouver l'ensemble des produits disponibles appartenant à la catégorie spécifiée. Pour cela nous utiliserons également la méthode GET en passant l'id de la catégorie comme un PathParam.

### **Endpoint pour ClientResource**

Pour cette ressource nous utilisons :

- L'URI « /client » avec comme action possible l'enregistrement d'un nouveau client qui se serait inscrit à partir du client web ou mobile ou la modification de ses informations. Pour cela nous utiliserons donc la méthode PUT.
- L'URI « /client/{id} » avec comme action possible de trouver l'ensemble des commandes effectuées par le client spécifié. Pour cela nous utiliserons donc la méthode GET en passant l'id du client comme un PathParam.

### **Endpoint pour CommandeResource**

Pour cette ressource nous utilisons :

- L'URI « /commande » avec comme action possible l'enregistrement d'une commande passée par un client à partir du client web ou mobile ainsi que la modification en cas d'erreur. Pour cela nous utiliserons donc la méthode PUT.
- L'URI « /commande/{numero} » avec comme action possible de trouver l'ensemble des articles commande appartenant à la commande spécifiée. Pour cela nous utiliserons donc la méthode GET en passant le numéro du client comme un PathParam

### **Endpoint pour MarqueResource**

Pour cette ressource nous utilisons :

- L'URI « /marque » avec comme action possible de trouver l'ensemble des marques disponibles. Pour cela nous utiliserons donc la méthode GET.
- L'URI « /marque/{id} » avec comme action possible de trouver l'ensemble des produits disponibles appartenant à la marque spécifiée. Pour cela nous utiliserons également la méthode GET en passant l'id de la marque comme un PathParam.

### **Endpoint pour ProduitResource**

Pour cette ressource nous utilisons :

- L'URI « /produit » avec comme action possible de trouver l'ensemble des produits disponibles. Pour cela nous utiliserons donc la méthode GET

- L'URI « /produit/{id} » avec comme action possible de récupérer les informations d'un produit spécifique. Pour cela nous utiliserons également la méthode GET en passant l'id du produit comme un PathParam

### **Endpoint pour StockResource**

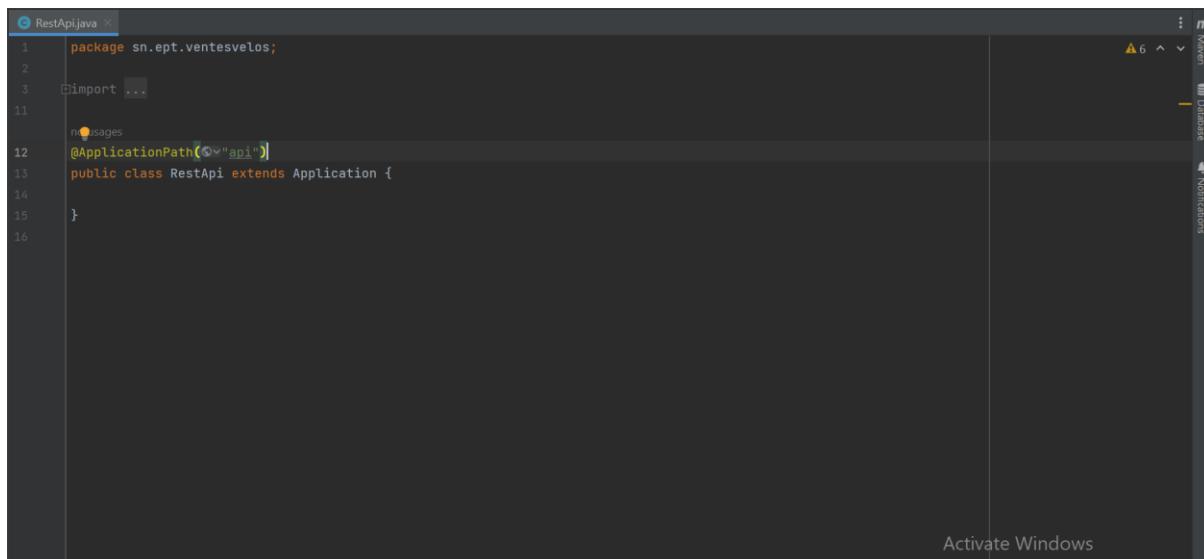
Pour cette ressource nous utilisons :

- L'URI « /stock » avec comme action possible de trouver la liste des stocks présents. Pour cela nous utiliserons donc la méthode GET.  
Nous avons également la possibilité d'enregister ou de modifier un stock lorsqu'il y a une modification de la quantité disponible après passage d'une commande. Pour cela nous utiliserons donc la méthode PUT.
- L'URI « /stock/{magasin\_id}/{produit\_id} » avec comme action possible de trouver les informations sur un stock spécifique. Pour cela nous utiliserons également la méthode GET en passant comme PathParams l'id du magasin et l'id du produit.  
Nous avons également la possibilité de supprimer un stock spécifique. Pour cela nous utiliserons donc la méthode DELETE en passant comme PathParams l'id du magasin et l'id du produit

## II. Conception des services web

### 1. Développement des services web

Pour développer les services web nous commençons donc par créer une classe **RestApi.java** dans laquelle nous utilisons l'annotation `@ApplicationPath("api")` pour spécifier le point d'entrée de l'API Rest. Toutes les ressources et sous-ressources définies dans votre application seront accessibles via des URL commençant par "/api".



```
RestApi.java
1 package sn.ept.ventesvelos;
2
3 import ...
4
5
6
7
8
9
10
11
12 @ApplicationPath("api")
13 public class RestApi extends Application {
14
15 }
16
```

Nous créons désormais nos ressources :

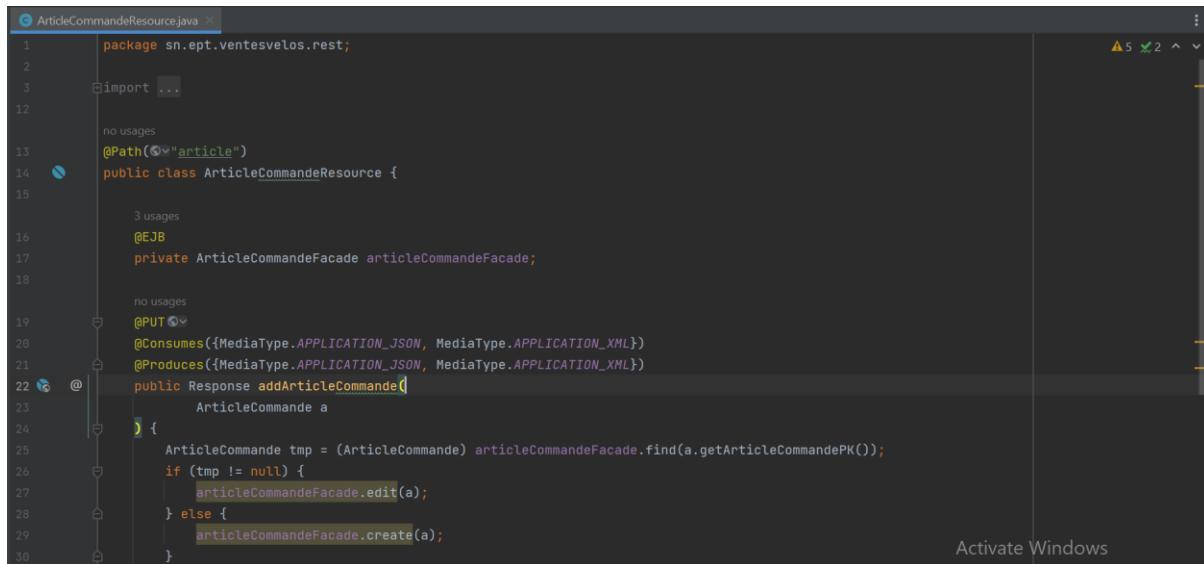
#### **ArticleCommandeResource.java**

Nous utilisons l'annotation `@Path` (« article ») utilisée en Jakarta RESTful Web Services (Jakarta RS) pour définir le chemin relatif d'une ressource ou d'une méthode d'une classe de ressource. On pourra donc accéder à cette en utilisant le chemin relatif « /api/article ». On injecte donc la façade correspondante en utilisant l'annotation `@EJB`.

Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec `@GET`, `@PUT`, `@DELETE` suivant le type de requête devant permettre d'y accéder. En utilisant `@Consumes` (`{MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}`) ou `@Produces` (`{MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}`) pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant le type de requête

Ici il s'agit donc de créer une méthode `addArticleCommande()` dans laquelle nous créons ou modifions l'article commande passée en paramètre à la méthode puis

nous l'envoyons comme réponse. On spécifie l'annotation @PUT pour spécifier que le type de requête permettant d'y accéder est PUT ainsi que @Produces et @Consumes pour que la communication soit faîte aux formats JSON et XML.



```
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5 no usages
6
7 @Path("article")
8 public class ArticleCommandeResource {
9
10    3 usages
11    @EJB
12    private ArticleCommandeFacade articleCommandeFacade;
13
14    no usages
15    @PUT
16    @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17    @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
18    public Response addArticleCommande(
19        ArticleCommande a
20    ) {
21        ArticleCommande tmp = (ArticleCommande) articleCommandeFacade.find(a.getArticleCommandePK());
22        if (tmp != null) {
23            articleCommandeFacade.edit(a);
24        } else {
25            articleCommandeFacade.create(a);
26        }
27    }
28}
29
30
```

## CategorieResource.java

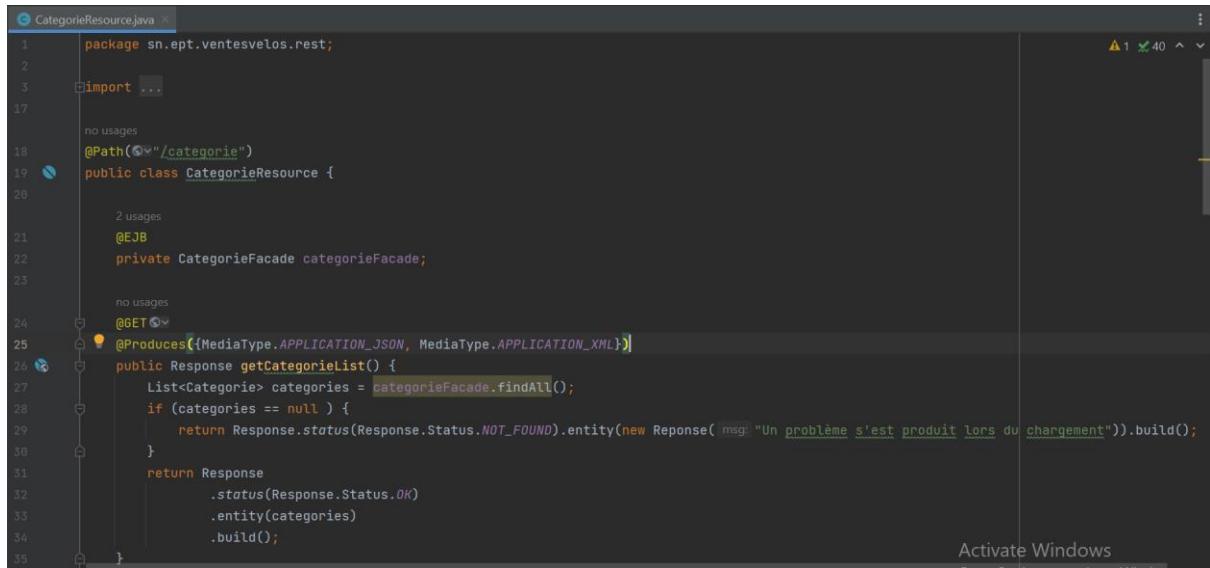
Nous utilisons l'annotation @Path (« categorie ») utilisée en Jakarta RESTful Web Services (Jakarta RS) pour définir le chemin relatif d'une ressource ou d'une méthode d'une classe de ressource. On pourra donc accéder à cette en utilisant le chemin relatif « /api/categorie ». On injecte donc la façade correspondante en utilisant l'annotation @EJB.

Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec @GET, @PUT, @DELETE suivant le type de requête devant permettre d'y accéder. En utilisant @Consumes ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) ou @Produces ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant le type de requête

Ici il s'agit donc de créer une méthode getCategorieList() dans laquelle nous récupérons l'ensemble des catégories présentes dans la base de données que nous envoyons comme réponse. On spécifie l'annotation @GET pour spécifier que le type de requête permettant d'y accéder est GET ainsi que l'annotation @Produces pour spécifier le format possible des réponses.

Nous créons également une méthode getProduitList() dans laquelle nous récupérons l'ensemble des produits appartenant à une catégorie spécifique dont l'id est passé en paramètre à la requête qui sera envoyé comme réponse. On spécifie l'annotation @GET et @Path en spécifiant le paramètre de chemin

d'accès {id}. On ajoute l'annotation @Produces pour spécifier le format possible des réponses.



```
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5 no usages
6
7 @Path("/categorie")
8 public class CategorieResource {
9
10     2 usages
11     @EJB
12     private CategorieFacade categorieFacade;
13
14     no usages
15     @GET
16     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17     public Response getCategorieList() {
18         List<Categorie> categories = categorieFacade.findAll();
19         if (categories == null) {
20             return Response.status(Response.Status.NOT_FOUND).entity(new Response( msg: "Un problème s'est produit lors du chargement")).build();
21         }
22         return Response
23             .status(Response.Status.OK)
24             .entity(categories)
25             .build();
26     }
27 }
```

## ClientResource.java

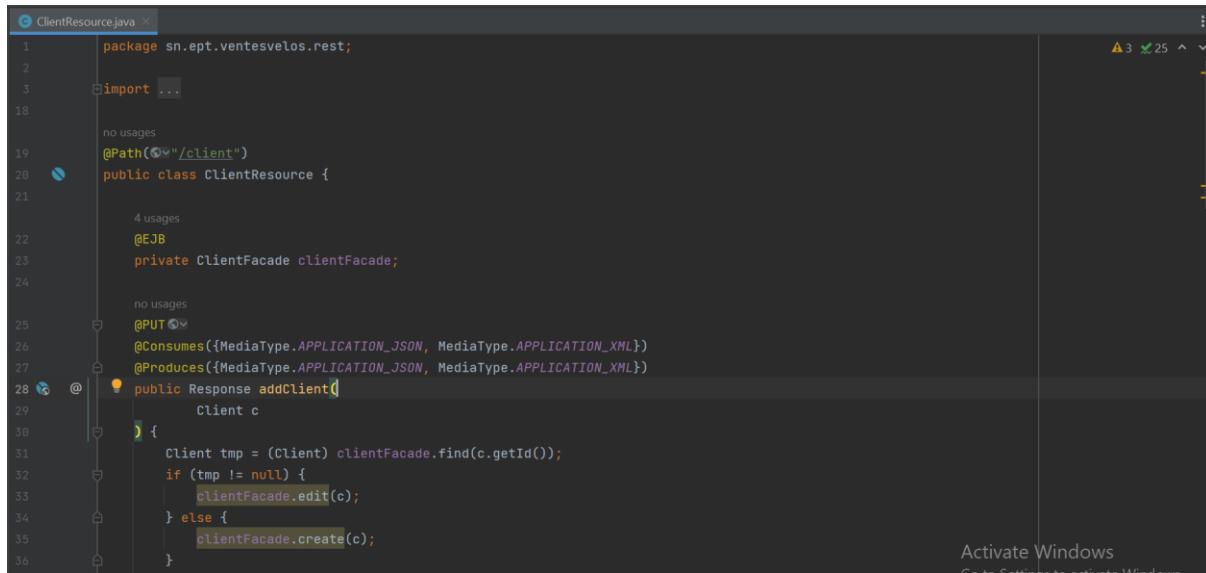
Nous utilisons l'annotation @Path (« client ») utilisée en Jakarta RESTful Web Services (Jakarta RS) pour définir le chemin relatif d'une ressource ou d'une méthode d'une classe de ressource. On pourra donc accéder à cette en utilisant le chemin relatif « /api/client ». On injecte donc la façade correspondante en utilisant l'annotation @EJB.

Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec @GET, @PUT, @DELETE suivant le type de requête devant permettre d'y accéder. En utilisant @Consumes ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) ou @Produces ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant le type de requête

Ici il s'agit donc de créer une méthode addClient() dans laquelle nous créons ou modifions le client passé en paramètre à la méthode puis nous l'envoyons comme réponse. On spécifie l'annotation @PUT pour spécifier que le type de requête permettant d'y accéder est PUT ainsi que les annotations @Produces et @Consumes pour que la communication soit faite aux formats JSON et XML.

Nous créons également une méthode getCommandeList() dans laquelle nous récupérons l'ensemble des commandes appartenant à un client spécifique dont l'id est passé en paramètre à la requête qui sera envoyé comme réponse. On spécifie l'annotation @GET et @Path en spécifiant le paramètre de chemin

d'accès {id}. On ajoute l'annotation @Produces pour spécifier le format possible des réponses.



```
ClientResource.java
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5 no usages
6
7 @Path("/client")
8 public class ClientResource {
9
10    4 usages
11    @EJB
12    private ClientFacade clientFacade;
13
14    no usages
15    @PUT
16    @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17    @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
18    public Response addClient(
19        Client c
20    ) {
21        Client tmp = (Client) clientFacade.find(c.getId());
22        if (tmp != null) {
23            clientFacade.edit(c);
24        } else {
25            clientFacade.create(c);
26        }
27    }
28
29
30
31
32
33
34
35
36 }
```

## CommandeResource.java

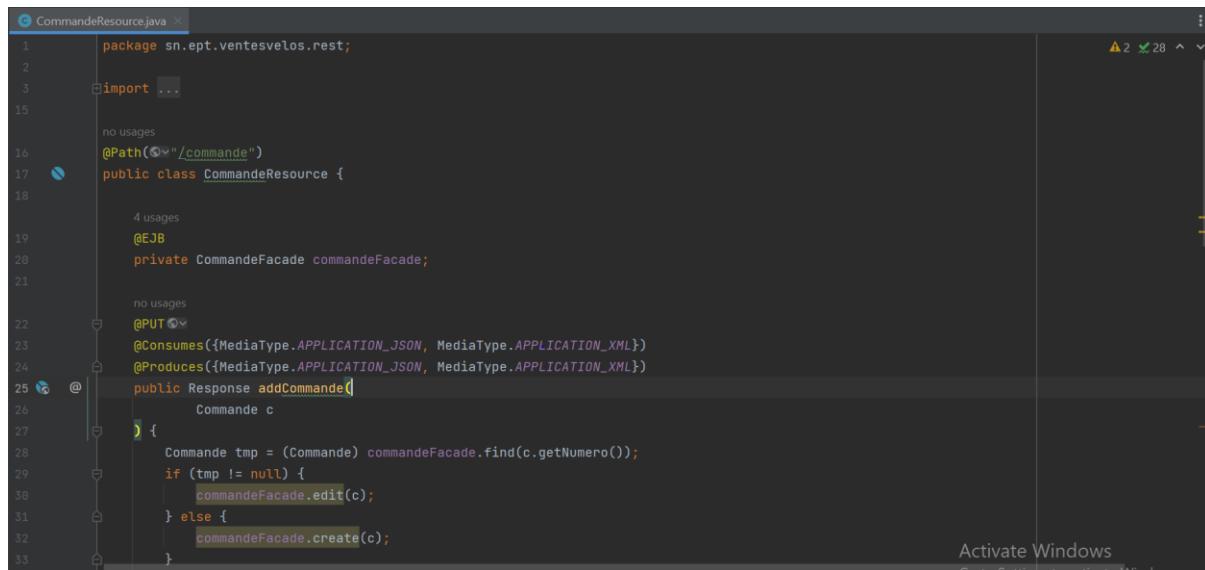
Nous utilisons l'annotation @Path (« commande ») utilisée en Jakarta RESTful Web Services (Jakarta RS) pour définir le chemin relatif d'une ressource ou d'une méthode d'une classe de ressource. On pourra donc accéder à cette en utilisant le chemin relatif « /api/commande ». On injecte donc la façade correspondante en utilisant l'annotation @EJB.

Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec @GET, @PUT, @DELETE suivant le type de requête devant permettre d'y accéder. En utilisant @Consumes ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) ou @Produces ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant le type de requête

Ici il s'agit donc de créer une méthode addCommande() dans laquelle nous créons ou modifions la commande passée en paramètre à la méthode puis nous l'envoyons comme réponse. On spécifie l'annotation @PUT pour spécifier que le type de requête permettant d'y accéder est PUT ainsi que les annotations @Produces et @Consumes pour que la communication soit faîte aux formats JSON et XML.

Nous créons également une méthode getArticleCommandeList() dans laquelle nous récupérons l'ensemble des articles commandes appartenant à une commande spécifique dont le numéro est passé en paramètre à la requête qui sera envoyé comme réponse. On spécifie l'annotation @GET et @Path en

spécifiant le paramètre de chemin d'accès {id}. On ajoute l'annotation @Produces pour spécifier le format possible des réponses.



```
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5 no usages
6
7 @Path("/commande")
8 public class CommandeResource {
9
10     4 usages
11     @EJB
12     private CommandeFacade commandeFacade;
13
14     no usages
15     @PUT
16     @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
18     public Response addCommande(
19         Commande c
20     ) {
21         Commande tmp = (Commande) commandeFacade.find(c.getNumero());
22         if (tmp != null) {
23             commandeFacade.edit(c);
24         } else {
25             commandeFacade.create(c);
26         }
27     }
28 }
```

## MarqueResource.java

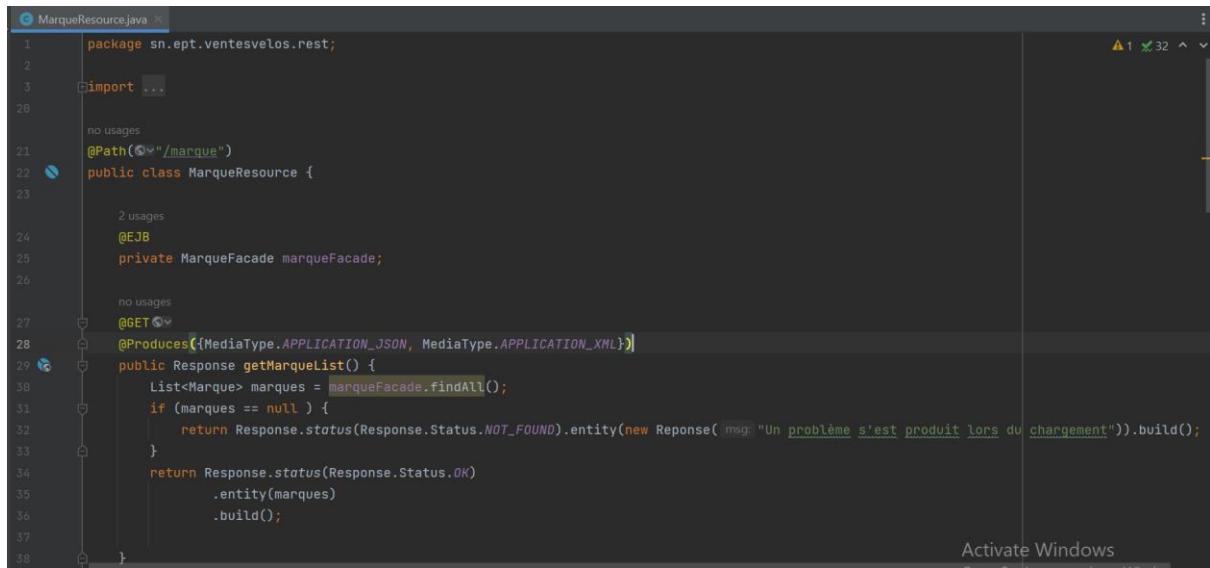
Nous utilisons l'annotation @Path (« marque ») utilisée en Jakarta RESTful Web Services (Jakarta RS) pour définir le chemin relatif d'une ressource ou d'une méthode d'une classe de ressource. On pourra donc accéder à cette en utilisant le chemin relatif « /api/marque ». On injecte donc la façade correspondante en utilisant l'annotation @EJB.

Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec @GET, @PUT, @DELETE suivant le type de requête devant permettre d'y accéder. En utilisant @Consumes ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) ou @Produces ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant le type de requête

Ici il s'agit donc de créer une méthode getMarqueList() dans laquelle nous récupérons l'ensemble des marques présentes dans la base de données que nous envoyons comme réponse. On spécifie l'annotation @GET pour spécifier que le type de requête permettant d'y accéder est GET ainsi que l'annotation @Produces pour spécifier le format possible des réponses.

Nous créons également une méthode getProduitList() dans laquelle nous récupérons l'ensemble des produits appartenant à une marque spécifique dont l'id est passé en paramètre à la requête qui sera envoyé comme réponse. On spécifie l'annotation @GET et @Path en spécifiant le paramètre de chemin

d'accès {id}. On ajoute l'annotation @Produces pour spécifier le format possible des réponses.



The screenshot shows a code editor window for a Java file named MarqueResource.java. The code defines a RESTful service for managing brands. It includes annotations for the path (@Path), EJB injection (@EJB), and HTTP methods (@GET). The @Produces annotation specifies that the service can return responses in JSON or XML format. The code also includes logic to handle a list of brands and return an appropriate response status based on the result.

```
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5 no usages
6
7 @Path("/marque")
8 public class MarqueResource {
9
10     2 usages
11     @EJB
12     private MarqueFacade marqueFacade;
13
14     no usages
15     @GET
16     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17     public Response getMarqueList() {
18         List<Marque> marques = marqueFacade.findAll();
19         if (marques == null) {
20             return Response.status(Response.Status.NOT_FOUND).entity(new Reponse( msg: "Un problème s'est produit lors du chargement")).build();
21         }
22         return Response.status(Response.Status.OK)
23             .entity(marques)
24             .build();
25     }
26
27 }
28
29
30
31
32
33
34
35
36
37
38 }
```

## ProduitResource.java

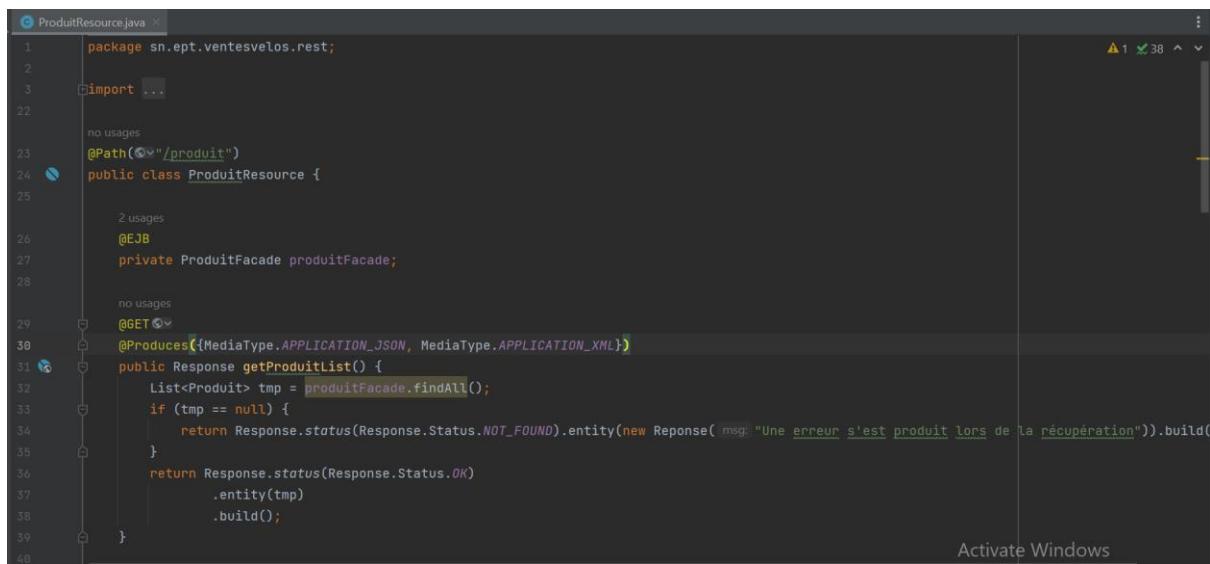
Nous utilisons l'annotation @Path (« produit ») utilisée en Jakarta RESTful Web Services (Jakarta RS) pour définir le chemin relatif d'une ressource ou d'une méthode d'une classe de ressource. On pourra donc accéder à cette en utilisant le chemin relatif « /api/produit ». On injecte donc la façade correspondante en utilisant l'annotation @EJB.

Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec @GET, @PUT, @DELETE suivant le type de requête devant permettre d'y accéder. En utilisant @Consumes ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) ou @Produces ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant le type de requête

Ici il s'agit donc de créer une méthode getProduitList() dans laquelle nous récupérons l'ensemble des produits présents dans la base de données que nous envoyons comme réponse. On spécifie l'annotation @GET pour spécifier que le type de requête permettant d'y accéder est GET ainsi que l'annotation @Produces pour spécifier le format possible des réponses.

