



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.....	7
1.	Développement des services web.....	7
2.	Documentation des services web.....	20
III.	Test des services web	29
IV.	Client Angular+bootstrap.....	37

I. Définir les ressources

1. Identification des ressources

Nous avons décidé pour le client 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.

Pour le client mobile, nous allons reproduire les mêmes fonctionnalités que celles disponibles sur l'application Jakarta en permettant à l'utilisateur de créer, modifier ou supprimer les entités disponibles au niveau de notre application.

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
- EmployeResource pour la gestion des employés
- MagasinResource pour la gestion des magasins

2. Concevoir les endpoints

Endpoint pour ArticleCommandeResource

Pour cette ressource nous utilisons :

- L'URI « /article » avec comme actions possibles :

- Enregistrer ou modifier des articles commandes. Pour cela nous utiliserons donc la méthode PUT afin de pouvoir enregistrer et modifier en cas d'erreur les articles commandes.
 - Trouver tous les articles commandes. Pour cela nous utiliserons donc la méthode GET
- L'URI « /article/{numero}/{ligne} » avec comme actions possibles
- Supprimer un article commande. Pour cela nous utiliserons donc la méthode DELETE en passant le numéro de commande et la ligne comme PathParams.

Endpoint pour CategorieResource

Pour cette ressource nous utilisons :

➤ L'URI « /categorie » avec comme actions possibles :

 - Enregistrer ou modifier une catégorie. Pour cela nous utiliserons donc la méthode PUT afin de pourvoir enregistrer et modifier en cas d'erreur les catégories
 - Trouver l'ensemble des catégories disponibles. Pour cela nous utiliserons donc la méthode GET.

➤ L'URI « /categorie/{id} » avec comme actions possibles

 - 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.
 - Supprimer une catégorie. Pour cela nous utiliserons donc la méthode DELETE en passant l'id de la catégorie comme un PathParam.

Endpoint pour ClientResource

Pour cette ressource nous utilisons :

➤ L'URI « /client » avec comme actions possibles :

 - Enregistrer un nouveau client ou la modification de ses informations. Pour cela nous utiliserons donc la méthode PUT.
 - Trouver l'ensemble des clients disponibles. Pour cela nous utiliserons donc la méthode GET

➤ L'URI « /client/{id} » avec comme actions possibles :

 - 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.
 - Supprimer un client. Pour cela nous utiliserons donc la méthode DELETE en passant l'id du client comme un PathParam.

Endpoint pour CommandeResource

Pour cette ressource nous utilisons :

- L'URI « /commande » avec comme actions possibles
 - Enregister une commande ainsi que la modification en cas d'erreur. Pour cela nous utiliserons donc la méthode PUT.
 - Trouver l'ensemble des commandes. Pour cela nous utiliserons donc la méthode GET
- L'URI « /commande/{numero} » avec comme actions possibles :
 - Trouver l'ensemble des articles commandes appartenant à la commande spécifiée. Pour cela nous utiliserons donc la méthode GET en passant le numéro du client comme un PathParam
 - Supprimer une commande. Pour cela nous utiliserons donc la méthode DELETE en passant le numéro de la commande comme un PathParam

Endpoint pour MarqueResource

Pour cette ressource nous utilisons :

- L'URI « /marque » avec comme actions possibles :
 - Enregistrer ou modifier une marque. Pour cela nous utiliserons la méthode PUT
 - Trouver l'ensemble des marques disponibles. Pour cela nous utiliserons donc la méthode GET.
- L'URI « /marque/{id} » avec comme actions possibles :
 - 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.
 - Supprimer une marque. Pour cela nous utiliserons la méthode DELETE en passant l'id de la marque comme un PathParam

Endpoint pour ProduitResource

Pour cette ressource nous utilisons :

- L'URI « /produit » avec comme actions possibles :
 - Enregistrer ou modifier un produit. Pour cela nous utiliserons la méthode PUT
 - Trouver l'ensemble des produits disponibles. Pour cela nous utiliserons donc la méthode GET
- L'URI « /produit/{id} » avec comme actions possibles :
 - Trouver les informations d'un produit spécifique. Pour cela nous utiliserons également la méthode GET en passant l'id du produit comme un PathParam
 - Supprimer un produit. Pour cela nous utiliserons donc la méthode DELETE en passant l'id du produit comme un PathParam

Endpoint pour StockResource

Pour cette ressource nous utilisons :

- L'URI « /stock » avec comme actions possibles :
 - Enregistrer ou modifier un stock. Pour cela nous utiliserons donc la méthode PUT
 - Supprimer un stock. Pour cela nous utiliserons la méthode DELETE
 - Trouver la liste des stocks présents. Pour cela nous utiliserons donc la méthode GET.
- L'URI « /stock/{magasin_id}/{produit_id} » avec comme actions possibles :
 - 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.
 - 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

Endpoint pour EmployeResource

Pour cette ressource nous utilisons :

- L'URI « /employe » avec comme actions possibles :
 - Enregistrer ou modifier un employé. Pour cela nous utiliserons la méthode PUT
 - Trouver l'ensemble des employés disponibles. Pour cela nous utiliserons donc la méthode GET.
- L'URI « employe/{id} » avec comme actions possibles :
 - Supprimer un employé. Pour cela nous utiliserons la méthode DELETE en passant l'id de la marque comme un PathParam

Endpoint pour MagasinResource

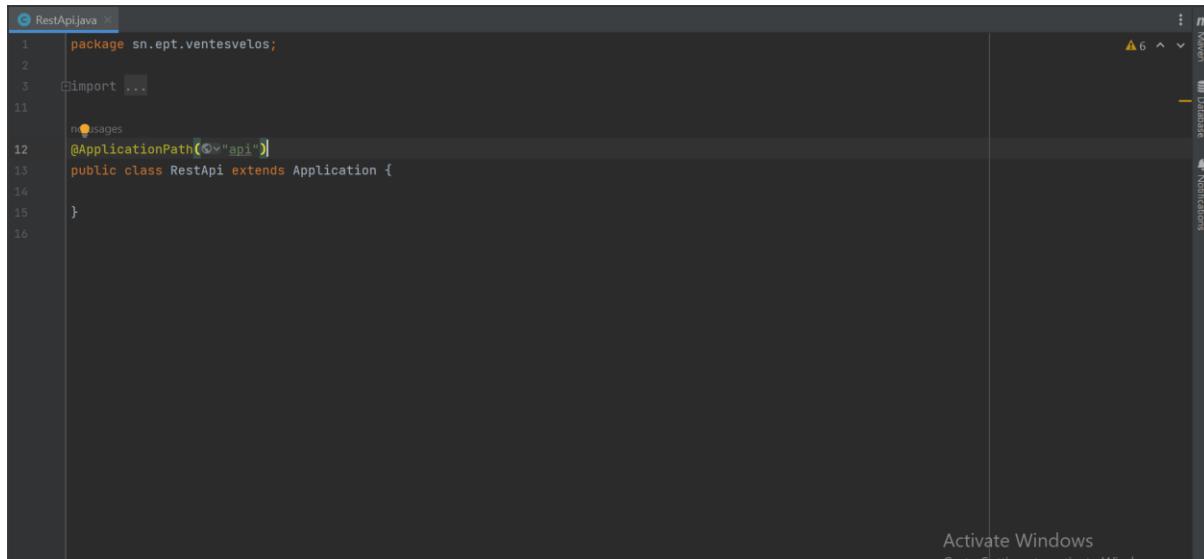
Pour cette ressource nous utilisons :

- L'URI « /magasin » avec comme actions possibles :
 - Enregistrer ou modifier un magasin. Pour cela nous utiliserons donc la méthode PUT afin de pourvoir enregistrer et modifier en cas d'erreur les catégories
 - Trouver l'ensemble des magasins disponibles. Pour cela nous utiliserons donc la méthode GET.
- L'URI « /magasin/{id} » avec comme actions possibles :
 - Supprimer un magasin. Pour cela nous utiliserons donc la méthode DELETE en passant l'id du magasin comme un PathParam

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".



The screenshot shows a Java code editor with a dark theme. The file 'RestApi.java' is open, containing the following code:

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

The code defines a class 'RestApi' that extends 'Application'. It uses the `@Path("api")` annotation to specify the base path for the API. The code editor interface includes tabs for Maven, Database, and Notifications, and a status bar at the bottom.

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`.

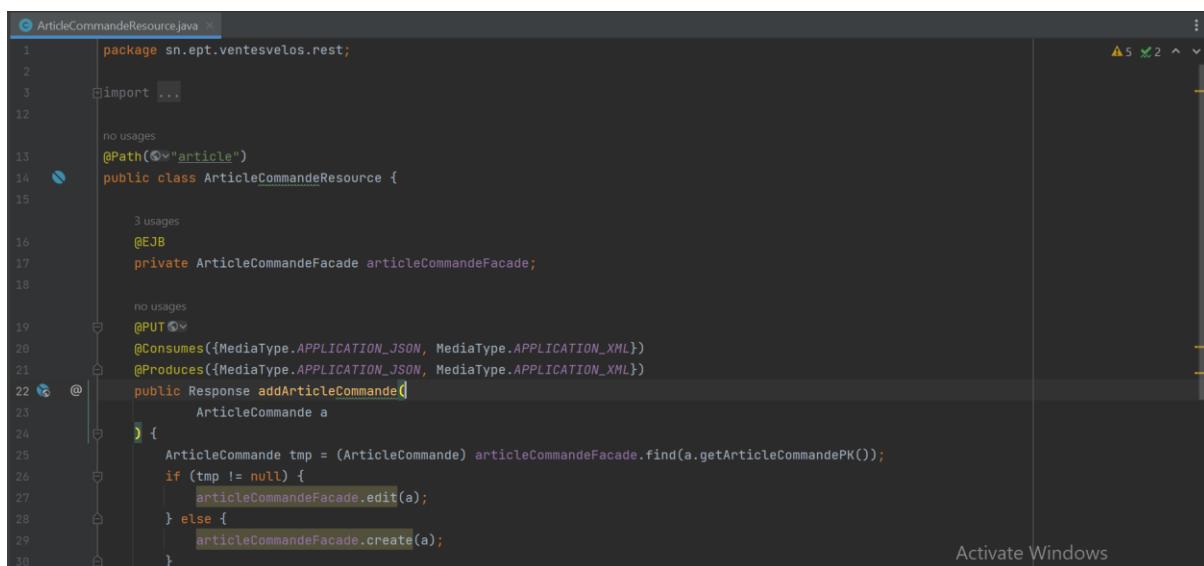
Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec `@GET`, `@PUT`, `@DELETE` suivant le type de requête devant permettre d'y accéder. En utilisant `@Consumes` (`{MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}`) ou `@Produces` (`{MediaType.APPLICATION_JSON, 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 faite aux formats `JSON` et `XML`.

On procède de manière similaire en créant la méthode `getArticlesCommandes()` avec l'annotation `@GET`.

Nous créons également la méthode `deleteArticleCommande()` dans laquelle nous supprimons un produit spécifique dont le numéro de commande et la ligne sont passés en paramètre. On spécifie l'annotation `@DELETE` et `@Path` en spécifiant les paramètres de chemin d'accès `{numero}/{ligne}`. 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("article")
7 public class ArticleCommandeResource {
8
9     3 usages
10    @EJB
11    private ArticleCommandeFacade articleCommandeFacade;
12
13    no usages
14    @PUT
15    @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
16    @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17    public Response addArticleCommande(
18        ArticleCommande a
19    ) {
20        ArticleCommande tmp = (ArticleCommande) articleCommandeFacade.find(a.getArticleCommandePK());
21        if (tmp != null) {
22            articleCommandeFacade.edit(a);
23        } else {
24            articleCommandeFacade.create(a);
25        }
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`.

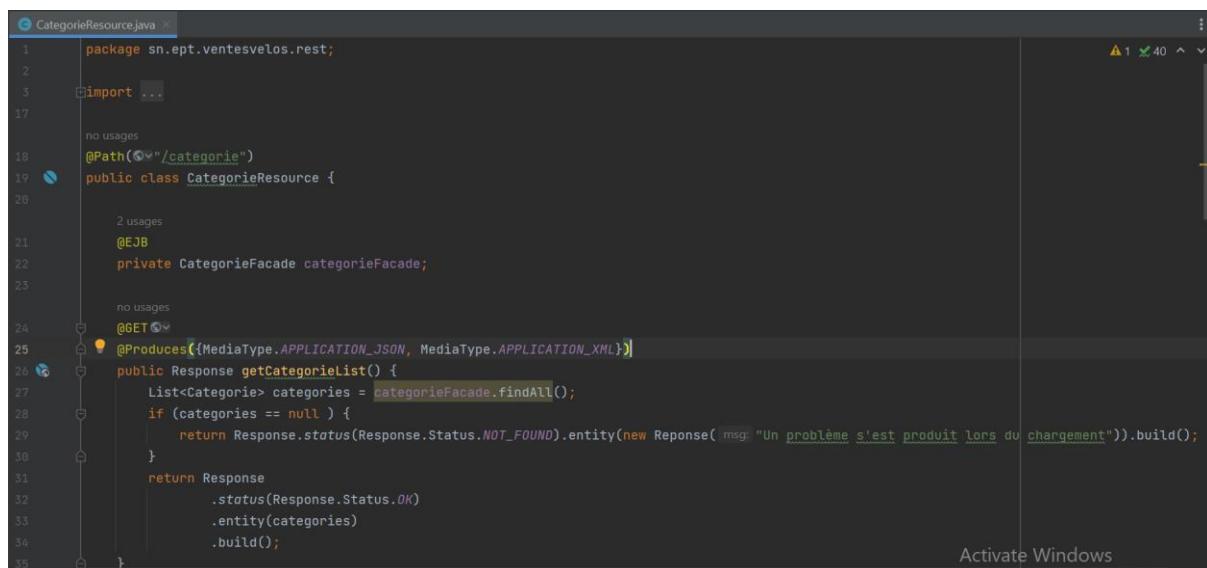
Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec `@GET`, `@PUT`, `@DELETE` suivant le type de requête devant permettre d'y accéder. En utilisant `@Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})` ou `@Produces ({MediaType.APPLICATION_JSON, 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 `getCategoriaList()` 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.

On procède de manière similaire en créant la méthode `addCategoria()` avec l'annotation `@PUT`.

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.

On procède de manière similaire en créant la méthode `deleteCategoria()` avec l'annotation `@DELETE`.



The screenshot shows a code editor window for a Java file named `CategorieResource.java`. The code defines a RESTful service for categories. It includes annotations for `@Path`, `@EJB`, `@GET`, and `@Produces`. The `@Produces` annotation specifies `{MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}`. The `getCategoriaList()` method retrieves all categories from a facade and returns them as a JSON or XML response. If no categories are found, it returns a 404 error message.

```
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,`

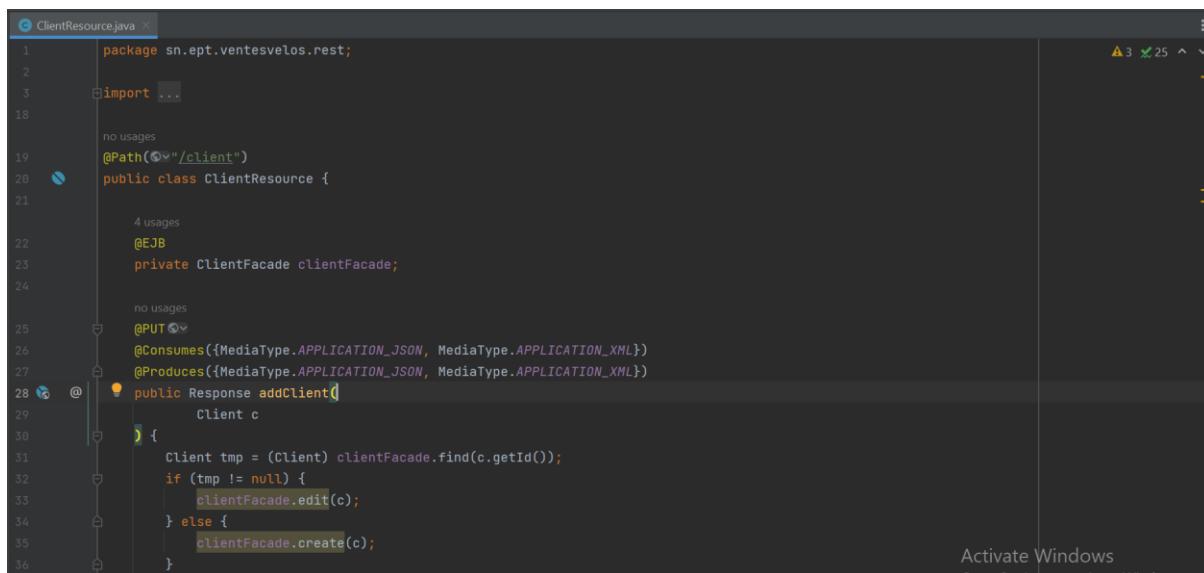
`MediaType.APPLICATION_XML})` pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant le type de requête

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.

On procède de manière similaire en créant la méthode `getClientList()` avec l'annotation `@GET`.

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.

On procède de manière similaire en créant la méthode `deleteClient()` avec l'annotation `@DELETE`.



```
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

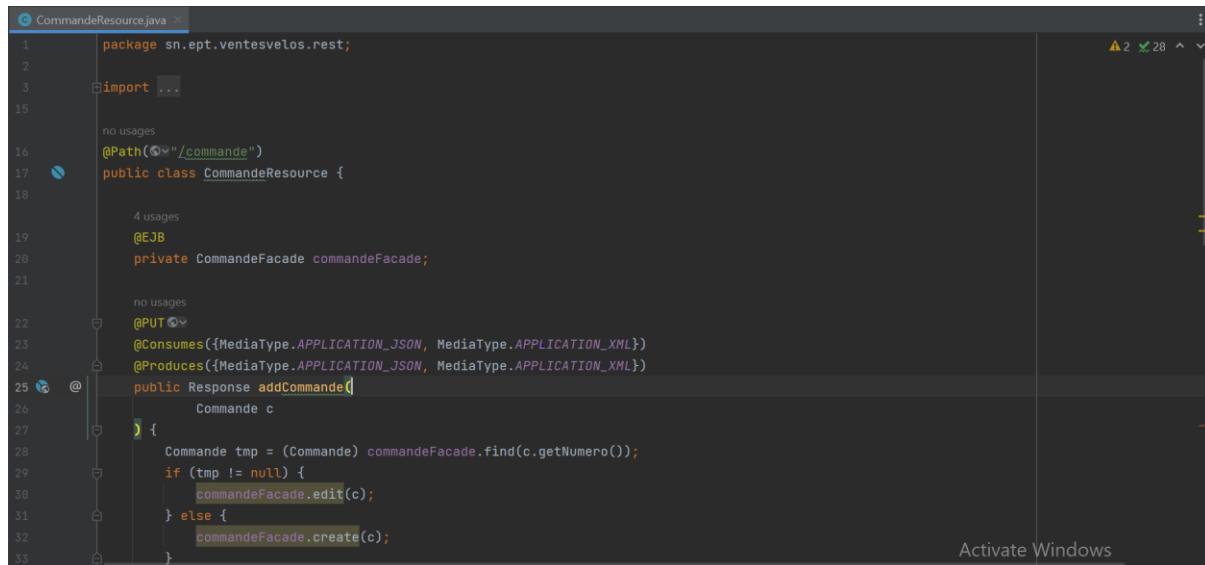
requête devant permettre d'y accéder. En utilisant @Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}) ou @Produces ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}) pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant 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 faite aux formats JSON et XML.

On procède de manière similaire en créant la méthode getCommandeList() avec l'annotation @GET.

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

On procède de manière similaire en créant la méthode deleteCommande() avec l'annotation @DELETE.



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

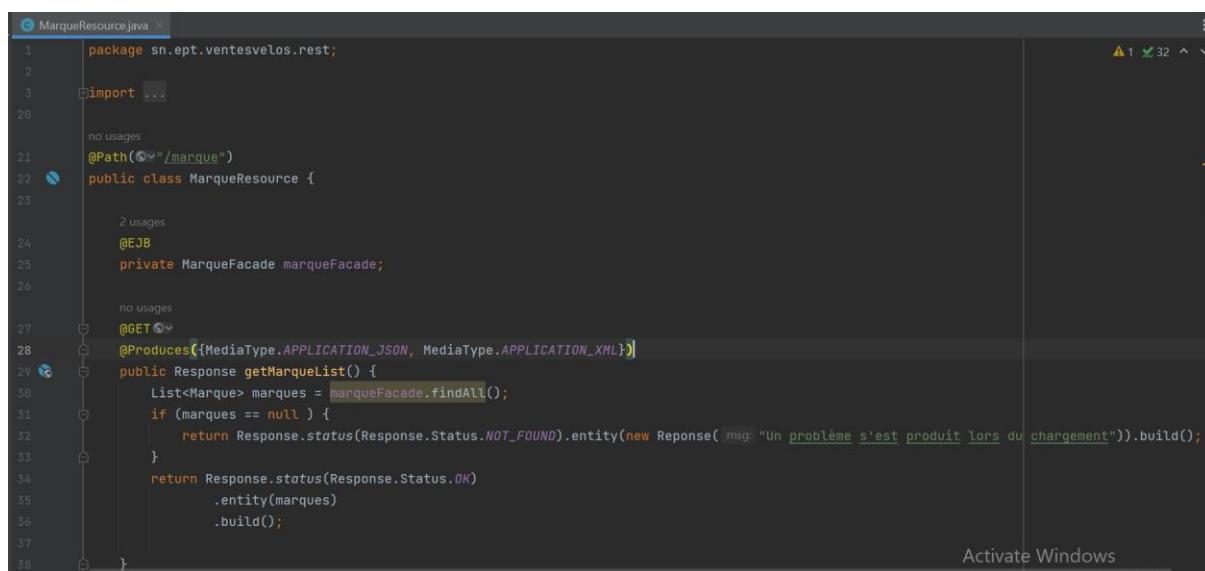
Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec @GET, @PUT, @DELETE suivant le type de requête devant permettre d'y accéder. En utilisant @Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}) ou @Produces ({MediaType.APPLICATION_JSON, 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.