Nous créons également une méthode getProduit() dans laquelle nous récupérons un produit spécifique dont l'id est passé en paramètre à la requête qui sera envoyé comme réponse. On spécifie l'annotation @GET et @Path en spécifiant le

paramètre de chemin d'accès {id}. On ajoute l'annotation @Produces pour spécifier le format possible des réponses.



```
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5 no usages
6 @Path("/produit")
7 public class ProduitResource {
8
9     2 usages
10    @EJB
11    private ProduitFacade produitFacade;
12
13    no usages
14    @GET
15    @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
16    public Response getProduitList() {
17        List<Produit> tmp = produitFacade.findAll();
18        if (tmp == null) {
19            return Response.status(Response.Status.NOT_FOUND).entity(new Response( msg: "Une erreur s'est produite lors de la récupération")).build();
20        }
21        return Response.status(Response.Status.OK)
22            .entity(tmp)
23            .build();
24    }
25 }
```

## StockResource.java

Nous utilisons l'annotation @Path (« stock ») utilisée en Jakarta RESTful Web Services (Jakarta RS) pour définir le chemin relatif d'une ressource ou d'une méthode d'une classe de ressource. On pourra donc accéder à cette en utilisant le chemin relatif « /api/stock ». On injecte donc la façade correspondante en utilisant l'annotation @EJB.

Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec @GET, @PUT, @DELETE suivant le type de requête devant permettre d'y accéder. En utilisant @Consumes ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) ou @Produces ({MediaType.APPLICATION\_JSON, MediaType.APPLICATION\_XML}) pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant le type de requête.

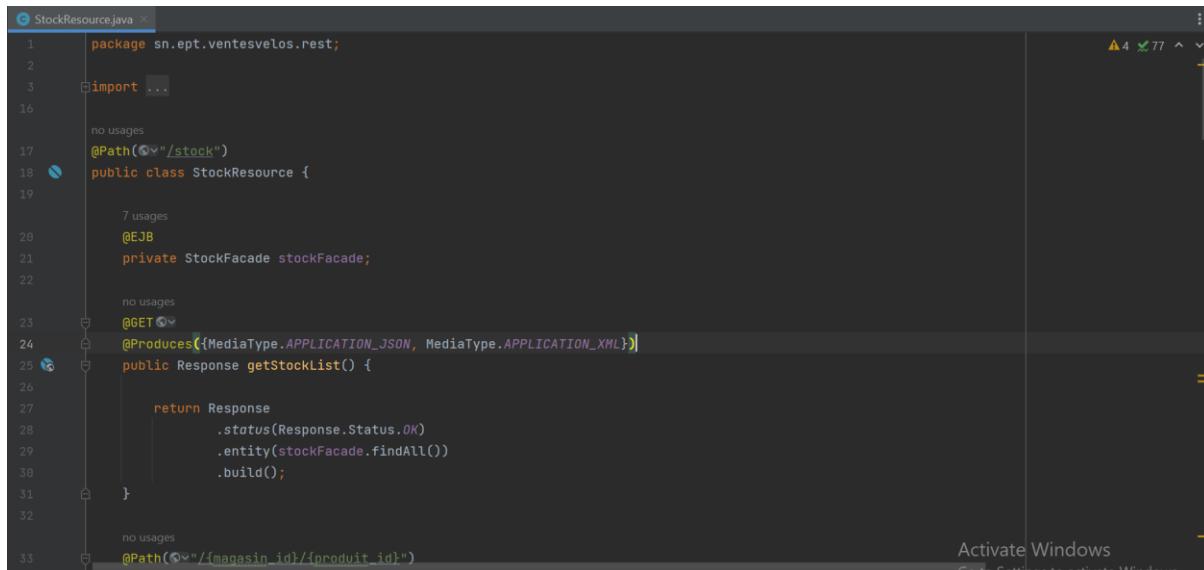
Ici il s'agit donc de créer une méthode getStockList() dans laquelle nous récupérons l'ensemble des stocks présents dans la base de données que nous envoyons comme réponse. On spécifie l'annotation @GET pour spécifier que le type de requête permettant d'y accéder est GET ainsi que l'annotation @Produces pour spécifier le format possible des réponses.

Nous créons une méthode addArticleCommande() dans laquelle nous créons ou modifions l'article commande passée en paramètre à la méthode puis nous l'envoyons comme réponse. On spécifie l'annotation @PUT pour spécifier que le type de requête permettant d'y accéder est PUT ainsi que les annotations

@Produces et @Consumes pour que la communication soit faîte aux formats JSON et XML.

Nous créons une méthode delete() dans laquelle nous supprimons un stock spécifique dont l'id du produit et du magasin sont passés en paramètre à la requête puis nous l'envoyons comme réponse. On spécifie l'annotation @GET pour spécifier que le type de requête permettant d'y accéder est GET et @Path en spécifiant les paramètres de chemin d'accès {produit\_id} et {magasin\_id}. On ajoute l'annotation @Produces pour spécifier le format possible des réponses

Nous créons également une méthode getStock() dans laquelle nous récupérons un stock spécifique dont l'id du produit et du magasin sont passés en paramètre à la requête puis nous l'envoyons comme réponse. On spécifie l'annotation @GET pour spécifier que le type de requête permettant d'y accéder est GET et @Path en spécifiant les paramètres de chemin d'accès {produit\_id} et {magasin\_id}. On ajoute l'annotation @Produces pour spécifier le format possible des réponses.

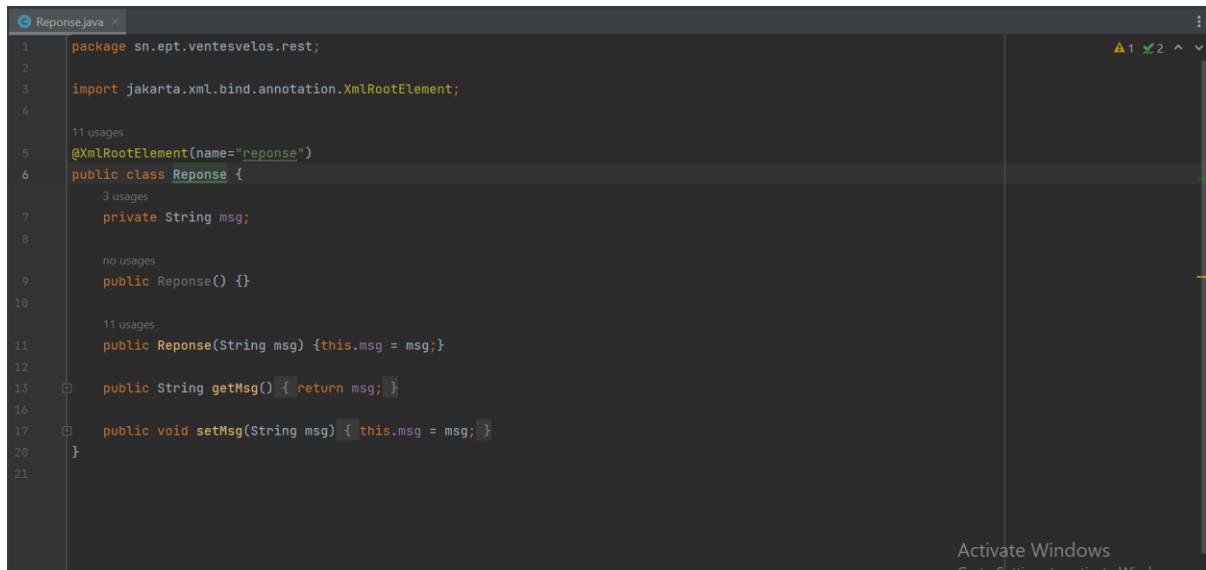


```
1 package sn.upt.ventesvelos.rest;
2
3 import ...
4
5 no usages
6
7 @Path("/stock")
8 public class StockResource {
9
10     7 usages
11     @EJB
12     private StockFacade stockFacade;
13
14     no usages
15     @GET
16     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17     public Response getStockList() {
18
19         return Response
20             .status(Response.Status.OK)
21             .entity(stockFacade.findAll())
22             .build();
23     }
24
25     no usages
26     @Path("/{magasin_id}/{produit_id}")
27 }
```

A cela nous ajoutons deux classes supplémentaires

### Reponse.java

Cette classe nous permet d'envoyer un message personnalisé. On ajoute également l'annotation @RootElement pour indiquer qu'elle est un élément racine lors de la sérialisation ou de la désérialisation vers XML, et vice versa.



```
Reponse.java
1 package sn.ept.ventesvelos.rest;
2
3 import jakarta.xml.bind.annotation.XmlRootElement;
4
5 @XmlRootElement(name="reponse")
6 public class Reponse {
7     private String msg;
8
9     public Reponse() {}
10
11    public Reponse(String msg) {this.msg = msg;}
12
13    public String getMsg() { return msg; }
14
15    public void setMsg(String msg) { this.msg = msg; }
16
17 }
18
19
20
21
```

Activate Windows  
Go to Settings to activate Windows

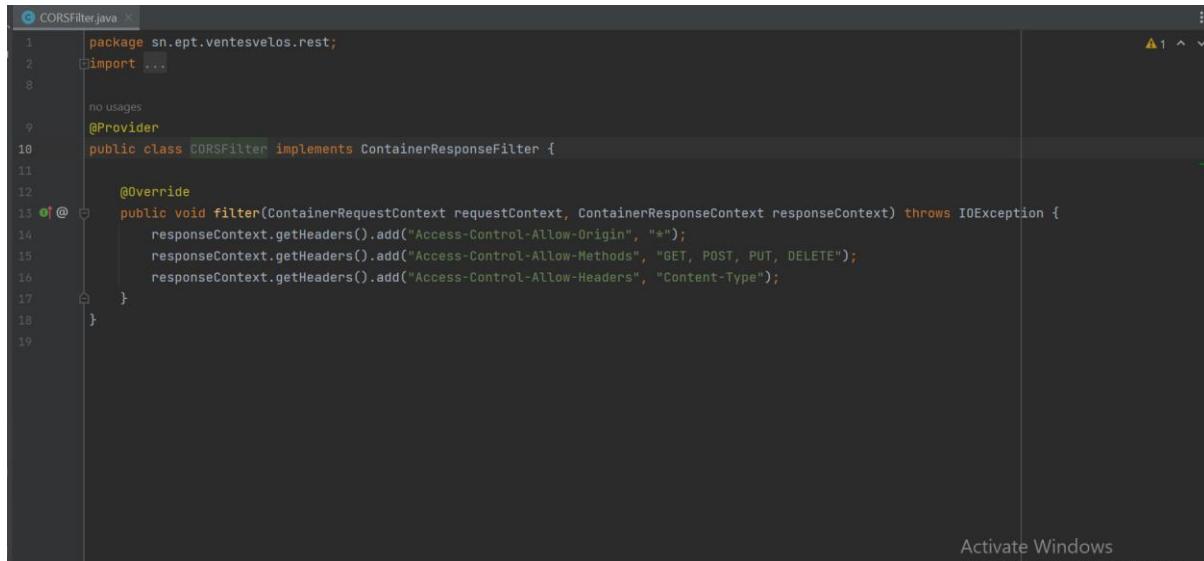
## CORSFilter.java

Etant donné que nous allons accéder à notre API depuis les clients web et mobile, en créant une classe CORSFilter qui implémente l'interface ContainerResponseFilter avec l'annotation @Provider, nous pouvons définir les en-têtes CORS nécessaires pour autoriser ou restreindre les requêtes cross-origin.

En définissant "Access-Control-Allow-Origin" sur "\*", nous autorisons les demandes de n'importe quelle origine.

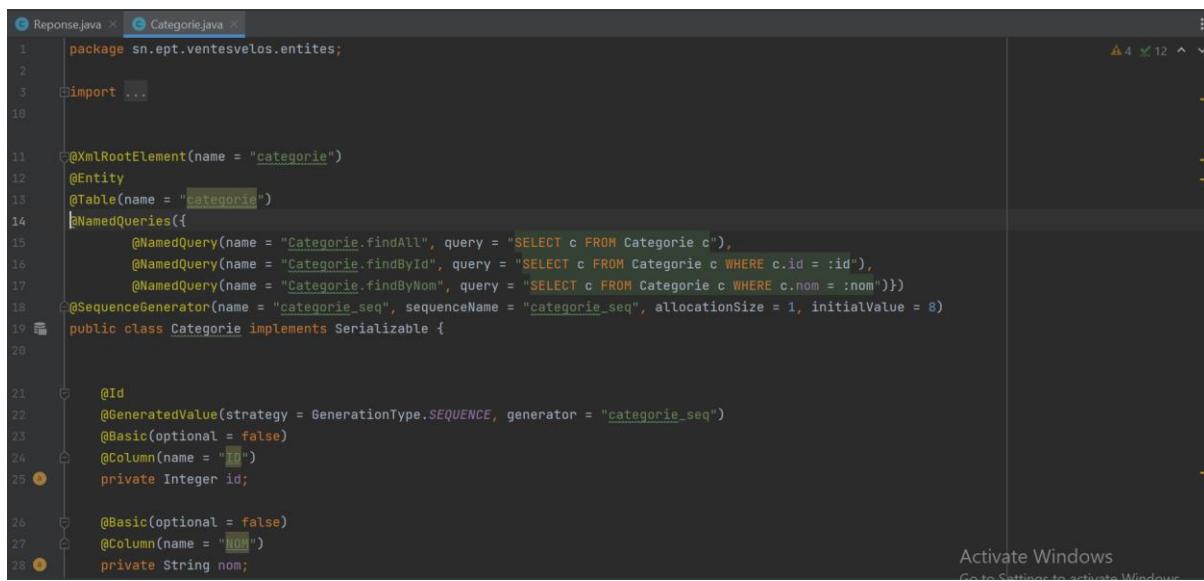
L'en-tête "Access-Control-Allow-Methods" spécifie les méthodes HTTP autorisées pour les requêtes cross-origin.