On procède de manière similaire en créant la méthode addMarque() avec l'annotation @PUT.

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.

On procède de manière similaire en créant la méthode deleteMarque() avec l'annotation @DELETE.



The screenshot shows the code editor of an IDE displaying the `MarqueResource.java` file. The code defines a RESTful resource for managing brands. It includes imports for `sn.ept.ventesvelos.rest` and `...
no usages`. The class `MarqueResource` has a single method `getMarqueList()`. This method uses `@Path("/marque")` to map to the '/marque' endpoint. It injects `MarqueFacade` via `@EJB`. The method uses `@GET` and `@Produces(MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML)` annotations. The implementation retrieves all brands from the facade and returns them as a JSON or XML response. A note in the code indicates a potential issue with loading.

```
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 Response("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     2 usages
28     @PUT
29     @Consumes(MediaType.APPLICATION_JSON)
30     @Produces(MediaType.APPLICATION_JSON)
31     public Response addMarque(@PathParam("id") Long id, Marque marque) {
32         if (id == null) {
33             return Response.status(Response.Status.BAD_REQUEST).entity("L'identifiant de la marque est obligatoire").build();
34         }
35         if (marque == null) {
36             return Response.status(Response.Status.BAD_REQUEST).entity("La marque à ajouter ne peut pas être nulle").build();
37         }
38     }
39
40     2 usages
41     @DELETE
42     @Consumes(MediaType.APPLICATION_JSON)
43     @Produces(MediaType.APPLICATION_JSON)
44     public Response deleteMarque(@PathParam("id") Long id) {
45         if (id == null) {
46             return Response.status(Response.Status.BAD_REQUEST).entity("L'identifiant de la marque à supprimer est obligatoire").build();
47         }
48         Marque marque = marqueFacade.find(id);
49         if (marque == null) {
50             return Response.status(Response.Status.NOT_FOUND).entity("La marque à supprimer n'existe pas").build();
51         }
52         marqueFacade.remove(marque);
53         return Response.status(Response.Status.OK).build();
54     }
55 }
```

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.

On procède de manière similaire en créant la méthode addProduit() avec l'annotation @PUT.

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.

On procède de manière similaire en créant la méthode deleteProduit() avec l'annotation @DELETE.

```
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5 no usages
6
7 @Path("/produit")
8 public class ProductResource {
9
10     2 usages
11     @EJB
12     private ProduitFacade produitFacade;
13
14     no usages
15     @GET
16     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17     public Response getProduitList() {
18         List<Produit> tmp = produitFacade.findAll();
19         if (tmp == null) {
20             return Response.status(Response.Status.NOT_FOUND).entity(new Reponse( msg: "Une erreur s'est produite lors de la récupération")).build();
21         }
22         return Response.status(Response.Status.OK)
23             .entity(tmp)
24             .build();
25     }
26
27 }
```

EmployeResource.java

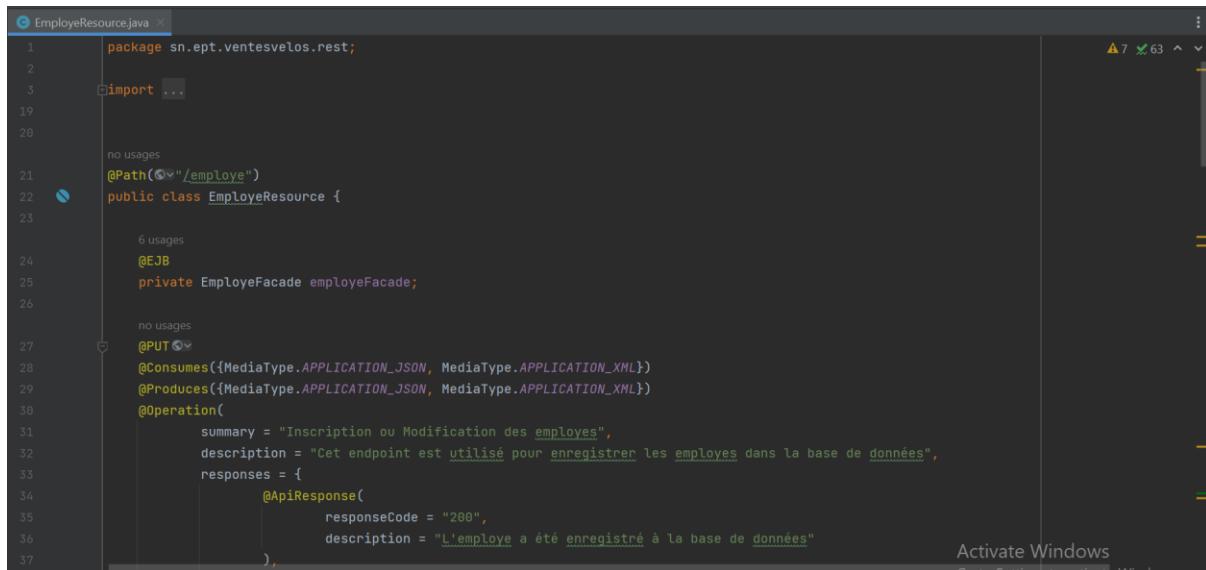
Nous utilisons l'annotation `@Path` (« employe ») utilisée en Jakarta RESTful Web Services (Jakarta RS) pour définir le chemin relatif d'une ressource ou d'une méthode d'une classe de ressource. On pourra donc accéder à cette en utilisant le chemin relatif « /api/employe ». On injecte donc la façade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

On procède de manière similaire en créant la méthode `getEmployeList()` avec l'annotation `@GET`.

Nous créons également une méthode `deleteEmploye()` dans laquelle nous supprimons un employé spécifique dont l'id est passé en paramètre à la requête qui sera envoyé comme réponse est vendeur. On spécifie l'annotation `@DELETE` 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("/employe")
8 public class EmployeResource {
9
10     6 usages
11     @EJB
12     private EmployeFacade employeFacade;
13
14     no usages
15     @PUT
16     @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
18     @Operation(
19         summary = "Inscription ou Modification des employés",
20         description = "Cet endpoint est utilisé pour enregistrer les employés dans la base de données",
21         responses = {
22             @ApiResponse(
23                 responseCode = "200",
24                 description = "L'employé a été enregistré à la base de données"
25             ),
26         }
27     )
28
29     7 usages
30     ...
31
32     ...
33
34     ...
35
36     ...
37 }
```

EmployeResource.java

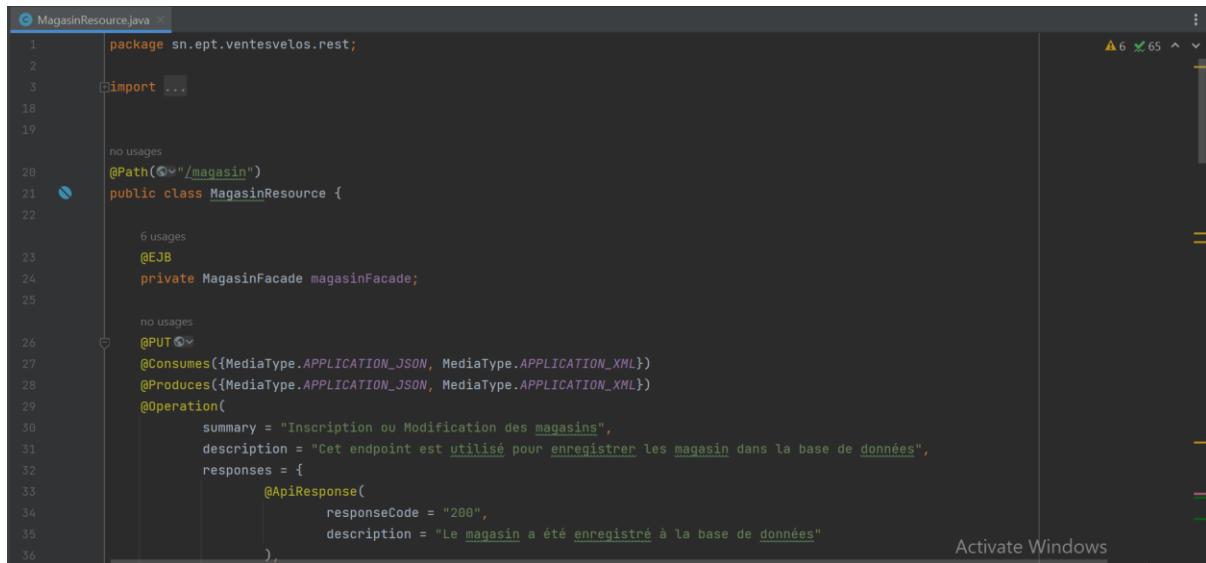
Nous utilisons l'annotation `@Path` (« magasin ») utilisée en Jakarta RESTful Web Services (Jakarta RS) pour définir le chemin relatif d'une ressource ou d'une méthode d'une classe de ressource. On pourra donc accéder à cette en utilisant le chemin relatif « /api/magasin ». On injecte donc la façade 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 `addMagasin()` dans laquelle nous créons ou modifions le magasin 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 faîte aux formats JSON et XML.

On procède de manière similaire en créant la méthode `getMagasinList()` avec l'annotation `@GET`.

Nous créons également une méthode `deleteMagsin()` dans laquelle nous supprimons un magasin spécifique dont l'id est passé en paramètre à la requête qui sera envoyé comme répons. On spécifie l'annotation `@DELETE` 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 ...
18
19 no usages
20 @Path("/magasin")
21 public class MagasinResource {
22
23     6 usages
24     @EJB
25     private MagasinFacade magasinFacade;
26
27     no usages
28     @PUT
29     @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
30     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
31     @Operation(
32         summary = "Inscription ou Modification des magasins",
33         description = "Cet endpoint est utilisé pour enregistrer les magasin dans la base de données",
34         responses = {
35             @ApiResponse(
36                 responseCode = "200",
37                 description = "Le magasin a été enregistré à la base de données"
38         ),
39     }
40 )
41 }
```

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`.

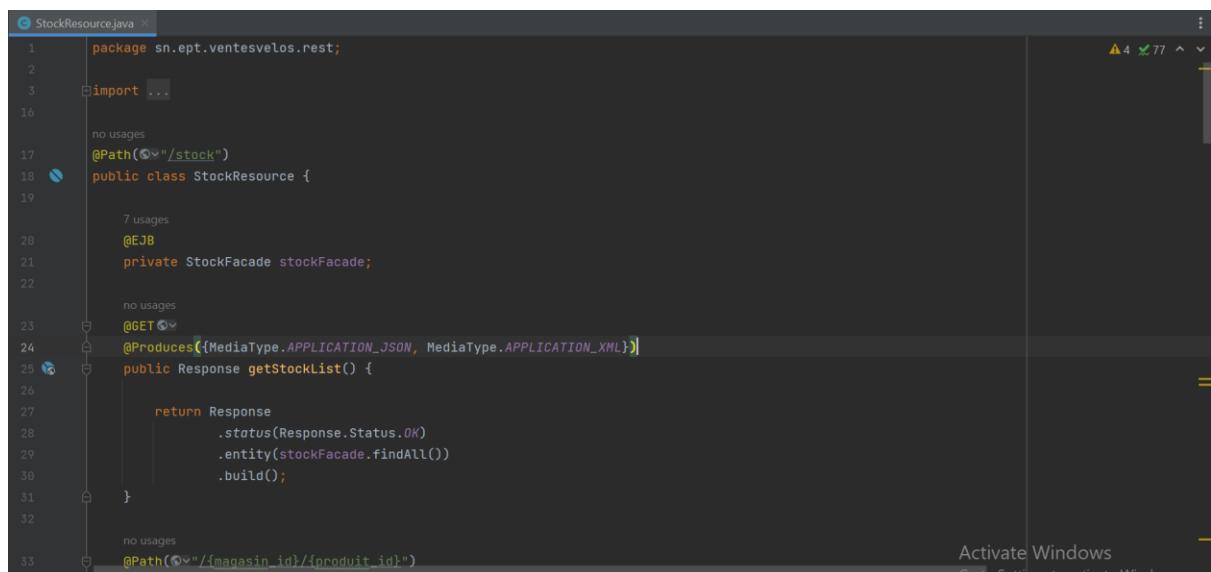
Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec `@GET`, `@PUT`, `@DELETE` suivant le type de requête devant permettre d'y accéder. En utilisant `@Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})` ou `@Produces ({MediaType.APPLICATION_JSON, 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.

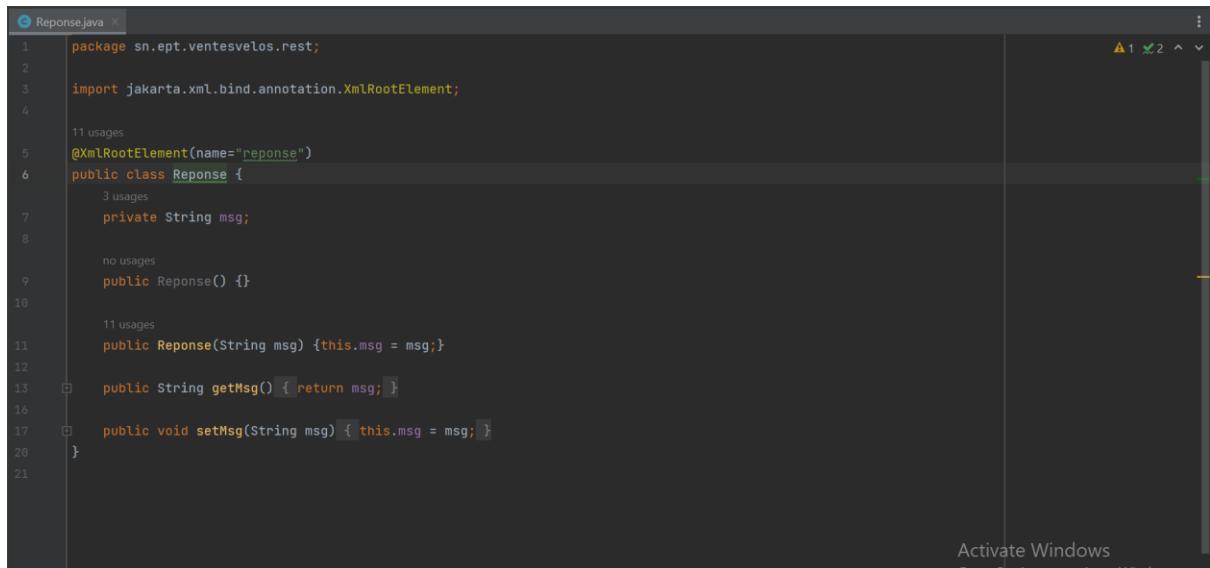


```
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5 no usages
6
7 @Path("/stock")
8 public class StockResource {
9
10     7 usages
11     @EJB
12     private StockFacade stockFacade;
13
14     no usages
15     @GET
16     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17     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 @XmlRootElement 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 /**
6  * 11 usages
7  * @XmlRootElement(name="reponse")
8  */
9 public class Reponse {
10     /**
11      * 3 usages
12      */
13     private String msg;
14
15     /**
16      * no usages
17      */
18     public Reponse() {}
19
20     /**
21      * 11 usages
22      */
23     public Reponse(String msg) {this.msg = msg;}
24
25     /**
26      * 11 usages
27      */
28     public String getMsg() { return msg; }
29
30     /**
31      * 11 usages
32      */
33     public void setMsg(String msg) { this.msg = msg; }
34 }
```

Activate Windows
Go to Settings to activate Windows

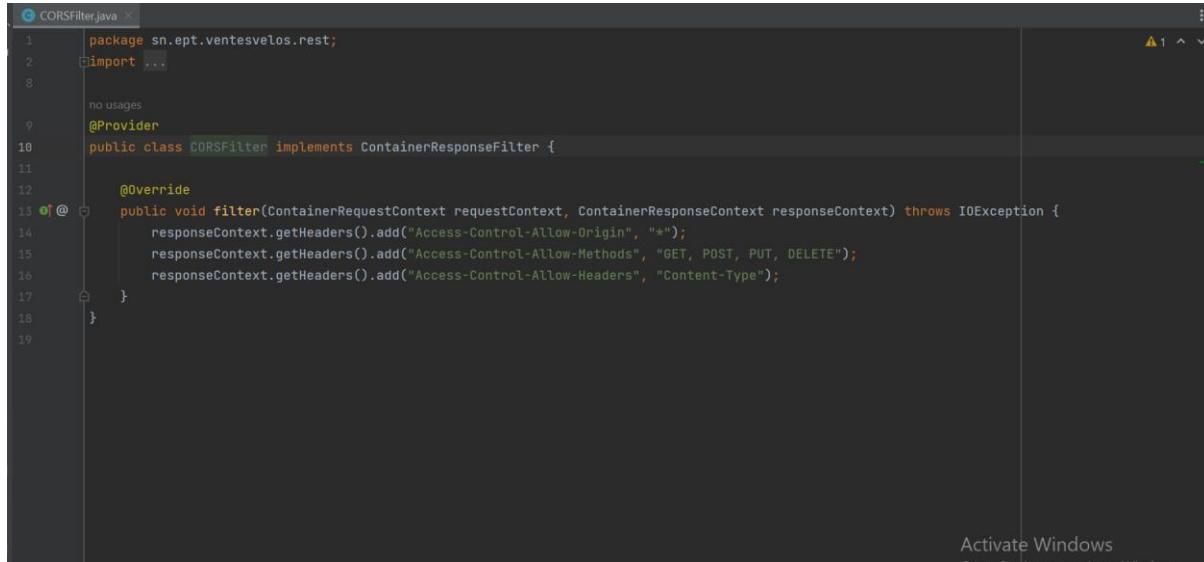
CORSFilter.java

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.

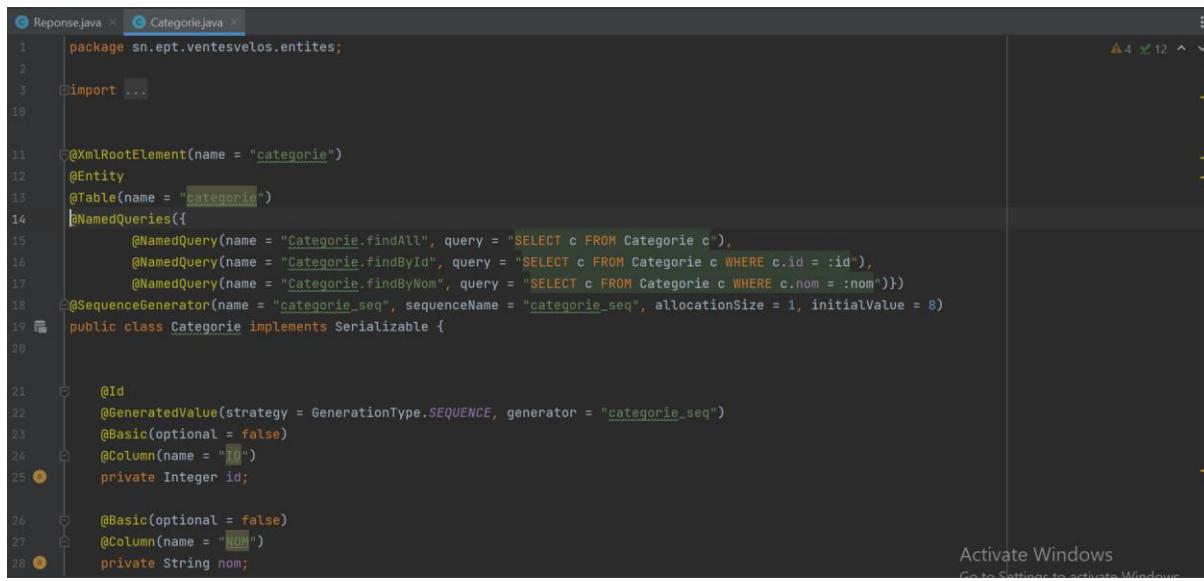
L'en-tête "Access-Control-Allow-Methods" spécifie les méthodes HTTP autorisées pour les requêtes cross-origin.