L'en-tête "Access-Control-Allow-Headers" spécifie les en-têtes autorisés dans la demande d'origine croisée.



```
1 package sn.ept.ventesvelos.rest;
2 import ...
3
4 no usages
5
6 @Provider
7 public class CORSFilter implements ContainerResponseFilter {
8
9     @Override
10    public void filter(ContainerRequestContext requestContext, ContainerResponseContext responseContext) throws IOException {
11        responseContext.getHeaders().add("Access-Control-Allow-Origin", "*");
12        responseContext.getHeaders().add("Access-Control-Allow-Methods", "GET, POST, PUT, DELETE");
13        responseContext.getHeaders().add("Access-Control-Allow-Headers", "Content-Type");
14    }
15
16
17
18 }
```

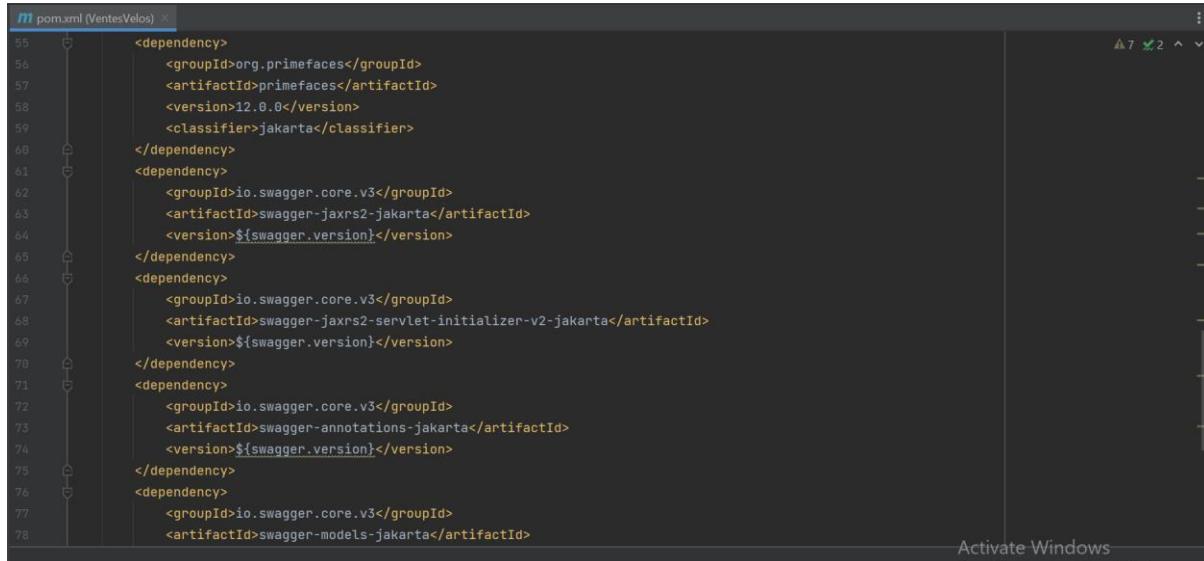
Pour les besoins de la communication au format XML on ajoute également l'annotation @XmlRootElement à nos entités. Comme on peut le voir dans cet exemple avec Catégorie.



```
1 package sn.ept.ventesvelos.entites;
2
3 import ...
4
5
6 @XmlRootElement(name = "categorie")
7 @Entity
8 @Table(name = "categorie")
9 @NamedQueries({
10     @NamedQuery(name = "Categorie.findAll", query = "SELECT c FROM Categorie c"),
11     @NamedQuery(name = "Categorie.findById", query = "SELECT c FROM Categorie c WHERE c.id = :id"),
12     @NamedQuery(name = "Categorie.findByNom", query = "SELECT c FROM Categorie c WHERE c.nom = :nom")})
13
14 @SequenceGenerator(name = "categorie_seq", sequenceName = "categorie_seq", allocationSize = 1, initialValue = 8)
15
16 public class Categorie implements Serializable {
17
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "categorie_seq")
21     @Basic(optional = false)
22     @Column(name = "ID")
23     private Integer id;
24
25     @Basic(optional = false)
26     @Column(name = "Nom")
27     private String nom;
```

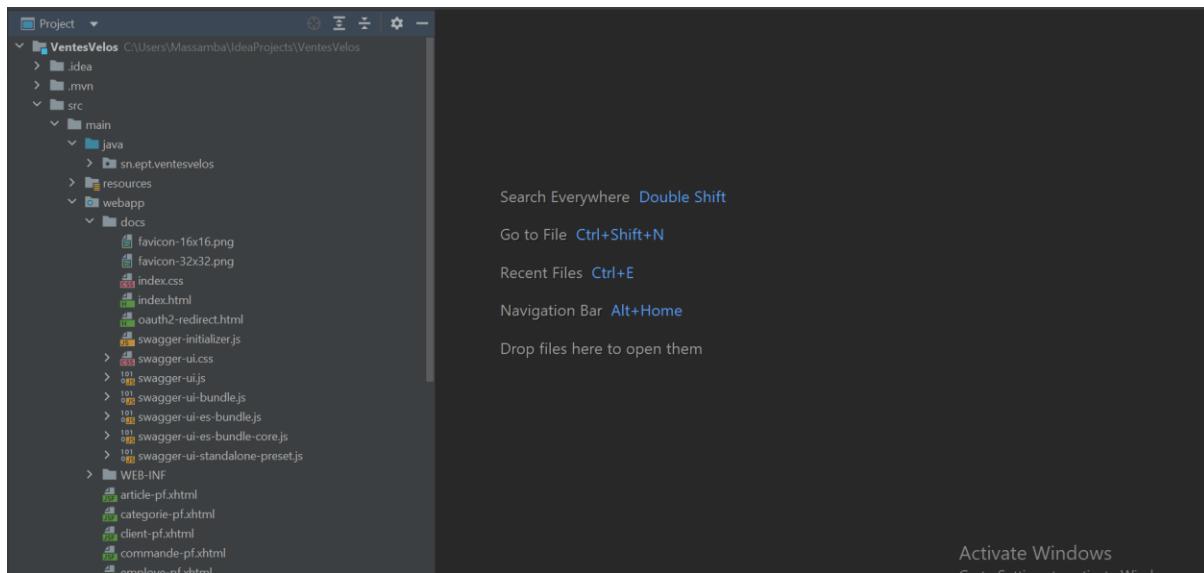
## 2. Documentation des services web

Pour documenter ces services nous allons utiliser swagger. Pour cela nous commençons d'abord par ajouter les dépendances nécessaires dans notre fichier pom.xml.



```
55 <dependency>
56     <groupId>org.primefaces</groupId>
57     <artifactId>primefaces</artifactId>
58     <version>12.0.0</version>
59     <classifier>jakarta</classifier>
60 </dependency>
61 <dependency>
62     <groupId>io.swagger.core.v3</groupId>
63     <artifactId>swagger-jaxrs2-jakarta</artifactId>
64     <version>${swagger.version}</version>
65 </dependency>
66 <dependency>
67     <groupId>io.swagger.core.v3</groupId>
68     <artifactId>swagger-jaxrs2-servlet-initializer-v2-jakarta</artifactId>
69     <version>${swagger.version}</version>
70 </dependency>
71 <dependency>
72     <groupId>io.swagger.core.v3</groupId>
73     <artifactId>swagger-annotations-jakarta</artifactId>
74     <version>${swagger.version}</version>
75 </dependency>
76 <dependency>
77     <groupId>io.swagger.core.v3</groupId>
78     <artifactId>swagger-models-jakarta</artifactId>
```

Nous téléchargeons ensuite swagger-ui au niveau de [github.com/swagger-api/swagger-ui](https://github.com/swagger-api/swagger-ui) et on copie le dossier dist dans webapp dans un dossier docs.



Nous ouvrons ensuite le fichier *swagger-initializer.js* dans lequel nous placer l'url de la documentation

```

1  window.onload = function() {
2      //<editor-fold desc="Changeable Configuration Block">
3
4      // the following lines will be replaced by docker/configurator, when it runs in a docker-container
5      window.ui = SwaggerUIBundle({
6          url: "../api/openapi.json",
7          dom_id: "#swagger-ui",
8          deepLinking: true,
9          presets: [
10              SwaggerUIBundle.presets.apis,
11              SwaggerUIStandalonePreset
12          ],
13          plugins: [
14              SwaggerUIBundle.plugins.DownloadUrl
15          ],
16          layout: "StandaloneLayout"
17      });
18
19  //</editor-fold>
20};
21

```

Au niveau du fichier **RestApi.java** nous allons ajouter la documentation en utilisant l'annotation `@OpenAPIDefinition` est utilisée pour définir les métadonnées de base pour la documentation d'une API REST à l'aide de Swagger. Nous utilisons les attributs :

- `info` permet de spécifier les informations générales sur l'API, telles que le titre, la description, la version, le contact et les informations de licence. En utilisant l'annotation `@Info` utilisée pour définir les informations détaillées sur l'API.
- `servers` permet de spécifier les serveurs ou les points de terminaison sur lesquels l'API est déployée. En utilisant l'annotation `@Server` qui définit un serveur avec l'URL de déploiement et une variable de serveur `urlDeploiement` avec une valeur par défaut. Cela permet de fournir une flexibilité dans le déploiement de l'API en permettant de modifier dynamiquement l'URL de déploiement via une variable.

```

13  @OpenAPIDefinition(
14      info = @Info(
15          title = "API POUR LA BOUTIQUE DE VENTES DE VELOS",
16          description = "Ceci est une documentation Swagger définissant les endpoints pour les différentes ressources nécessaire aux applications",
17          contact = @Contact(
18              name = "Mohamed Massamba Sene",
19              email = "test@example.com",
20              url = "https://www.google.sn"
21          ),
22          version = "1.0.0",
23          license = @License(name = "OpenSource")
24      ),
25      servers = {
26          @Server(
27              url = "{urlDeploiement}",
28              variables = {
29                  @ServerVariable(name="urlDeploiement", defaultValue = "http://localhost:8080/VentesVelos-1.0-SNAPSHOT/")
30              }
31          )
32      }
33  )
34  public class RestApi extends Application {
35
36  }
37

```

Au niveau de nos ressources, nous utilisons les annotations pour documenter nos services web.

- *@Operation* utilisée pour définir les informations sur l'opération HTTP exposée par l'API. Elle prend plusieurs paramètres tels que le résumé, la description et les réponses attendues pour cette opération avec les annotations *@ApiResponse* utilisée pour définir une réponse spécifique à une opération. Elle prend des paramètres tels que le code de réponse, la description et éventuellement un contenu spécifique associé à la réponse.
- *@Parameter* utilisée pour décrire un paramètre d'une opération. Elle permet de spécifier des informations telles que le nom du paramètre, la description, s'il est requis ou non, et d'autres propriétés.
- *@Path* utilisée pour spécifier le chemin d'accès relatif à la ressource dans l'URL de l'API. Elle peut être utilisée au niveau de la classe pour définir un chemin de base commun pour toutes les méthodes de la ressource, ou au niveau de chaque méthode pour spécifier un chemin spécifique pour cette méthode.
- *@PathParam* utilisée pour extraire la valeur d'un paramètre de chemin d'accès à partir de l'URL de la requête.
- *@Content* utilisée pour spécifier le contenu d'une réponse dans différents formats. Elle permet de définir des exemples de contenu pour les différentes réponses possibles.
- *@ExampleObject* utilisée pour définir un exemple d'objet dans la documentation Swagger. Elle permet d'illustrer la structure et les valeurs attendues pour un objet donné.

Comme nous pouvons le voir dans cet exemple :

```
ClientResource.java
no usages
25  @PUT
26  @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
27  @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
28  @Operation(
29      summary = "Inscription ou Modification des clients",
30      description = "Cet endpoint est utilisé pour enregistrer les clients dans la base de données après leur inscription ou lorsque des modifications sont effectuées."
31      responses = {
32          @ApiResponse(
33              responseCode = "200",
34              description = "Le client a été enregistré à la base de données"
35          ),
36          @ApiResponse(
37              responseCode = "500",
38              description = "Une erreur s'est produite lors de l'enregistrement du client"
39      }
40  )
41
42  @
43  public Response addClient(
44      @Parameter(
45          name = "Client",
46          description = "Le client que vous souhaitez enregistrer ou modifier",
47          required = true
48      )
49      Client c
50  )
```

```
ClientResource.java
51     if (tmp != null) {
52         clientFacade.edit(c);
53     } else {
54         clientFacade.create(c);
55     }
56     return Response
57         .status(Response.Status.OK)
58         .entity(c)
59         .build();
60 }
61
no usages
62 @Path("/{id}")
63 @GET
64 @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
65 @Operation(
66     summary = "Trouver les commandes",
67     description = "Cet endpoint est utilisé pour obtenir les commandes du client",
68     responses = {
69         @ApiResponse(
70             responseCode = "200",
71             description = "L'ensemble des commandes ont été retourné"
72         ),
73         @ApiResponse(
74             responseCode = "404",
75             description = "Le client correspondant à l'id indiqué n'a pas pu être trouvé"
76         )
77     }
78 )
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
```

Activate Windows  
Go to Settings to activate Windows

```
ClientResource.java
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
```

Activate Windows  
Go to Settings to activate Windows

```
ClientResource.java
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
```

Activate Windows  
Go to Settings to activate Windows

Nous pouvons donc accéder à la documentation en utilisant le chemin relatif « /docs/index.html » et nous voyons qu'en final notre API documentée apparaît comme dans l'aperçu suivant :

The screenshot shows two instances of the Swagger UI browser extension. The top instance displays the main API documentation page for 'API POUR LA BOUTIQUE DE VENTES DE VELOS'. It includes a 'Servers' dropdown set to '{urlDeploiement}', a computed URL of 'http://localhost:8080/VentesVelos-1.0-SNAPSHOT/', and a 'Server variables' section with 'urlDeploiement' set to 'http://localhost:8080/Ventes'. The bottom instance is a detailed view of the 'default' endpoint, listing various API operations:

- PUT** /api/article Ajout d'un article à une commande
- GET** /api/categorie Liste des catégories
- GET** /api/categorie/{id} Trouver les produits
- PUT** /api/client Inscription ou Modification des clients
- GET** /api/client/{id} Trouver les commandes
- PUT** /api/commande Ajout d'une commande
- GET** /api/commande/{numero} Trouver les articles d'une commande
- GET** /api/marque Liste des marques

The screenshot shows two instances of the Swagger UI interface, both displaying the API documentation for the `VentesVélos-1.0-SNAPSHOT` application.

**Top Window (API Overview):**

- Endpoints:**
  - `GET /api/marque` Liste des marques
  - `GET /api/marque/{id}` Trouver les produits
  - `GET /api/produit/{id}` Trouver un produit
  - `GET /api/produit` Liste des produits
  - `GET /api/stock` Liste des stocks
  - `PUT /api/stock` Ajout d'un stock
  - `GET /api/stock/{magasin_id}/{produit_id}` Trouver un stock
  - `DELETE /api/stock/{magasin_id}/{produit_id}` Suppression de stock
- Schemas:** A section titled "Schemas" is visible.

**Bottom Window (Article Commande Endpoint):**

- Endpoint:** `PUT /api/article` Ajout d'un article à une commande
- Description:** Cet endpoint est utilisé pour ajouter les articles commandés à la base de données.
- Parameters:** No parameters
- Request body (required):** `application/json`  
A large text area displays the JSON schema for the request body:

```
{ "articleCommandePK": { "numeroCommande": 0, "ligne": 0 }, "quantite": 0, "prixDepart": 0, "remise": 0, "produitID": { "id": 0, "nom": "string", "anneeModel": 0, "prixDepart": 0, "categorieID": { "id": 0, "nom": "string" } } }
```

localhost:8080/VentesVélos-1.0-SNAPSHOT/docs/index.html#/default/getProduitList

GET /api/categorie Liste des catégories

Cet endpoint est utilisé pour obtenir l'ensemble des catégories de produits disponibles

Parameters

No parameters

Responses

Code	Description	Links
default	default response	No links

Media type: application/json

Controls Accept header.

Activate Windows  
Go to Settings to activate Windows.

GET /api/categorie/{id} Trouver les produits

Cet endpoint est utilisé pour obtenir l'ensemble des produits appartenant à une catégorie

Parameters

Name Description

**id** \* required  
integer(\$int32) L'identifiant de la catégorie  
(path)

Responses

Code	Description	Links
200	L'ensemble des produits ont été retourné	Activate Windows Go to Settings to No links Windows.

### III. Test des services web

Nous créons donc un nouveau projet Java appelé ResourceTest dans lequel nous ajoutons les dépendances nécessaires au niveau du fichier pom.xml.

The screenshot shows the 'New Project' dialog in IntelliJ IDEA. The 'Name' field is set to 'ResourceTest'. The 'Location' field shows the default path '~\IdeaProjects\ResourceTest'. The 'Create Git repository' checkbox is unchecked. Under 'Language', 'Java' is selected. Under 'Build system', 'Maven' is selected. Under 'JDK', 'corretto-17 java version "17.0.2"' is chosen. The 'Add sample code' checkbox is checked. In the 'Advanced Settings' section, 'GroupId' is set to 'sn.ept' and 'ArtifactId' is set to 'ResourceTest'. At the bottom right, there are 'Create' and 'Cancel' buttons.

**pom.xml (ResourceTest)**

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>${junit.version}</version>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit.version}</version>
</dependency>
<!-- RestAssured Dependencies -->
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>5.3.0</version>
</dependency>
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>json-schema-validator</artifactId>
    <version>5.3.0</version>
</dependency>
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>xml-path</artifactId>
```

```

m pom.xml (ResourceTest) x
40     <dependency>
41         <groupId>io.rest-assured</groupId>
42         <artifactId>xml-path</artifactId>
43         <version>5.3.0</version>
44     </dependency>
45     <!-- Jakarta RS -->
46     <dependency>
47         <groupId>jakarta.platform</groupId>
48         <artifactId>jakarta.jakartaee-api</artifactId>
49         <version>9.1.0</version>
50         <scope>provided</scope>
51     </dependency>
52     <!-- Hamcrest -->
53     <dependency>
54         <groupId>org.hamcrest</groupId>
55         <artifactId>hamcrest-all</artifactId>
56         <version>1.3</version>
57     </dependency>
58   </dependencies>
59
60   <build>
61     <plugins>
62         <!-- Run during build -->
63         <plugin>
64             <groupId>org.apache.maven.plugins</groupId>
65             <artifactId>maven-surefire-plugin</artifactId>
66             <version>3.0.0-M5</version>
67             <configuration>
68                 <includes>
69                     <include>*Test.java</include>
70                 </includes>
71             </configuration>
72         </plugin>
73     </plugins>
74   </build>
75
76
77 </project>

```

Nous créons ensuite des classes pour effectuer les tests unitaires sur chaque fonctionnalité

### ArticleTest.java

Dans cette classe nous allons tester la ressource ArticleCommandeResource en effectuant des tests sur l'ensemble des actions possibles définies au niveau de l'endpoint.

Pour cela on utilise l'annotation @BeforeAll pour créer une méthode setup() dans laquelle nous renseignons l'URL permettant d'accéder à l'endpoint et nous utilisons l'annotation @Test sur l'ensemble des méthodes permettant de tester les actions.

```

1 package sn.ept;
2
3 import ...
13
no usages
14 public class ArticleTest {
15
    no usages
16     @BeforeAll
17     public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVélos-1.0-SNAPSHOT/api"; }
18
    no usages
19     @Test
20     public void testAddArticleJSON() {
21         String requestBody = "{\"articleCommandePK\": {\"numeroCommande\": 1, \"ligne\": 6}, \"quantite\":1, \"remise\": 0, "
22     }

```

Run: ArticleTest

Tests passed: 2 of 2 tests – 4 sec 693 ms

ArticleTest (sn.ept) 4 sec 693 ms "C:\Program Files\Amazon Corretto\jdk17.0.2\_8\bin\java.exe" ...

testAddArticleJSON() 4 sec 328 ms

testAddArticleXML() 365 ms

Process finished with exit code 0

## CategorieTest.java

Dans cette classe nous allons tester la ressource CategorieResource en effectuant des tests sur l'ensemble des actions possibles définies au niveau de l'endpoint.

Pour cela on utilise l'annotation `@BeforeAll` pour créer une méthode `setup()` dans laquelle nous renseignons l'URL permettant d'accéder à l'endpoint et nous utilisons l'annotation `@Test` sur l'ensemble des méthodes permettant de tester les actions.

```

10
no usages
11 public class CategorieTest {
12
    no usages
13     @BeforeAll
14     public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVélos-1.0-SNAPSHOT/api"; }
15
    no usages
16     @Test
17     public void testGetCategorieListJSON() {
18         given()
19             .RequestSpecification()
20             .header("Accept", MediaType.APPLICATION_JSON)
21             .when()
22             .get("/categorie") Response

```

Run: CategorieTest

Tests passed: 3 of 3 tests – 3 sec 136 ms

CategorieTest (sn.ept) 3 sec 136 ms "C:\Program Files\Amazon Corretto\jdk17.0.2\_8\bin\java.exe" ...

testGetCategorieListJSON() 2 sec 837 ms

testGetProductListJSON() 258 ms

testGetProductListNotFound() 41 ms

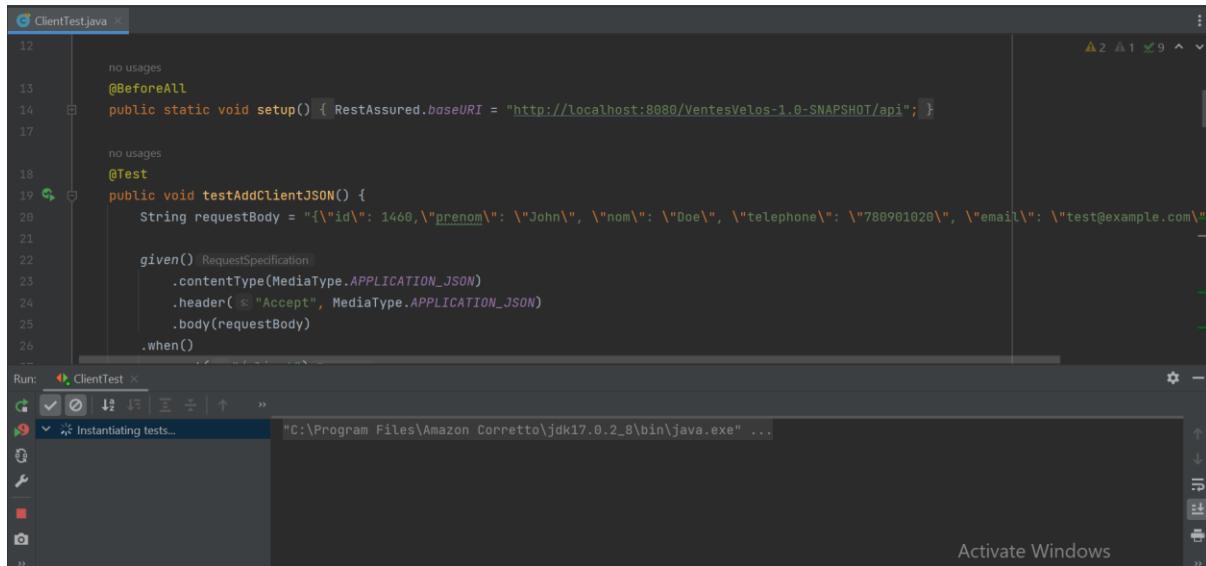
Process finished with exit code 0

## ClientTest.java

Dans cette classe nous allons tester la ressource ClientResource en effectuant des tests sur l'ensemble des actions possibles définies au niveau de l'endpoint.

Pour cela on utilise l'annotation `@BeforeAll` pour créer une méthode `setup()` dans laquelle nous renseignons l'URL permettant d'accéder à l'endpoint et nous

utilisons l'annotation `@Test` sur l'ensemble des méthodes permettant de tester les actions.

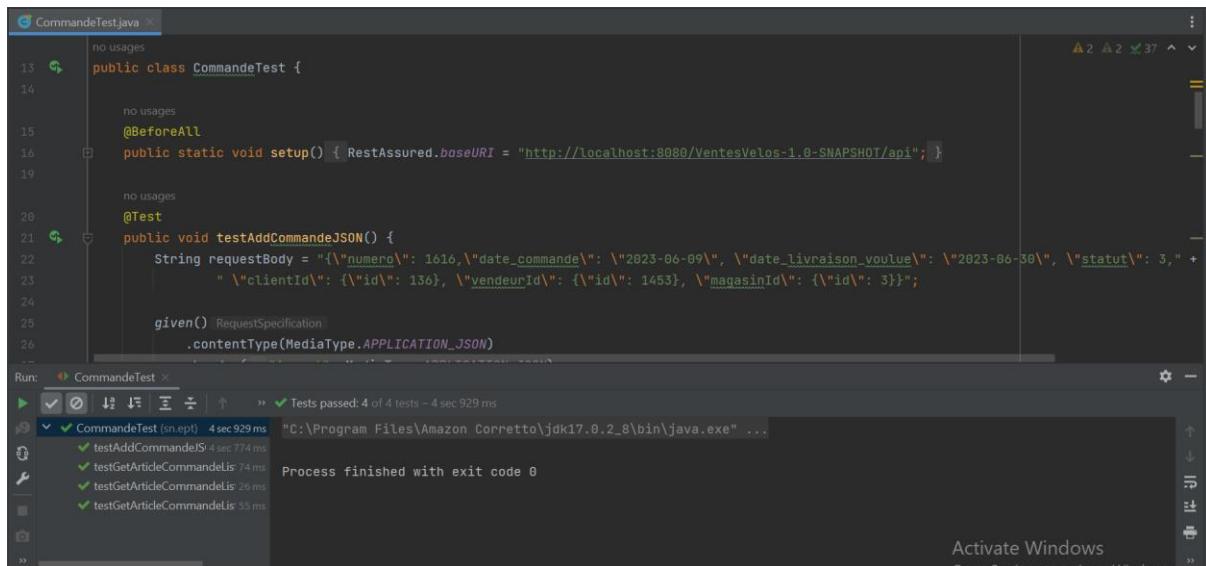


```
ClientTest.java
12
13     no usages
14     @BeforeAll
15     public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVélos-1.0-SNAPSHOT/api"; }
16
17     no usages
18     @Test
19     public void testAddClientJSON() {
20         String requestBody = "{\"id\": 1460, \"prenom\": \"John\", \"nom\": \"Doe\", \"telephone\": \"780901020\", \"email\": \"test@example.com\"}";
21
22         given()
23             .contentType(MediaType.APPLICATION_JSON)
24             .header("Accept", MediaType.APPLICATION_JSON)
25             .body(requestBody)
26         .when()
27
Run: ClientTest
```

## CommandeTest.java

Dans cette classe nous allons tester la ressource CommandeResource en effectuant des tests sur l'ensemble des actions possibles définies au niveau de l'endpoint.

Pour cela on utilise l'annotation `@BeforeAll` pour créer une méthode `setup()` dans laquelle nous renseignons l'URL permettant d'accéder à l'endpoint et nous utilisons l'annotation `@Test` sur l'ensemble des méthodes permettant de tester les actions.