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 }
```

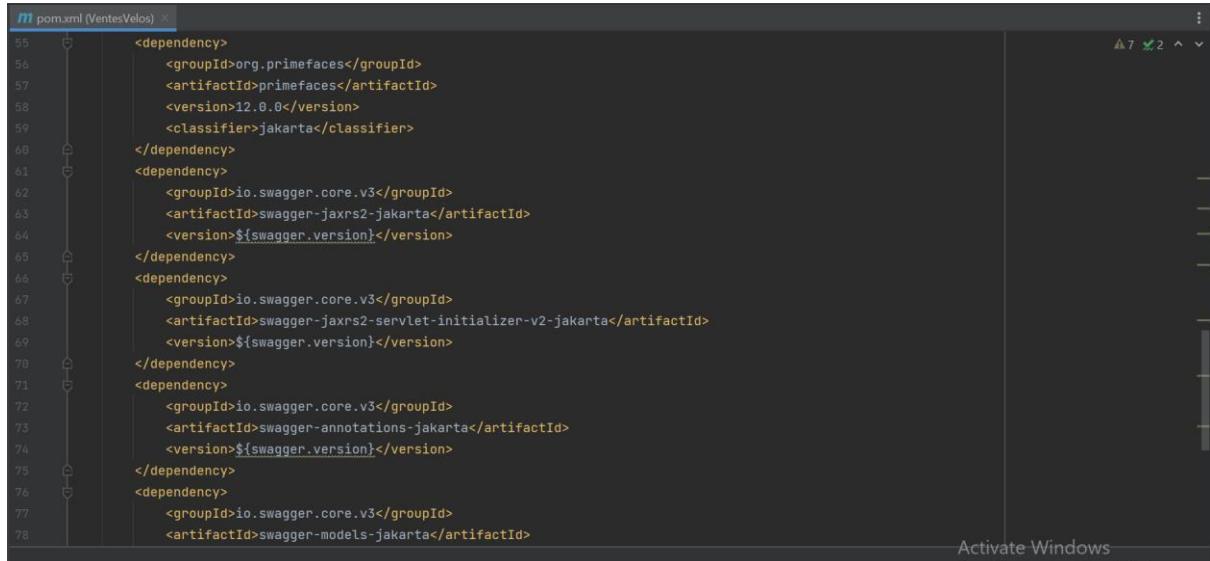
Pour les besoins de la communication au format XML on ajoute également l'annotation @XmlRootElement à nos entités. Comme on peut le voir dans cet exemple avec Catégorie.



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

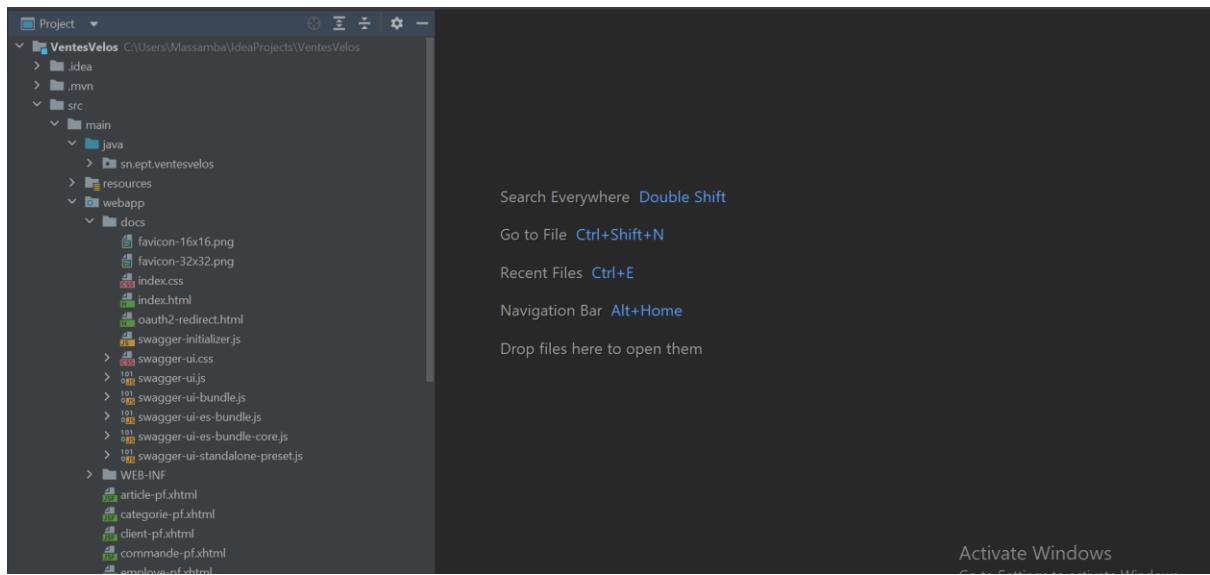
2. Documentation des services web

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

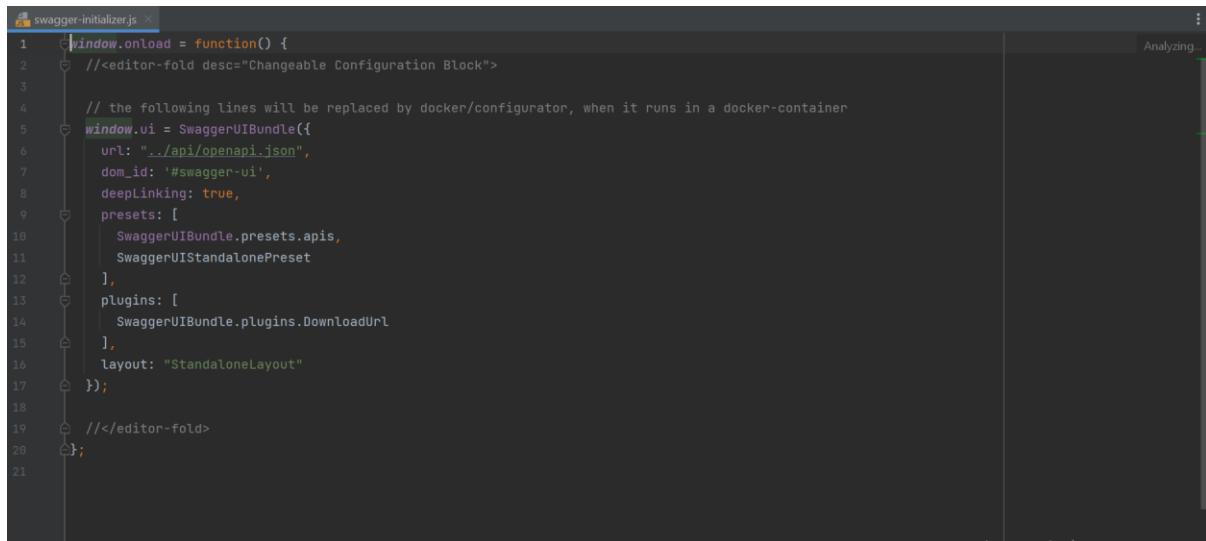


```
55 <dependency>
56     <groupId>org.primefaces</groupId>
57     <artifactId>primefaces</artifactId>
58     <version>12.0.0</version>
59     <classifier>jakarta</classifier>
60 </dependency>
61 <dependency>
62     <groupId>io.swagger.core.v3</groupId>
63     <artifactId>swagger-jaxrs2-jakarta</artifactId>
64     <version>${swagger.version}</version>
65 </dependency>
66 <dependency>
67     <groupId>io.swagger.core.v3</groupId>
68     <artifactId>swagger-servlet-initializer-v2-jakarta</artifactId>
69     <version>${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 et on copie le dossier dist dans webapp dans un dossier docs.



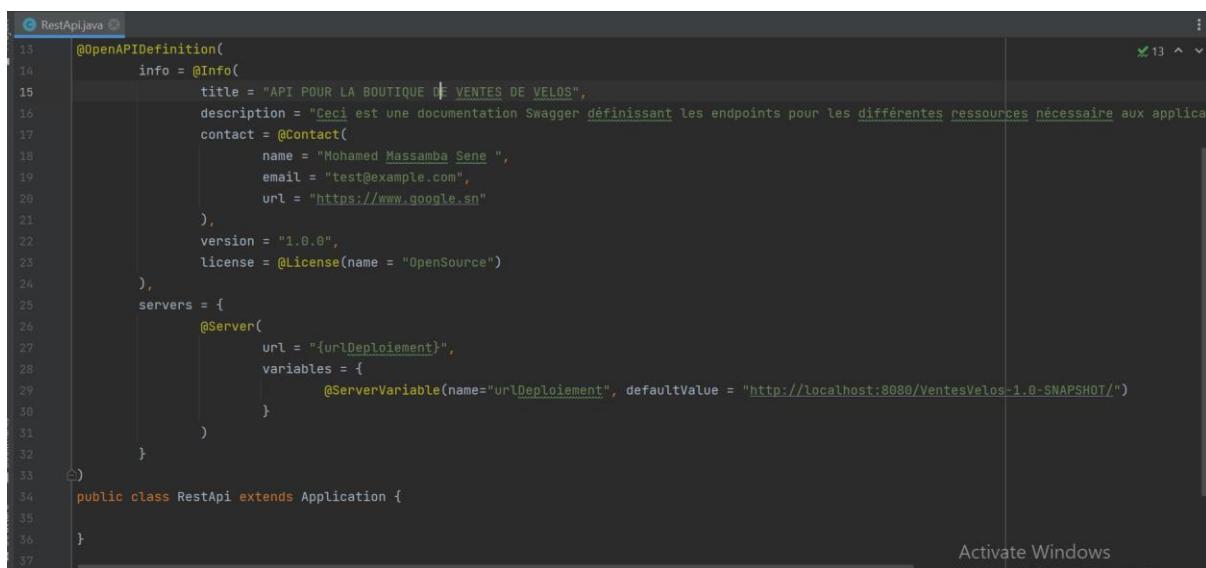
Nous ouvrons ensuite le fichier *swagger-initializer.js* dans lequel nous placer l'url de la documentation



```
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.

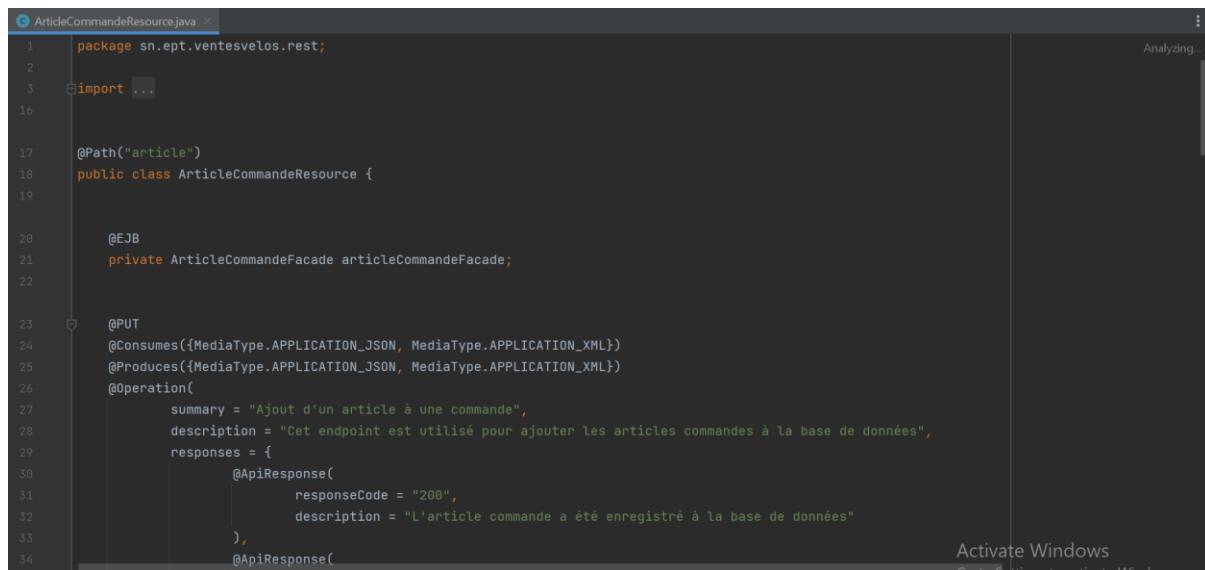


```
13 @OpenAPIDefinition(
14     info = @Info(
15         title = "API POUR LA BOUTIQUE DE VENTES DE VELOS",
16         description = "Ceci est une documentation Swagger définissant les endpoints pour les différentes ressources nécessaire aux applicat",
17         contact = @Contact(
18             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 }
```

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 :