```
CommandeTest.java
13
14     no usages
15     public class CommandeTest {
16
16     no usages
17     @BeforeAll
18     public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVélos-1.0-SNAPSHOT/api"; }
19
20     no usages
21     @Test
22     public void testAddCommandeJSON() {
23         String requestBody = "{\"numero\": 1616, \"date_commande\": \"2023-06-09\", \"date_livraison_voulu\": \"2023-06-30\", \"statut\": 3, \"clientId\": {\"id\": 136}, \"vendeurId\": {\"id\": 1453}, \"magasinId\": {\"id\": 3}}";
24
25         given()
26             .contentType(MediaType.APPLICATION_JSON)
```

## MarqueTest.java

Dans cette classe nous allons tester la ressource MarqueResource en effectuant des tests sur l'ensemble des actions possibles définies au niveau de l'endpoint.

Pour cela on utilise l'annotation `@BeforeAll` pour créer une méthode `setup()` dans laquelle nous renseignons l'URL permettant d'accéder à l'endpoint et nous utilisons l'annotation `@Test` sur l'ensemble des méthodes permettant de tester les actions.

The screenshot shows a Java IDE interface with two tabs: "MarqueTest.java" and "Run".

**MarqueTest.java:**

```
no usages
@BeforeAll
public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVélos-1.0-SNAPSHOT/api"; }

no usages
@Test
public void testGetMarqueListJSON() {
    given()
        .header(accept, MediaType.APPLICATION_JSON)
    .when()
        .get("/marque")
    .then()
        .statusCode(200)
        .contentType(MediaType.APPLICATION_JSON)
}
```

**Run:**

- MarqueTest
- Tests passed: 3 of 3 tests - 2 sec 331 ms
- MarqueTest (sn.ept) 2 sec 331 ms
  - testGetProduitListSOI 2 sec 222 ms
  - testGetMarqueListJSON() 68 ms
  - testGetProduitListNotFound 41 ms

Process finished with exit code 0

## ProduitTest.java

Dans cette classe nous allons tester la ressource ProduitResource en effectuant des tests sur l'ensemble des actions possibles définies au niveau de l'endpoint.

Pour cela on utilise l'annotation `@BeforeAll` pour créer une méthode `setup()` dans laquelle nous renseignons l'URL permettant d'accéder à l'endpoint et nous utilisons l'annotation `@Test` sur l'ensemble des méthodes permettant de tester les actions.

The screenshot shows the IntelliJ IDEA interface with the following details:

- ProductTest.java** is the active file, containing a single test method `testGetProduitListJSON()`.
- The code editor highlights the `setup()` method with a red underline, indicating a potential issue.
- Run** tab: Shows the test run configuration for `ProductTest`. The output indicates "Tests passed: 3 of 3 tests - 2 sec 642 ms".
- Toolbars and Status Bar:** Includes standard IntelliJ icons for file operations, a search bar, and a status bar at the bottom right that says "Activate Windows".

## StockTest.java

Dans cette classe nous allons tester la ressource StockResource en effectuant des tests sur l'ensemble des actions possibles définies au niveau de l'endpoint.

Pour cela on utilise l'annotation `@BeforeAll` pour créer une méthode `setup()` dans laquelle nous renseignons l'URL permettant d'accéder à l'endpoint et nous utilisons l'annotation `@Test` sur l'ensemble des méthodes permettant de tester les actions.

The screenshot shows an IDE interface with two main panes. The top pane displays the code for `StockTest.java`. The code includes a `setup()` method setting the base URI to `"http://localhost:8080/VentesVélos-1.0-SNAPSHOT/api"`, and a `@Test` annotated method `testGetStockListJSON()` using RestAssured to make a GET request to `"/stock"`. The bottom pane shows the test results in a 'Run' window titled 'StockTest [snapt]'. It lists 8 tests passed, each with a green checkmark and its execution time: `testAddStockJSON` (3 sec 693 ms), `testGetStockNotFound()` (59 ms), `testGetStockJSON()` (68 ms), `testDeleteStockXML()` (56 ms), `testDeleteStockNotFound()` (23 ms), `testDeleteStockNotFoundX()` (25 ms), `testDeleteStockJSON()` (27 ms), and `testGetStockListJSON()` (1 sec 177 ms). The status bar at the bottom right indicates "Process finished with exit code 0".

## IV. Client Angular+bootstrap

Nous créons un nouveau projet Angular en activant le routage

```
npm install
C:\Users\Massamba>cd OneDrive/Desktop
C:\Users\Massamba\OneDrive\Desktop>ng new BikeShop
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? SCSS [ https://sass-lang.com/documentation/syntax#scss ]
]
CREATE BikeShop/angular.json (2885 bytes)
CREATE BikeShop/package.json (1040 bytes)
CREATE BikeShop/README.md (1962 bytes)
CREATE BikeShop/tsconfig.json (901 bytes)
CREATE BikeShop/.editorconfig (274 bytes)
CREATE BikeShop/.gitignore (548 bytes)
CREATE BikeShop/tsconfig.app.json (263 bytes)
CREATE BikeShop/tsconfig.spec.json (273 bytes)
CREATE BikeShop/.vscode/extensions.json (130 bytes)
CREATE BikeShop/.vscode/launch.json (470 bytes)
CREATE BikeShop/.vscode/tasks.json (938 bytes)
CREATE BikeShop/src/main.ts (214 bytes)
CREATE BikeShop/src/favicon.ico (1642 bytes)
CREATE BikeShop/src/index.html (294 bytes)
CREATE BikeShop/src/styles.scss (80 bytes)
CREATE BikeShop/src/app/app-routing.module.ts (245 bytes)
CREATE BikeShop/src/app/app.module.ts (393 bytes)
CREATE BikeShop/src/app/app.component.html (23115 bytes)
CREATE BikeShop/src/app/app.component.spec.ts (997 bytes)
CREATE BikeShop/src/app/app.component.ts (213 bytes)
CREATE BikeShop/src/app/app.component.scss (0 bytes)
CREATE BikeShop/src/assets/.gitkeep (0 bytes)
\ Installing packages (npm)...
```

Pour utiliser bootstrap nous installons les modules suivants *bootstrap bootstrap-icons jquery popper.js* pour pouvoir utiliser le CSS et le JS fournit et on ajoute les chemins dans le fichier *angular.json*

```
angular.json M
...
  "main": "src/main.ts",
  "polyfills": [
    "zone.js"
  ],
  "tsConfig": "tsconfig.app.json",
  "inlineStyleLanguage": "scss",
  "assets": [
    "src/favicon.ico",
    "src/assets"
  ],
  "styles": [
    "src/styles.scss",
    "node_modules/bootstrap/dist/css/bootstrap.min.css",
    "node_modules/bootstrap-icons/font/bootstrap-icons.css"
  ],
  "scripts": [
    "node_modules/jquery/dist/jquery.min.js",
    "node_modules/popper.js/dist/umd/popper.min.js",
    "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
  ]
}

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

C:\Users\Massamba\OneDrive\Documents\JDK\Projet\BikeShop>npm install bootstrap bootstrap-icons jquery popper.js
up to date, audited 970 packages in 6s
110 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities
C:\Users\Massamba\OneDrive\Documents\JDK\Projet\BikeShop>
```

Nous créons les modèles pour nos entités définies au niveau du projet JakartaEE. Voici un aperçu de quelques-uns :

```
cart.models.ts
```

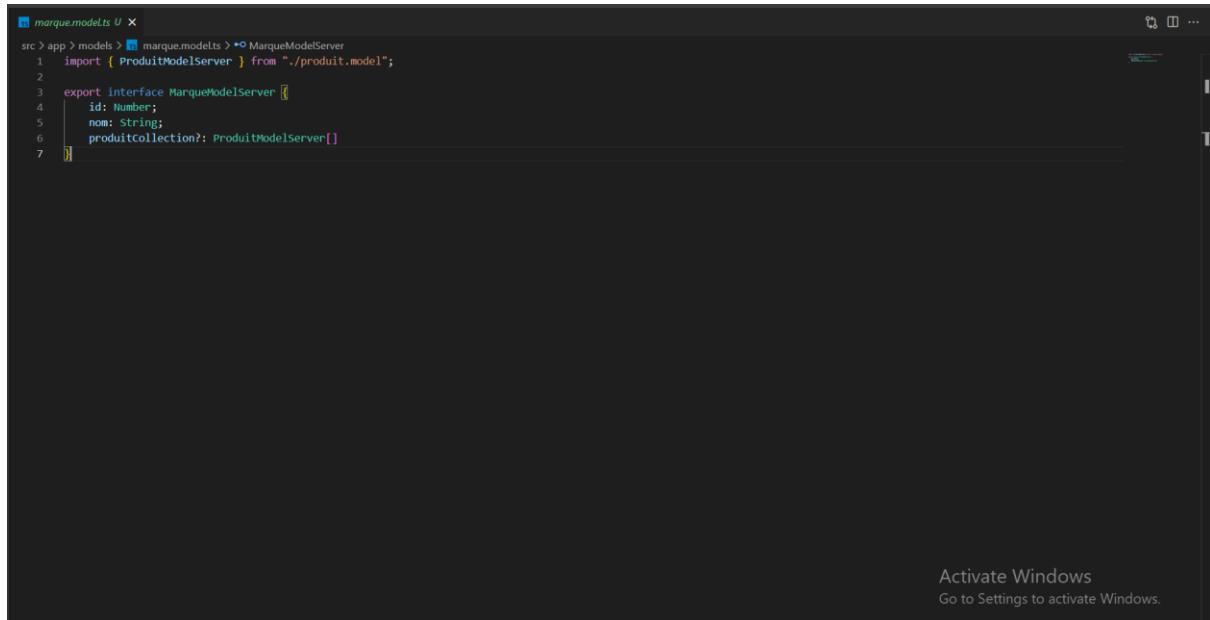
```
src > app > models > cart.models > ...
1 import { ProduitModelServer } from "./produit.model";
2 import { StockModelServer } from "./stock.model";
3
4 export interface CartModelServer {
5   total: number;
6   itemsInCart: number;
7   data: CartProduitServer[]
8 }
9
10 export interface CartProduitServer {
11   product: ProduitModelServer;
12   numInCart: number;
13   stock?: StockModelServer
14 }
```

Activate Windows  
Go to Settings to activate Windows.

```
commande.models.ts
```

```
src > app > models > commande.models > CommandeModelServer > dateLivraisonVoulu
1 import { ArticleModelServer } from "./article.model";
2 import { EmployeModelServer } from "./employe.model";
3 import { MagasinModelServer } from "./magasin.model";
4
5 export interface CommandeModelServer {
6   numero?: Number;
7   statut: Number;
8   dateCommande: string;
9   dateLivraisonVoulu: string;
10  dateLivraison?: Date;
11  clientId: ClientDetailsModelServer;
12  vendeurId: EmployeModelServer;
13  magasinId?: MagasinModelServer;
14  articleCommandeCollection: ArticleModelServer[];
15 }
16
17 export interface ClientDetailsModelServer {
18   id: number;
19   prenom: String;
20   nom: String;
21   telephone: String;
22   email: String;
23   adresse: {
24     adresse: String;
25     ville: String;
26     etat: String;
27     codeZip: String
28   };
29 }
```

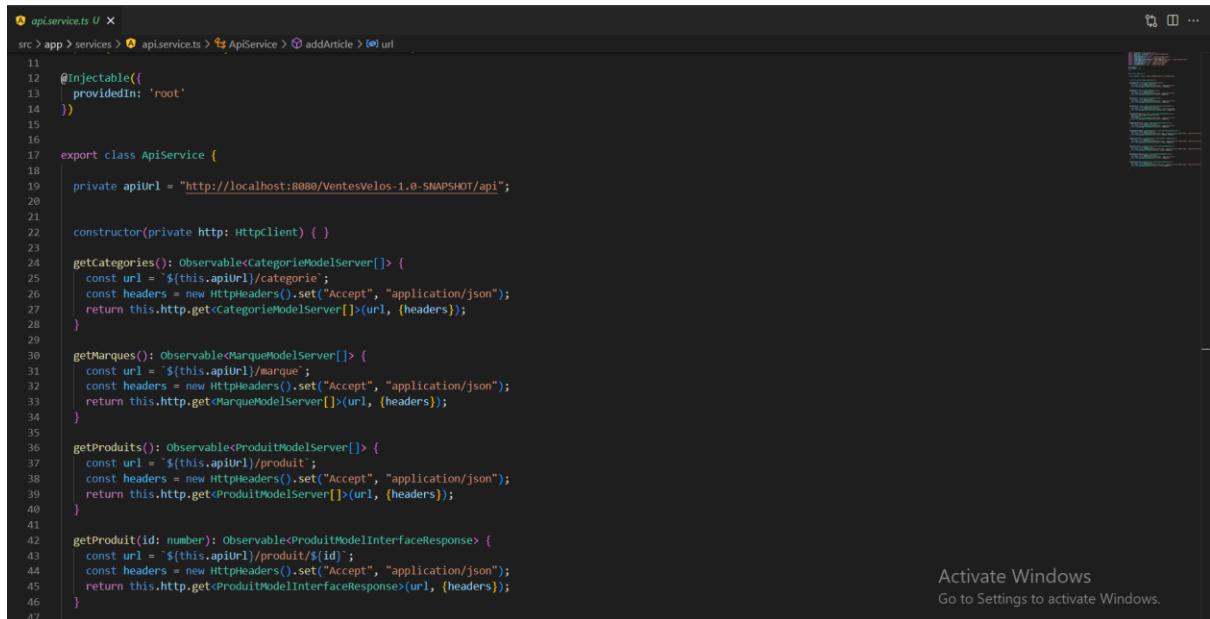
Activate Windows  
Go to Settings to activate Windows.



```
src > app > models > marque.models > MarqueModelServer
1 import { ProductModelServer } from "./product.model";
2
3 export interface MarqueModelServer {
4   id: Number;
5   nom: String;
6   productcollection?: ProductModelServer[];
7 }
```

Activate Windows  
Go to Settings to activate Windows.

Nous créons également les services pour communiquer avec les services Web Restful (api), le panier (cart), les utilisateurs de l'application web (users) et l'authentification (auth)



```
src > app > services > ApiService > ApiService > addArticle > url
1
2 @Injectable({
3   providedIn: 'root'
4 })
5
6 export class ApiService {
7
8   private apiUrl = "http://localhost:8080/ventesVelos-1.0-SNAPSHOT/api";
9
10  constructor(private http: HttpClient) { }
11
12  getCategories(): Observable<CategorieModelServer[]> {
13    const url = `${this.apiUrl}/categorie`;
14    const headers = new HttpHeaders().set("Accept", "application/json");
15    return this.http.get<CategorieModelServer[]>(url, {headers});
16  }
17
18  getMarques(): Observable<MarqueModelServer[]> {
19    const url = `${this.apiUrl}/marque`;
20    const headers = new HttpHeaders().set("Accept", "application/json");
21    return this.http.get<MarqueModelServer[]>(url, {headers});
22  }
23
24  getProducts(): Observable<ProductModelServer[]> {
25    const url = `${this.apiUrl}/produit`;
26    const headers = new HttpHeaders().set("Accept", "application/json");
27    return this.http.get<ProductModelServer[]>(url, {headers});
28  }
29
30  getProduct(id: number): Observable<ProduitModelInterfaceResponse> {
31    const url = `${this.apiUrl}/produit/${id}`;
32    const headers = new HttpHeaders().set("Accept", "application/json");
33    return this.http.get<ProduitModelInterfaceResponse>(url, {headers});
34  }
35
36  getProduit(id: number): Observable<ProduitModelInterfaceResponse> {
37    const url = `${this.apiUrl}/produit/${id}`;
38    const headers = new HttpHeaders().set("Accept", "application/json");
39    return this.http.get<ProduitModelInterfaceResponse>(url, {headers});
40  }
41
42  getProduit(id: number): Observable<ProduitModelInterfaceResponse> {
43    const url = `${this.apiUrl}/produit/${id}`;
44    const headers = new HttpHeaders().set("Accept", "application/json");
45    return this.http.get<ProduitModelInterfaceResponse>(url, {headers});
46  }
47}
```

Activate Windows  
Go to Settings to activate Windows.