```
ArticleCommandeResource.java
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5
6 @Path("article")
7 public class ArticleCommandeResource {
8
9
10    @EJB
11    private ArticleCommandeFacade articleCommandeFacade;
12
13
14    @PUT
15    @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
16    @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17    @Operation(
18        summary = "Ajout d'un article à une commande",
19        description = "Cet endpoint est utilisé pour ajouter les articles commandés à la base de données",
20        responses = {
21            @ApiResponse(
22                responseCode = "200",
23                description = "L'article commandé a été enregistré à la base de données"
24            ),
25            @ApiResponse(
26                responseCode = "400",
27                description = "Le produit n'existe pas dans la base de données"
28            )
29        }
30    )
31    void addArticle(@PathParam("idCommande") Long idCommande, ArticleCommande articleCommande);
32
33
34 }
```

```
ArticleCommandeResource.java
34     @ApiResponse(
35         responseCode = "500",
36         description = "Une erreur s'est produite lors de l'enregistrement de l'article commande"
37     )
38 }
39 )
40 @Public
41 public Response addArticleCommande(
42     @Parameter(
43         name = "Article Commande",
44         description = "L'article commande que vous souhaitez ajouter",
45         required = true
46     )
47     ArticleCommande a
48 ) {
49     ArticleCommande tmp = (ArticleCommande) articleCommandeFacade.find(a.getArticleCommandePK());
50     if (tmp != null) {
51         articleCommandeFacade.edit(a);
52     } else {
53         articleCommandeFacade.create(a);
54     }
55     return Response
56         .status(Response.Status.OK)
57         .entity(a)
58         .build();
59 }
```

Activate Windows

```
ArticleCommandeResource.java
no usages
61     @Path("/{numero}/{ligne}")
62     @DELETE
63     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
64     @Operation(
65         summary = "Suppression d'article commande",
66         description = "Cet endpoint est utilisé pour supprimer les articles commandes de la base de données",
67         responses = {
68             @ApiResponse(
69                 responseCode = "200",
70                 description = "L'article commande a été supprimé de la base de données"
71             ),
72             @ApiResponse(
73                 responseCode = "500",
74                 description = "Une erreur s'est produite lors de la suppression de l'article commande"
75             ),
76             @ApiResponse(
77                 responseCode = "404",
78                 description = "Article commande correspondant à la clé indiquée est introuvable",
79                 content = {
80                     @Content(
81                         mediaType = MediaType.APPLICATION_JSON,
82                         examples = {
83                             @ExampleObject(name="Article commande introuvable")
84                         }
85                     )
86                 }
87             )
88         }
89     )
90     public Response deleteArticleCommande(
91         @PathParam("numero")
92         @Parameter(description = "Le numero de commande", required = true)
93         int numero,
94         @PathParam("ligne")
95         @Parameter(description = "La ligne de l'article commande", required = true)
96         int ligne
97     ) {
98     ArticleCommande tmp = (ArticleCommande) articleCommandeFacade.find(new ArticleCommandePK(numero, ligne));
99     if (tmp == null) {
100         return Response.status(Response.Status.NOT_FOUND).entity(new Reponse(msg: "L'article commande de clé (" + numero + ", " + ligne + ") n'a pas été trouvé"));
101     }
102     articleCommandeFacade.remove(tmp);
103 }
```

Activate Windows

```
ArticleCommandeResource.java
78     description = "Article commande correspondant à la clé indiquée est introuvable",
79     content = {
80         @Content(
81             mediaType = MediaType.APPLICATION_JSON,
82             examples = {
83                 @ExampleObject(name="Article commande introuvable")
84             }
85         )
86     }
87 }
88 )
89 )
90     public Response deleteArticleCommande(
91         @PathParam("numero")
92         @Parameter(description = "Le numero de commande", required = true)
93         int numero,
94         @PathParam("ligne")
95         @Parameter(description = "La ligne de l'article commande", required = true)
96         int ligne
97     ) {
98     ArticleCommande tmp = (ArticleCommande) articleCommandeFacade.find(new ArticleCommandePK(numero, ligne));
99     if (tmp == null) {
100         return Response.status(Response.Status.NOT_FOUND).entity(new Reponse(msg: "L'article commande de clé (" + numero + ", " + ligne + ") n'a pas été trouvé"));
101     }
102     articleCommandeFacade.remove(tmp);
103 }
```

Activate Windows

```
ArticleCommandeResource.java
182     articleCommandeFacade.remove(tmp);
183     return Response.status(Response.Status.OK).entity(new Reponse( msg: "L'article commande de clé ("+ numero +", "+ ligne +") a été supprimé"))
184 }
185
no usages
186 @GET
187 @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
188 @Operation(
189     summary = "Récupération des articles commandes",
190     description = "Cet endpoint est utilisé pour récupérer tous les articles commande de la base de données",
191     responses = {
192         @ApiResponse(
193             responseCode = "200",
194             description = "Les articles commandes ont été supprimées à la base de données"
195         ),
196         @ApiResponse(
197             responseCode = "404",
198             description = "Une erreur s'est produite lors de la récupération de l'article commande"
199     })
200 )
201
202 public Response getArticlesCommandes() {
203     List<ArticleCommande> categories = articleCommandeFacade.findAll();
204     if (categories == null) {
205         return Response.status(Response.Status.NOT_FOUND).entity(new Reponse( msg: "Un problème s'est produit lors de la récupération")).build();
206     }
207
208     return Response
209         .status(Response.Status.OK)
210         .entity(categories)
211         .build();
212 }
```

```
ArticleCommandeResource.java
110
111     ),
112     @ApiResponse(
113         responseCode = "404",
114         description = "Une erreur s'est produite lors de la récupération de l'article commande"
115     )
116 }
117
118 public Response getArticlesCommandes() {
119     List<ArticleCommande> categories = articleCommandeFacade.findAll();
120     if (categories == null) {
121         return Response.status(Response.Status.NOT_FOUND).entity(new Reponse( msg: "Un problème s'est produit lors de la récupération")).build();
122     }
123
124     return Response
125         .status(Response.Status.OK)
126         .entity(categories)
127         .build();
128 }
129
130 }
```

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

The screenshot shows the main page of the Swagger UI interface for the VentesVelos API. At the top, there's a navigation bar with tabs for 'Servers' (set to '{urlDeploiement}'), 'Explore', and a sidebar with various icons. Below the header, the title 'API POUR LA BOUTIQUE DE VENTES DE VELOS' is displayed, along with version '1.0.0' and 'OAS3'. A brief description follows: 'Ceci est une documentation Swagger définissant les endpoints pour les différentes ressources nécessaire aux application Web et Mobile pour la vente de vélos'. It also includes author information ('Mohamed Massamba Sene - Website', 'Send email to Mohamed Massamba Sene', 'OpenSource') and a computed URL ('http://localhost:8080/VentesVelos-1.0-SNAPSHOT/').

Servers
{urlDeploiement}

Computed URL: http://localhost:8080/VentesVelos-1.0-SNAPSHOT/

Server variables

urlDeploiement http://localhost:8080/Ventes

Activate Windows
Go to Settings to activate Windows

The second part of the screenshot shows the detailed list of endpoints for the 'default' server. The list is organized by resource type (articles, categories, clients) and includes methods like GET, POST, PUT, DELETE, and PATCH. Each endpoint is accompanied by a brief description.

- articles:**
 - GET /api/article Récupération des articles commandes
 - PUT /api/article Ajout d'un article à une commande
 - DELETE /api/article/{numero}/{ligne} Suppression d'article commande
- categories:**
 - GET /api/categorie Liste des catégories
 - PUT /api/categorie Ajout d'une catégorie
 - GET /api/categorie/{id} Trouver les produits
 - DELETE /api/categorie/{id} Suppression de catégorie
- clients:**
 - GET /api/client Liste des clients
 - PUT /api/client Inscription ou Modification des clients
 - GET /api/client/{id} Trouver les commandes

Activate Windows
Go to Settings to activate Windows

Swagger UI

localhost:8080/VentesVelos-1.0-SNAPSHOT/docs/index.html

Activate Windows
Go to Settings to activate Windows

Method	Path	Description
GET	/api/client/{id}	Trouver les commandes
DELETE	/api/client/{id}	Suppression de client
GET	/api/commande	Liste des clients
PUT	/api/commande	Ajout d'une commande
GET	/api/commande/{numero}	Trouver les articles d'une commande
DELETE	/api/commande/{numero}	Suppression de commande
GET	/api/employe	Liste des employés
PUT	/api/employe	Inscription ou Modification des employés
DELETE	/api/employe/{id}	Suppression de l'employé
GET	/api/magasin	Liste des magasins
PUT	/api/magasin	Inscription ou Modification des magasins

Swagger UI

localhost:8080/VentesVelos-1.0-SNAPSHOT/docs/index.html

Activate Windows
Go to Settings to activate Windows

Method	Path	Description
DELETE	/api/magasin/{id}	Suppression du magasin
GET	/api/marque	Liste des marques
PUT	/api/marque	Ajout d'une marque
GET	/api/marque/{id}	Trouver les produits
DELETE	/api/marque/{id}	Suppression de marque
GET	/api/produit	Liste des produits
PUT	/api/produit	Ajout d'un produit
GET	/api/produit/{id}	Trouver un produit
DELETE	/api/produit/{id}	Suppression de produit
GET	/api/stock	Liste des stocks
PUT	/api/stock	Ajout d'un stock

The screenshot shows the Swagger UI interface for a REST API. The top navigation bar displays the title "Swagger UI" and the URL "localhost:8080/VentesVelos-1.0-SNAPSHOT/docs/index.html". The main content area lists several API endpoints:

- DELETE /api/produit/{id}** Suppression de produit
- 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

Below the endpoints, there is a section titled "Schemas" containing definitions for:

- Adresse
- ArticleCommande
- ArticleCommandePK

A watermark for "Activate Windows" is visible in the bottom right corner.

The screenshot shows the Swagger UI interface for the "/api/categorie" endpoint. The top navigation bar displays the title "Swagger UI" and the URL "localhost:8080/VentesVelos-1.0-SNAPSHOT/docs/index.html#/default/getProduitList". A "Sign in" button is highlighted with a red circle.

The endpoint details are as follows:

- GET /api/categorie** Liste des catégories

Description: 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.

Below the endpoint details, another entry for the "/api/categorie/{id}" endpoint is partially visible.

Swagger UI

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

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 <small>* required</small>	L'identifiant de la catégorie <small>(path)</small>

Cancel

Execute

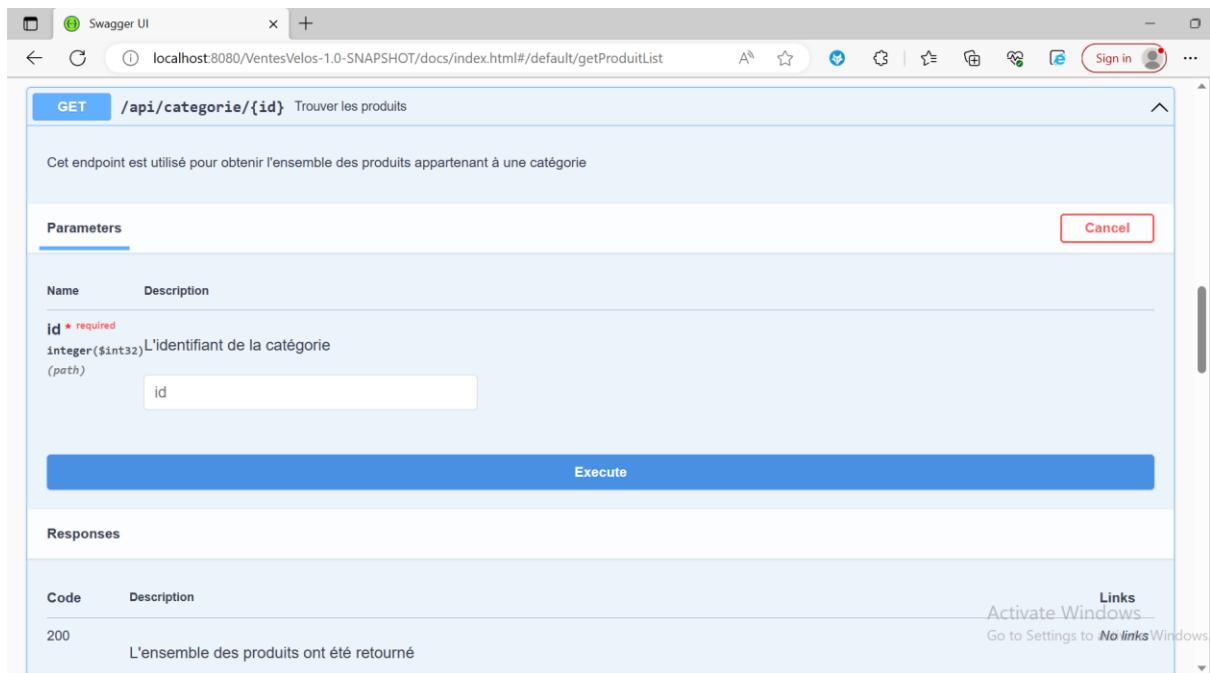
Responses

Code	Description
200	L'ensemble des produits ont été retourné

Links

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.