```
auth.service.ts U X
src > app > services > auth.service.ts > ...
1 ~ import { Injectable } from '@angular/core';
2 import { UserModelServer } from './models/user.model';
3 import { Router } from '@angular/router';
4 import { CartService } from './cart.service';
5 |
6 @Injectable({
7   providedIn: 'root'
8 })
9 export class AuthService {
10
11   private isLoggedInIn:boolean = false;
12   private userData: UserModelServer | null;
13   private STORAGE_KEY = "users";
14
15   constructor(private router: Router, private cartService: CartService) { }
16
17   login(user: any): boolean {
18     const index = this.getUsers().findIndex(result => (result.client.email === user.client.email) && (result.password === user.password));
19     if (index !== -1) {
20       this.isLoggedInIn = true;
21       this.userData = this.getUsers()[index];
22       this.router.navigate(['']);
23     }
24     return this.isLoggedInIn;
25   }
26
27   logout(): void {
28     this.isLoggedInIn = false;
29     this.userData = null;
30     this.cartService.clearCart();
31     this.router.navigate(['']);
32   }
33
34   getLoggedInIn(): boolean {
35     return this.isLoggedInIn;
36   }
37 }
```

Activate Windows  
Go to Settings to activate Windows.

```
cart.service.ts U X
src > app > services > cart.service.ts > ...
6 }
7
8 export class CartService {
9   private cartItems: CartModelServer = {
10     total: 0,
11     itemsInCart: 0,
12     data: []
13   };
14
15   constructor() { }
16
17   getItems(): CartModelServer{
18     return this.cartItems;
19   }
20
21   addItem(item: CartProduitServer): void {
22     this.cartItems.data.push(item);
23     this.cartItems.itemsInCart += 1;
24     this.calculateTotal();
25   }
26
27   removeItem(item: CartProduitServer): void {
28     const {available, index} = this.checkAvailability(item);
29     if (available) {
30       this.cartItems.data.splice(index, 1);
31     }
32     this.cartItems.itemsInCart -= 1;
33     this.calculateTotal();
34   }
35
36   updateItem(item: CartProduitServer, increaseQuantity: Boolean, quantite?: number): void {
37     const {available, index} = this.checkAvailability(item);
38     if (quantite) {
39       if (available) {
40         if (increaseQuantity) {
41           this.cartItems.data[index].numInCart += quantite;
42         } else {
43       }
44     }
45   }
46
47   calculateTotal(): void {
48     let total = 0;
49     for (let item of this.cartItems.data) {
50       total += item.price * item.numInCart;
51     }
52     this.cartItems.total = total;
53   }
54 }
```

Activate Windows  
Go to Settings to activate Windows.

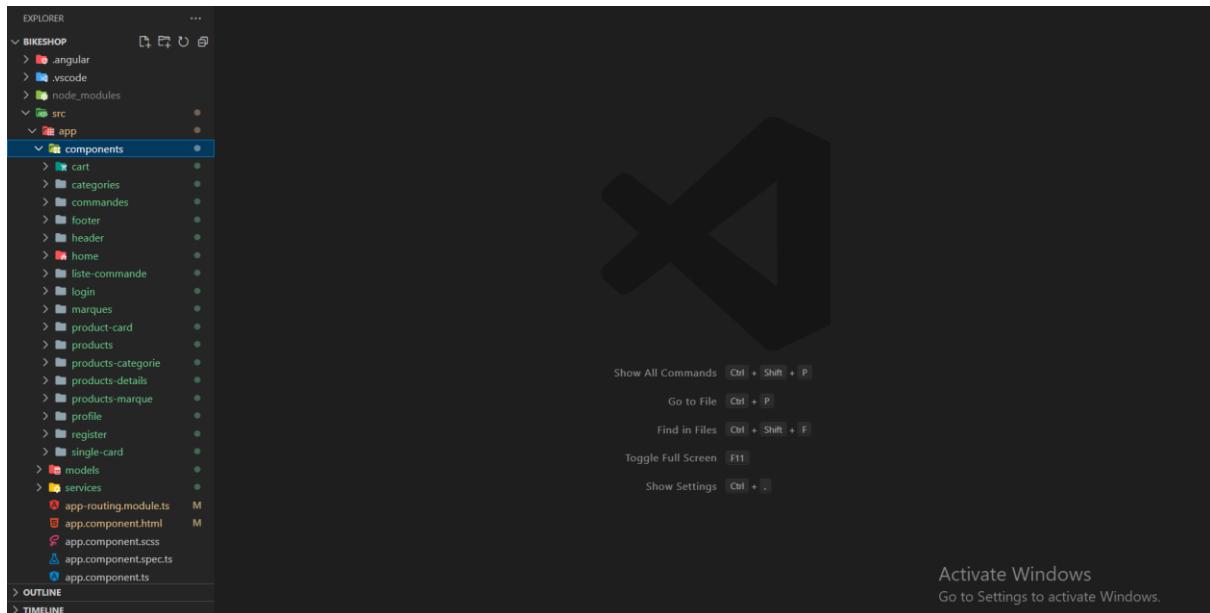
```

src > app > services > users.service.ts > ...
10
11 export class UserService {
12
13   private STORAGE_KEY = "users";
14   private COMMANDE_STORAGE = "num_commande";
15   private CLIENT_STORAGE = "num_client";
16   private users = this.getUsers();
17
18   constructor(private apiService: ApiService, private cartService: CartService, private authService: AuthService) { }
19
20   saveUser(user: UserModelServer): void {
21     user.client.id = this.getClientNumeroStart();
22     user.client.commandeCollection = [];
23     user.cart = this.cartService.getItems();
24     this.apiService.addClient(user.client).subscribe(
25       result => {
26         this.users.push(user);
27         localStorage.setItem(this.STORAGE_KEY, JSON.stringify(this.users));
28         this.authService.login(user);
29       },
30       error => {
31         console.log(`Une erreur s'est produite: ${error}`);
32       }
33     )
34     this.setClientNumeroStart();
35   }
36
37   updateUser(user: UserModelServer): void {
38     this.apiService.addClient(user.client).subscribe(
39       result => {
40         this.deleteUser(user);
41         this.users.push(user);
42         localStorage.setItem(this.STORAGE_KEY, JSON.stringify(this.users));
43       },
44       error => {
45         console.log(`Une erreur s'est produite: ${error}`);
46       }
47     )
48   }

```

Activate Windows  
Go to Settings to activate Windows.

Nous créons également nos composants qui pourront être réutiliser plus tard dans nos applications. Nous créons donc l'ensemble des composants lister dans l'images ci-dessous



Nous importons les modules HttpClient, FormsModule, FontAwesomeModule, AppRoutingModules et nous ajoutons nos services au en tant que *providers*

```

src > app > app.module.ts ...
28 import { ListeCommandeComponent } from './components/liste-commande/liste-commande.component';
29
30
31 @NgModule({
32   declarations: [
33     AppComponent,
34     HeaderComponent,
35     FooterComponent,
36     HomeComponent,
37     CategoriesComponent,
38     MarquesComponent,
39     SingleCardComponent,
40     ProductCardComponent,
41     ProductsComponent,
42     ProductsCategorieComponent,
43     ProductsMarqueComponent,
44     ProductsDetailsComponent,
45     CartComponent,
46     CommandesComponent,
47     LoginComponent,
48     RegisterComponent,
49     ProfileComponent,
50     ListeCommandeComponent,
51   ],
52   imports: [
53     BrowserModule,
54     AppRoutingModule,
55     FontAwesomeModule,
56     HttpClientModule,
57     FormsModule
58   ],
59   providers: [CartService, ApiService, UserService, AuthService],
60   bootstrap: [AppComponent]
61 })
62 export class AppModule { }
63

```

Activate Windows  
Go to Settings to activate Windows.

Nous ajoutons ensuite des routes pour nos différents composants pour gérer le routage et naviguer entre les pages.