```

package sn.ept;

import ...;

no usages
public class ArticleTest {

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

    no usages
    @Test

```

Run: ArticleTest x

- ArticleTest (sn.ept) 9 sec 383 ms
 - testDeleteArticleJSON 1 sec 770 ms
 - testGetArticleList() 5 sec 706 ms
 - testDeleteArticleNotFound 190 ms
 - testAddArticleJSON() 1 sec 303 ms
 - testDeleteArticleNotFound() 40 ms
 - testDeleteArticleXML() 32 ms
 - testAddArticleXML() 342 ms

Process finished with exit code 0

Activate Windows
Go to Settings to activate Windows.

123:36 CRLF UTF-8 4 spaces

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.

```

package sn.ept;

import ...;

no usages
public class CategorieTest {

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

    no usages

```

Run: CategorieTest x

- CategorieTest (sn.ept) 2 sec 808 ms
 - testDeleteCategorieS 1 sec 551 ms
 - testAddCategorieXML() 855 ms
 - testGetProductList(JSON) 204 ms
 - testAddCategorieJSON() 75 ms
 - testDeleteCategorieNotFound 28 ms
 - testDeleteCategorieNotFound() 26 ms
 - testGetProductListNotFound() 21 ms
 - testDeleteCategorieXML() 22 ms
 - testGetCategorieList() 26 ms

Process finished with exit code 0

Activate Windows
Go to Settings to activate Windows.

28:35 CRLF UTF-8 4 spaces

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.

The screenshot shows the IntelliJ IDEA interface. The top navigation bar includes File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help, and ResourceTest - ClientTest.java. The Project tool window on the left shows the file ClientTest.java. The code defines a package sn.ept and a class ClientTest with a static setup method that sets the base URI to "http://localhost:8080/VentesVélos-1.0-SNAPSHOT/api". The Run tool window at the bottom shows the results of a run named ClientTest, indicating 10 tests passed in 3 seconds and 463 ms. The terminal output shows "Process finished with exit code 0". The status bar at the bottom right shows "Activate Windows Go to Settings to activate Windows.", the time "19:22", and encoding "CRLF UTF-8 4 spaces".

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.

```

package sn.ept;

import ...;

no usages
public class CommandeTest {

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

    no usages
}

```

Run: CommandeTest x

- CommandeTest (sn.ept) 7 sec 530 ms
 - testDeleteCommande 2 sec 271 ms
 - testDeleteCommandeJSON 121 ms
 - testGetCommandeList 3 sec 891 ms
 - testAddCommandeJSON 1 sec 15 ms
 - testGetArticleCommandeList 32 ms
 - testDeleteCommandeXML 82 ms
 - testGetArticleCommandeList 28 ms
 - testGetArticleCommandeList 63 ms
 - testDeleteCommandeNotFound 27 ms

Process finished with exit code 0

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.

```

package sn.ept;

import ...;

no usages
public class MarqueTest {

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

    no usages
}

```

Run: MarqueTest x

- MarqueTest (sn.ept) 4 sec 113 ms
 - testGetProduitListSOI 2 sec 867 ms
 - testGetMarqueListJSON() 38 ms
 - testDeleteMarqueNotFound 63 ms
 - testDeleteMarqueNotFound 50 ms
 - testDeleteMarqueXML() 61 ms
 - testDeleteMarqueJSON() 42 ms
 - testAddMarqueJSON() 964 ms
 - testGetProduitListNotFound 28 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:

- Code Editor:** The file `ProductTest.java` is open, showing Java code using RestAssured for API testing. It includes a `@BeforeAll` setup method and various `@Test` methods for product operations.
- Run Tool Window:** The `Run` tool window is active, showing the execution of the `ProductTest` class. It displays 8 tests passed in 5 seconds, with detailed timing for each test.
- Status Bar:** The status bar at the bottom right indicates "Tests passed: 8 (moments ago)".

EmployeTest.java

Dans cette classe nous allons tester la ressource `EmployeResource` 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.

```
ResourceTest > src > main > java > sn > ept > EmployeTest > testDeleteEmployee
```

```
EmployeTest.java
```

```
1 package sn.ept;
2
3 import ...
4
5 no usages
6 public class EmployeTest {
7
8     no usages
9     @BeforeAll
10    public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVelos-1.0-SNAPSHOT/api"; }
11
12    no usages
13}
14
15
16
17
18
```

Run: EmployeTest x

EmployeTest (sn.ept) 4 sec 505 ms "C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" ...
Process finished with exit code 0

Tests passed: 4 (moments ago)

MagasinTest.java

Dans cette classe nous allons tester la ressource MagasinResource 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.

```
ResourceTest > src > main > java > sn > ept > MagasinTest
```

```
MagasinTest.java
```

```
1 package sn.ept;
2
3 import ...
4
5 no usages
6 public class MagasinTest {
7
8     no usages
9     @BeforeAll
10    public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVelos-1.0-SNAPSHOT/api"; }
11
12    no usages
13}
14
15
16
17
18
```

Run: MagasinTest x

MagasinTest (sn.ept) 5 sec 776 ms "C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" ...
Process finished with exit code 0

Tests passed: 4 (moments ago)

StockTest.java

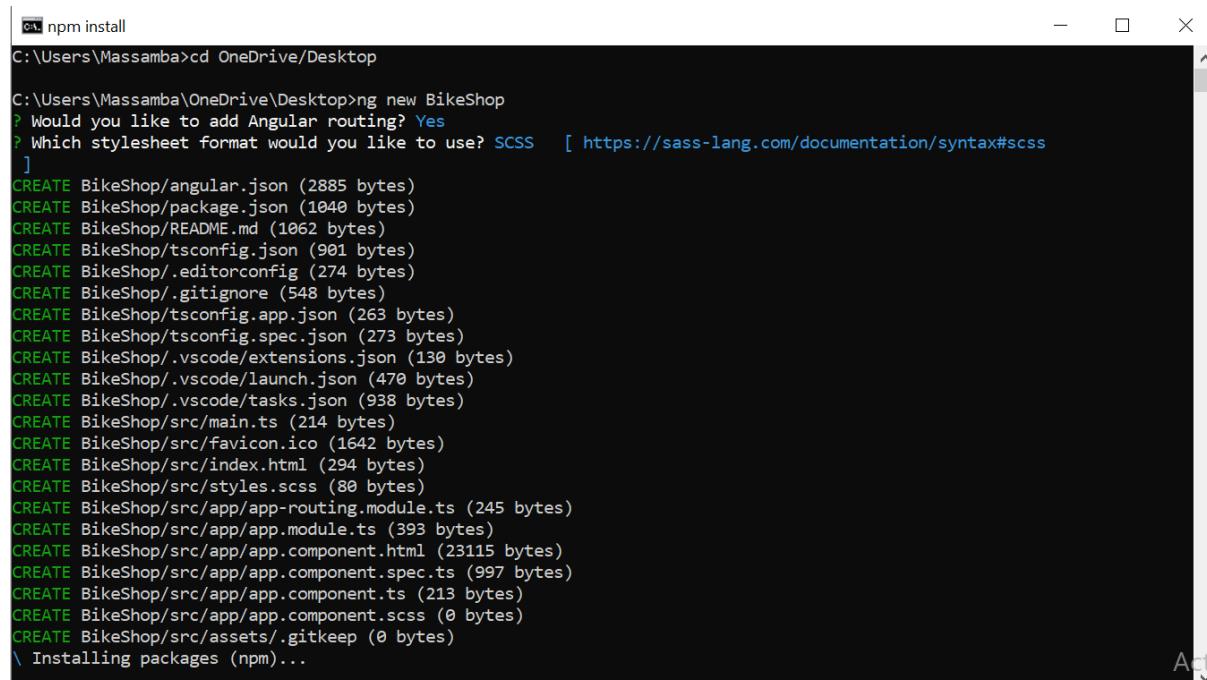
Dans cette classe nous allons tester la ressource StockResource en effectuant des tests sur l'ensemble des actions possibles définies au niveau de l'endpoint.

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 a Java IDE interface with two main panes. The top pane displays the code for `StockTest.java`. It includes a `setup()` method setting the base URI to `"http://localhost:8080/VentesVelos-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'. It lists 8 tests under 'StockTest (suite)' with execution times: `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). All tests passed in 5 seconds and 128 milliseconds. The command used was `"C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" ...`.

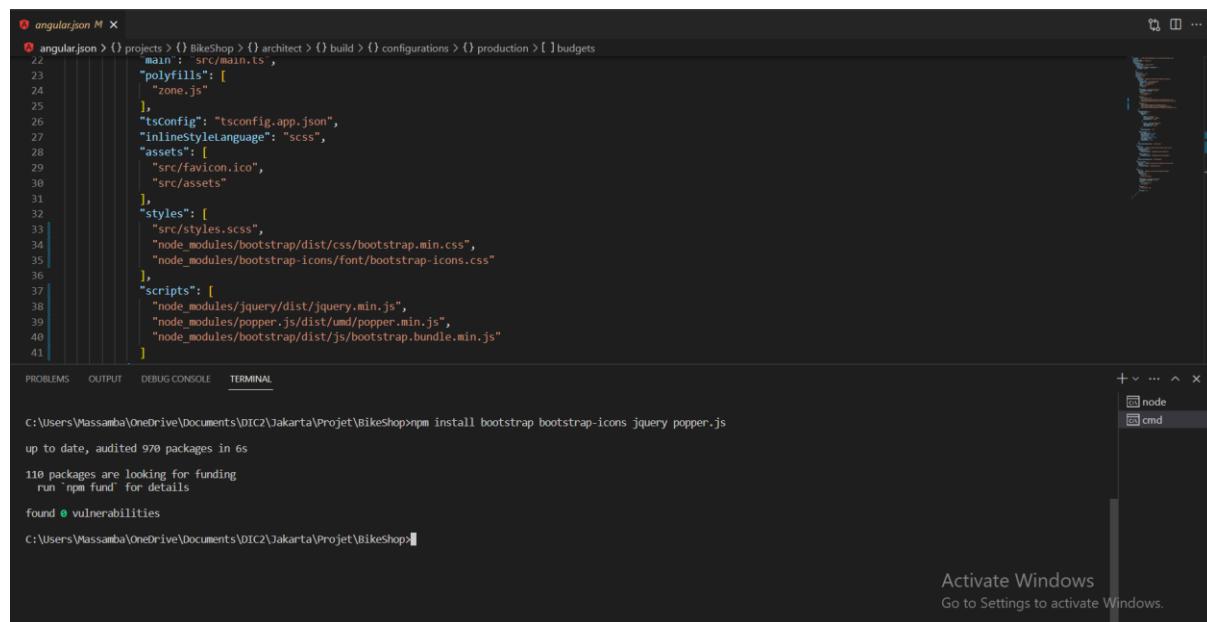
IV. Client Angular+bootstrap

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 (1062 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
...
22   "main": "src/main.ts",
23   "polyfills": [
24     "zone.js"
25   ],
26   "tsConfig": "tsconfig.app.json",
27   "inlineStyleLanguage": "scss",
28   "assets": [
29     "src/favicon.ico",
30     "src/assets"
31   ],
32   "styles": [
33     "src/styles.scss",
34     "node_modules/bootstrap/dist/css/bootstrap.min.css",
35     "node_modules/bootstrap-icons/font/bootstrap-icons.css"
36   ],
37   "scripts": [
38     "node_modules/jquery/dist/jquery.min.js",
39     "node_modules/popper.js/dist/umd/popper.min.js",
40     "node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"
41   ]
}

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
```