```

src > app > app-routing.module.ts ...
17 const routes: Routes = [
18   {
19     path: '', component: HomeComponent
20   },
21   {
22     path: 'register', component: RegisterComponent
23   },
24   {
25     path: 'login', component: LoginComponent
26   },
27   {
28     path: 'profile', component: ProfileComponent
29   },
30   {
31     path: 'liste-commandes', component: ListeCommandeComponent
32   },
33   {
34     path: 'categories', component: CategoriesComponent
35   },
36   {
37     path: 'categories/:slug', component: ProductsCategorieComponent
38   },
39   {
40     path: 'marques', component: MarquesComponent
41   },
42   {
43     path: 'marques/:slug', component: ProductsMarqueComponent
44   },
45   {
46     path: 'produits', component: ProductsComponent
47   },
48   {
49     path: 'produits/:id', component: ProductsDetailsComponent
50   },
51   {
52     path: 'cart', component: CartComponent
53   },
54   {
55     path: 'commande', component: CommandesComponent
56   }
57 ];
58
59 @NgModule({
60   imports: [RouterModule.forRoot(routes)],
61   exports: [RouterModule]
62 })
63

```

Activate Windows  
Go to Settings to activate Windows.

Au final nous obtenons donc le client web Angular avec les interfaces suivantes

- Page Accueil

The screenshot shows a web browser window with the title bar "BikeShop" and the address bar "localhost:4200". The page content includes a logo for "BikeShop", navigation links for "Accueil", "Tout", "Categories", and "Marques", a "Connexion" button, and a shopping cart icon. A main heading "Recherche, Achat, Livraison" is displayed, followed by a descriptive text about the website's purpose. Below the text is a photograph of a bicycle shop interior with many bicycles on display. At the bottom of the page is a footer with links for "A PROPOS DE NOUS", "SUPPORT", "INFORMATIONS", and "SERVICES", along with social media icons and a "Sign in" button.

## ➤ Page Inscription

BikeShop

Connexion 

### Création de compte

Prenom  Nom  Telephone   
 Adresse   
 Etat  Code Zip  Ville   
 Email   
 Mot de passe  Confirmer mot de passe

Vous avez déjà un compte? [Connectez-vous](#)

[S'inscrire](#)

#### A PROPOS DE NOUS

Nous sommes une boutique de ventes de vélos et voici notre portail web pour la vente en ligne

#### SUPPORT

FAQ  
Knowledge base

#### INFORMATIONS

A propos de nous  
Contactez nous

#### SERVICES

Mon compte  
Voir panier

Activate Windows  
Go to Settings to activate Windows



BikeShop

Accueil Tout Categories Marques

Connexion 

### Création de compte

Prenom  
John

Nom  
Doe

Telephone  
791030018

Adresse  
Thies Route VCN Nord

Etat  
CA

Code Zip  
A10

Ville  
Thies

Email  
test@example.com

Mot de passe  
\*\*\*\*\*

Confirmer mot de passe  
\*\*\*\*\*

Vous avez déjà un compte? [Connectez-vous](#)

[S'inscrire](#)

#### A PROPOS DE NOUS

Nous sommes une boutique de ventes de vélos et voici notre portail web pour la vente en ligne

#### SUPPORT

FAQ  
Knowledge base

#### INFORMATIONS

A propos de nous  
Contactez nous

#### SERVICES

Mon compte  
Voir panier

Activate Windows  
Go to Settings to activate Windows



BikeShop

Accueil Tout Categories Marques

Mon Compte 

### Recherche, Achat, Livraison

BikeShop est une la vitrine Web de la boutique de ventes vélos où vous pouvez passer des commandes en lignes. Grâce à un navigateur Web, les clients peuvent rechercher un vélo qui leur convient par catégorie ou marque, puis leur ajouter au panier et enfin passer une commande en utilisant une transaction par carte de crédit.



#### A PROPOS DE NOUS

Nous sommes une boutique de ventes de

#### SUPPORT

FAQ

#### INFORMATIONS

A propos de nous

#### SERVICES

Mon compte

Activate Windows  
Go to Settings to activate Windows

## ➤ Page Connexion

The screenshot shows the BikeShop login page. At the top, there is a header with the logo "BikeShop", a navigation bar with links "Accueil", "Tout", "Categories", "Marques", a "Connexion" button, and a shopping cart icon showing "0". Below the header, a section titled "Connectez Vous" contains two input fields: "Email" and "Mot de passe". A red-bordered "Connexion" button is centered below the fields. Below the button, a link "Pas encore de compte? Inscrivez-vous" is visible. At the bottom of the page, there are four main sections: "A PROPOS DE NOUS", "SUPPORT", "INFORMATIONS", and "SERVICES".

A PROPOS DE NOUS	SUPPORT	INFORMATIONS	SERVICES
Nous sommes une boutique de ventes de vélos et voici notre portail web pour la vente en ligne	FAQ Knowledge base Garantie Nouveauté	A propos de nous Contactez nous Politique de confidentialité Termes & Conditions	Mon compte Voir panier Aide Activate Windows Go to Settings to activate Windows.

This screenshot shows the same BikeShop login page as above, but with user input. The "Email" field now contains "test@example.com" and the "Mot de passe" field contains a series of asterisks. The rest of the page structure remains identical to the first screenshot.

A PROPOS DE NOUS	SUPPORT	INFORMATIONS	SERVICES
Nous sommes une boutique de ventes de vélos et voici notre portail web pour la vente en ligne	FAQ Knowledge base Garantie Nouveauté	A propos de nous Contactez nous Politique de confidentialité Termes & Conditions	Mon compte Voir panier Aide Activate Windows Go to Settings to activate Windows.

**Recherche, Achat, Livraison**

BikeShop est une la vitrine Web de la boutique de ventes vélos où vous pouvez passez des commandes en lignes. Grâce à un navigateur Web, les clients peuvent rechercher un vélo qui leur convient par catégorie ou marque, puis leur ajouter au panier et enfin passer une commande en utilisant une transaction par carte de crédit.

A PROPOS DE NOUS SUPPORT INFORMATIONS SERVICES

Nous sommes une boutique de ventes de [FAQ](#) [A propos de nous](#) [Mon compte](#)

Activate Windows  
Go to Settings to activate Windows

## ➤ Options Mon Compte

**Recherche, Achat, Livraison**

BikeShop est une la vitrine Web de la boutique de ventes vélos où vous pouvez passez des commandes en lignes. Grâce à un navigateur Web, les clients peuvent rechercher un vélo qui leur convient par catégorie ou marque, puis leur ajouter au panier et enfin passer une commande en utilisant une transaction par carte de crédit.

A PROPOS DE NOUS SUPPORT INFORMATIONS SERVICES

Nous sommes une boutique de ventes de [FAQ](#) [A propos de nous](#) [Mon compte](#)

Activate Windows  
Go to Settings to activate Windows

## Informations

BikeShop

Acceuil Tout Categories Marques

Mon Compte 0

### Modification de compte

Prenom: John

Nom: Doe

Telephone: 791030018

Adresse: Thies Route VCN Nord

Etat: CA

Code Zip: A10

Ville: Thies

Email: test@example.com

Mot de passe: \*\*\*\*\*

Confirmer mot de passe:

**Modifier**

**Supprimer**

A PROPOS DE NOUS SUPPORT INFORMATIONS SERVICES Activate Windows  
Go to Settings to activate Windows

Nous sommes une boutique de ventes de

FAQ

A propos de nous

Mon compte

## Commandes

BikeShop

Acceuil Tout Categories Marques

Mon Compte 0

### Aucune Commande

**Retourner**

A PROPOS DE NOUS SUPPORT INFORMATIONS SERVICES Activate Windows  
Go to Settings to activate Windows

Nous sommes une boutique de ventes de vélos et voici notre portail web pour la vente en ligne

FAQ

Knowledge base

Garantie

Nouveauté

A propos de nous

Contactez nous

Politique de confidentialité

Termes & Conditions

Mon compte

Voir panier

Aide

Copyright © All rights reserved Jakarta EE

## Déconnexion

## Recherche, Achat, Livraison

BikeShop est une la vitrine Web de la boutique de ventes vélos où vous pouvez passez des commandes en lignes. Grâce à un navigateur Web, les clients peuvent rechercher un vélo qui leur convient par catégorie ou marque, puis leur ajouter au panier et enfin passer une commande en utilisant une transaction par carte de crédit.



### ➤ Voir tous les produits

MOUNTAIN BIKES	MOUNTAIN BIKES	MOUNTAIN BIKES	MOUNTAIN BIKES
TREK 820 - 2016... \$380 Trek	RITCHIE TIMBERWOLF F... \$750 Ritchey	SURLY WEDNESDAY FRAM... \$1000 Surly	TREK FUEL EX 8 29 - ... \$2900 Trek

**Sélectionner un produit spécifique**



Qty

**AJOUTER AU PANIER**

**TREK 820 - 2016**

**\$380.00**

CATEGORY: MOUNTAIN BIKES  
MARQUE: TREK  
ANNÉE: 2016

**Santa Cruz Bikes**

- 📍 3700 Portola Drive, Santa Cruz, CA, 95060
- 📞 (831) 476-4321
- ✉️ santacruz@bikes.shop
- 👁 27

**Baldwin Bikes**

- 📍 4200 Chestnut Lane, Baldwin, NY, 11432
- 📞 (516) 379-8888
- ✉️ baldwin@bikes.shop
- 👁 14

**Rowlett Bikes**

- 📍 8000 Fairway Avenue, Rowlett, TX, 75088
- 📞 (972) 530-5555
- ✉️ rowlett@bikes.shop
- 👁 14

Activate Windows  
Go to Settings to activate Windows

## ➤ Voir les catégories

**Children Bicycles**  
[VOIR TOUT](#)



**Comfort Bicycles**  
[VOIR TOUT](#)



**Cruisers Bicycles**  
[VOIR TOUT](#)



**Cyclocross Bicycles**  
[VOIR TOUT](#)



**Electric Bikes**  
[VOIR TOUT](#)



**Mountain Bikes**  
[VOIR TOUT](#)



**Road**  
[VOIR TOUT](#)



Activate Windows  
Go to Settings to activate Windows

## Produits d'une catégorie

BikeShop

Connexion 0

## COMFORT BICYCLES

COMFORT BICYCLES  
ELECTRA TOWNIE ORIGI...  
\$550  
Electra

COMFORT BICYCLES  
ELECTRA TOWNIE ORIGI...  
\$500  
Electra

COMFORT BICYCLES  
ELECTRA TOWNIE ORIGI...  
\$600  
Electra

COMFORT BICYCLES  
ELECTRA TOWNIE ORIGI...  
\$490  
Electra

Activate Windows  
Go to Settings to activate Windows

### Selectionner un produit

BikeShop

Connexion 0

### ELECTRA TOWNIE ORIGINAL 7D - 2015/2016

**\$500.00**

CATEGORY: COMFORT BICYCLES  
MARQUE: ELECTRA  
ANNÉE: 2016

qty 1  AJOUTER AU PANIER

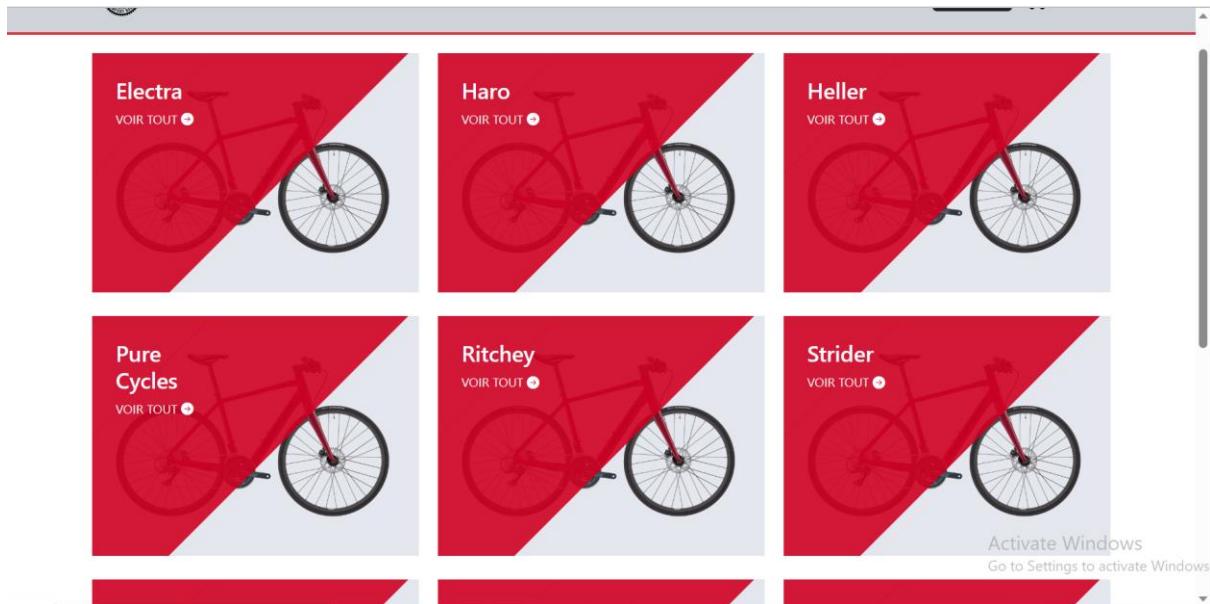
Santa Cruz Bikes  
3700 Portola Drive, Santa Cruz, CA, 95060  
(831) 476-4321  
santacruz@bikes.shop  
10

Baldwin Bikes  
4200 Chestnut Lane, Baldwin, NY, 11432  
(516) 379-8888  
baldwin@bikes.shop  
18

Rowlett Bikes  
8000 Fairway Avenue, Rowlett, TX, 75088

Activate Windows  
Go to Settings to activate Windows

➤ Voir les marques



## Produits d'une marque

A screenshot of a website showing a grid of Haro mountain bikes. The grid has two rows of four items each. Each item shows a red Haro mountain bike with its name and price: HARO FLIGHTLINE ONE ... \$380, HARO FLIGHTLINE TWO ... \$550, HARO SHIFT R3 - 2017... \$1470, and HARO SR 1.1 - 2017... \$540. The 'HARO' brand name is centered above the grid. A watermark for 'Activate Windows Go to Settings to activate Windows' is visible in the bottom right corner.

## Selectionner un produit

➤ Ajout de produits au panier

BikeShop

Accueil Tout Categories Marques

Connexion 1

Image	Nom	Prix	Quantite	Magasin	Sous Total
	Haro Flightline Two 26 Plus - 2017	\$550.00	- 6 +	Santa Cruz Bikes	\$3,300.00
			<b>TOTAL</b>	<b>\$3,300.00</b>	

[Retourner](#) [Créer un compte pour passer vos commandes](#)

---

**A PROPOS DE NOUS**

Nous sommes une boutique de ventes de vélos et voici notre portail web pour la vente en ligne

**SUPPORT**

FAQ  
Knowledge base  
Garantie  
Nouveauté

**INFORMATIONS**

A propos de nous  
Contactez nous  
Politique de confidentialité  
Termes & Conditions

**SERVICES**

Mon compte  
Voir panier  
Aide

Activate Windows  
Go to Settings to activate Windows

## Créer un compte pour la commande

BikeShop

Accueil Tout Categories Marques

Connexion 1

### Création de compte

Prenom John	Nom Dieng	Telephone 791030018
Adresse Thies Route VCN Nord		
Etat CA	Code Zip A10	Ville Thies
Email email@example.com		
Mot de passe *****	Confirmer mot de passe *****	

Vous avez déjà un compte? [Connectez-vous](#)

[S'inscrire](#)

---

[A PROPOS DE](#)

[SUPPORT](#)

[INFORMATIONS](#)

[SERVICES](#)

Activate Windows  
Go to Settings to activate Windows

BikeShop

Accueil Tout Categories Marques

Mon Compte

1

Image	Nom	Prix	Quantite	Magasin	Sous Total
	Haro Flightline Two 26 Plus - 2017	\$550.00	6	Santa Cruz Bikes	\$3,300.00
			TOTAL		<b>\$3,300.00</b>

Retourner

Payer

---

**A PROPOS DE NOUS**

Nous sommes une boutique de ventes de vélos et voici notre portail web pour la vente en ligne

**SUPPORT**

FAQ  
Knowledge base  
Garantie  
Nouveauté

**INFORMATIONS**

A propos de nous  
Contactez nous  
Politique de confidentialité  
Termes & Conditions

**SERVICES**

Mon compte  
Voir panier  
Aide

Activate Windows  
Go to Settings to activate Windows

## ➤ Payer et passer commande

BikeShop

Accueil Tout Categories Marques

Mon Compte

1

**DETAILS LIVRAISON**

Prenom: Dieng  
Nom: Dieng  
Telephone: 791030018

Adresse Mail: email@example.com

Adresse: Thies Route VCN Nord

Etat: CA  
Code Zip: A10  
Ville: Thies

Livraison Voulue: 06/29/2023

**VOTRE COMMANDE**

PRODUIT	SOUS TOTAL
6x Haro Flightline Two 26 Plus - 2017	\$3,300.00
<b>TOTAL</b>	<b>\$3,300.00</b>

Placer Commande

Activate Windows  
Go to Settings to activate Windows

 BikeShop

Accueil Tout Categories Marques

Mon Compte 

# Merci!

Votre commande a été ajoutée.

Numéro Commande : **1616**

Image	Titre	Prix	Quantité
	Haro Flightline Two 26 Plus - 2017	\$550.00	6

## A PROPOS DE NOUS

Nous sommes une boutique de vente de vélos et voici notre portail

## SUPPORT

FAQ

Knowledge base

## INFORMATIONS

A propos de nous

Contactez nous

## SERVICES

Mon compte

Activate Windows  
Go to Settings to activate Windows.

Voir panier

## ➤ Lister les commandes

 BikeShop

Accueil Tout Categories Marques

Mon Compte 

# Merci!

Votre commande a été ajoutée.

Numéro Commande : **1616**

Image	Titre	Prix	Quantité
	Haro Flightline Two 26 Plus - 2017	\$550.00	6

## A PROPOS DE NOUS

Nous sommes une boutique de vente de vélos et voici notre portail

## SUPPORT

FAQ

Knowledge base

## INFORMATIONS

A propos de nous

Contactez nous

## SERVICES

Mon compte

Activate Windows  
Go to Settings to activate Windows.

Voir panier

The screenshot shows the BikeShop website's command history page. At the top, there is a navigation bar with the logo "BikeShop", links for "Accueil", "Tout", "Categories", and "Marques", a "Mon Compte" button, and a shopping cart icon with a "0" notification.

**Liste de vos commandes**

Numéro Commande : **1616**

Image	Titre	Prix	Quantité
	Haro Flightline Two 26 Plus - 2017	\$550.00	6

**A PROPOS DE NOUS**

Nous sommes une boutique de ventes de vélos et voici notre portail web pour la vente en ligne

**SUPPORT**

- FAQ
- Knowledge base
- Garantie
- Nouveauté

**INFORMATIONS**

- A propos de nous
- Contactez nous
- Politique de confidentialité
- Termes & Conditions

**SERVICES**

- Mon compte
- Voir panier
- Aide
- Activate Windows  
Go to Settings to activate Windows

## ➤ Modification du stock

The screenshot shows a product page for a Haro Flightline Two 26 Plus - 2017 bicycle. On the left, there is a large image of the red bicycle, a quantity selector set to "1", and a red "AJOUTER AU PANIER" button. To the right, there is detailed product information:

**HARO FLIGHTLINE TWO 26 PLUS - 2017**  
**\$550.00**

CATEGORY: MOUNTAIN BIKES  
MARQUE: HARO  
ANNÉE: 2017

**Santa Cruz Bikes**

- 3700 Portola Drive, Santa Cruz, CA, 95060
- (831) 476-4321
- santacruz@bikes.shop
- 7

**Baldwin Bikes**

- 4200 Chestnut Lane, Baldwin, NY, 11432
- (516) 379-8888
- baldwin@bikes.shop
- 18

**Activate Windows**  
Go to Settings to activate Windows

Nous pouvons également constater que ces modifications apparaissent au niveau de la base de données

## ➤ Nous voyons ici les deux clients que nous avons eu à créer

The screenshot shows a database interface with two tables: "client" and "personne".

**client**

	id	dtype	email	nom	prenom	telephone
1	1459	Client	email@example.com	Dieng	John	791030018
2	1458	Client	test@example.com	Doe	John	791030018

**personne**

	id	adresse	code_zip	etat	ville
1	1459	Thies Route VCN Nord	A10	CA	Thies
2	1458	Thies Route VCN Nord	A10	CA	Thies

## ➤ Nous pouvons également voir la commande créée

commande

	numero	date_commande	date_livraison	date_livraison_voulu	statut	client_id	magasin_id	vendeur_id
1	1616	2023-06-20	<null>	2023-06-29	1	1459	1	1449

➤ Nous pouvons voir l'article commande

article\_commande

	prix_depart	quantite	remise	ligne	numero_commande	produit_id
1	550	6	0	1	1616	38

➤ La modification du stock

console\_3

```
1 ✓ SELECT * FROM stock WHERE produit_id=38 AND magasin_id=1;
```

Services

- GlassFish Server
- Running
- GlassFish 7.0.0 [local]
- VentesVelos:wa

Output

	quantite	produit_id	magasin_id
1	7	38	1