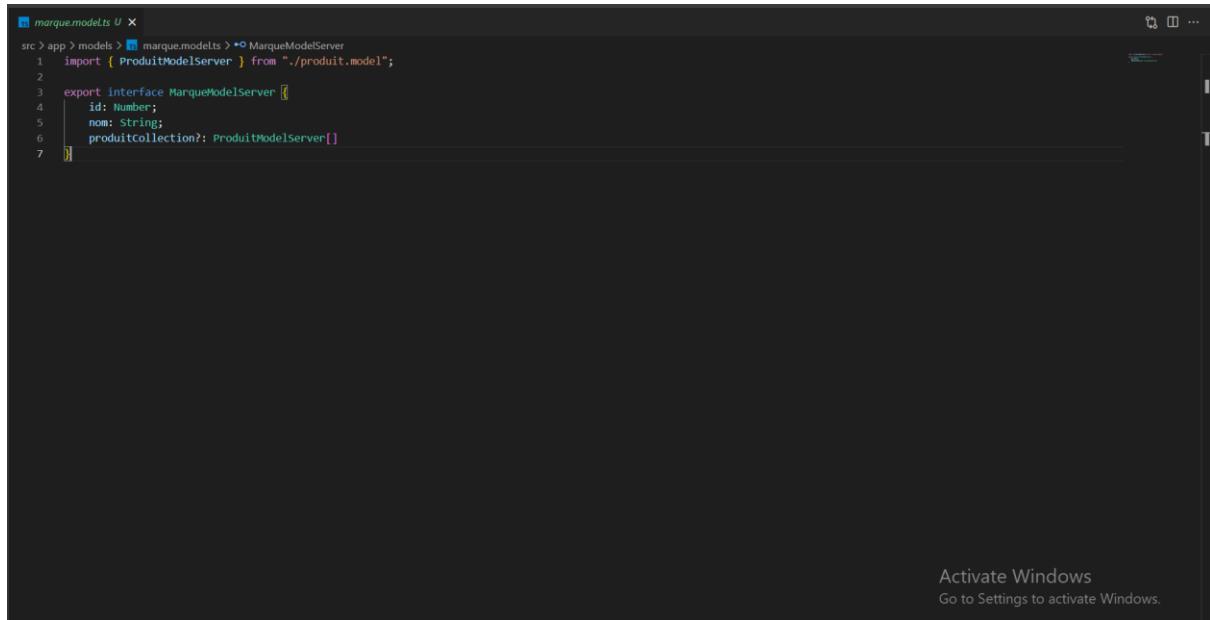
```
src > app > models > cart.models > ...
1 import { ProduitModelServer } from "./produit.model";
2 import { StockModelServer } from "./stock.model";
3
4 export interface CartModelServer {
5   total: number;
6   itemsInCart: number;
7   data: CartProduitServer[]
8 }
9
10 export interface CartProduitServer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

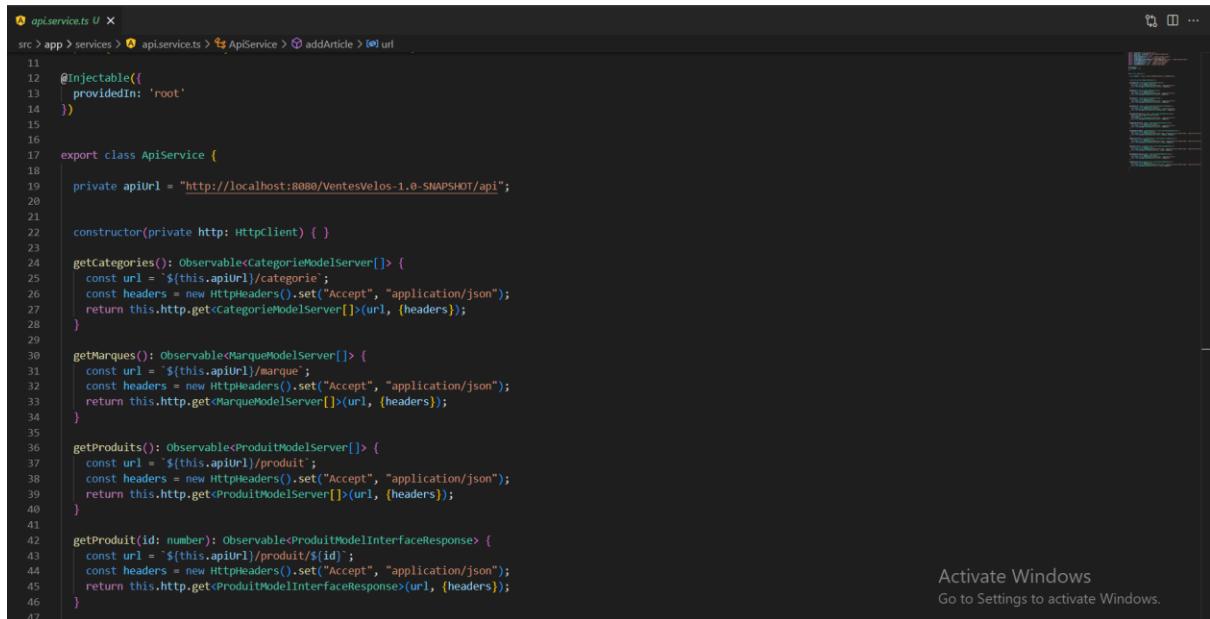
Activate Windows
Go to Settings to activate Windows.



```
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                     this.cartItems.data[index].numInCart -= quantite;
44                 }
45             }
46         }
47     }
48 }
```

Activate Windows
Go to Settings to activate Windows.

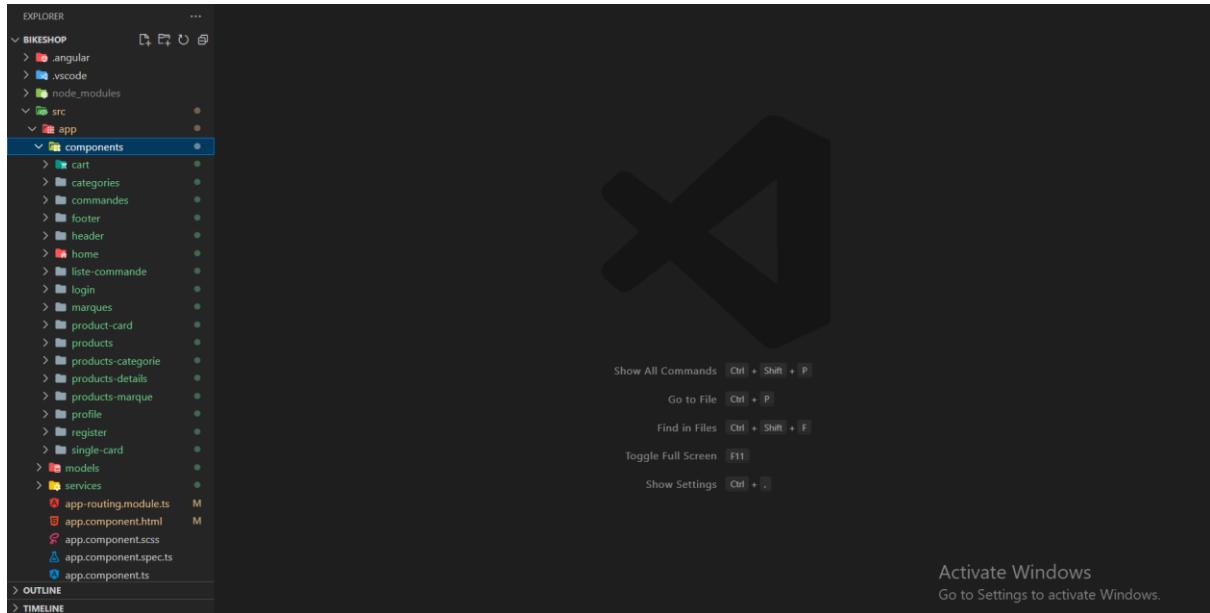
```

10  export class UserService {
11
12    private STORAGE_KEY = "users";
13    private COMMANDE_STORAGE = "num_commande";
14    private CLIENT_STORAGE = "num_client";
15    private users = this.getUsers();
16
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    }

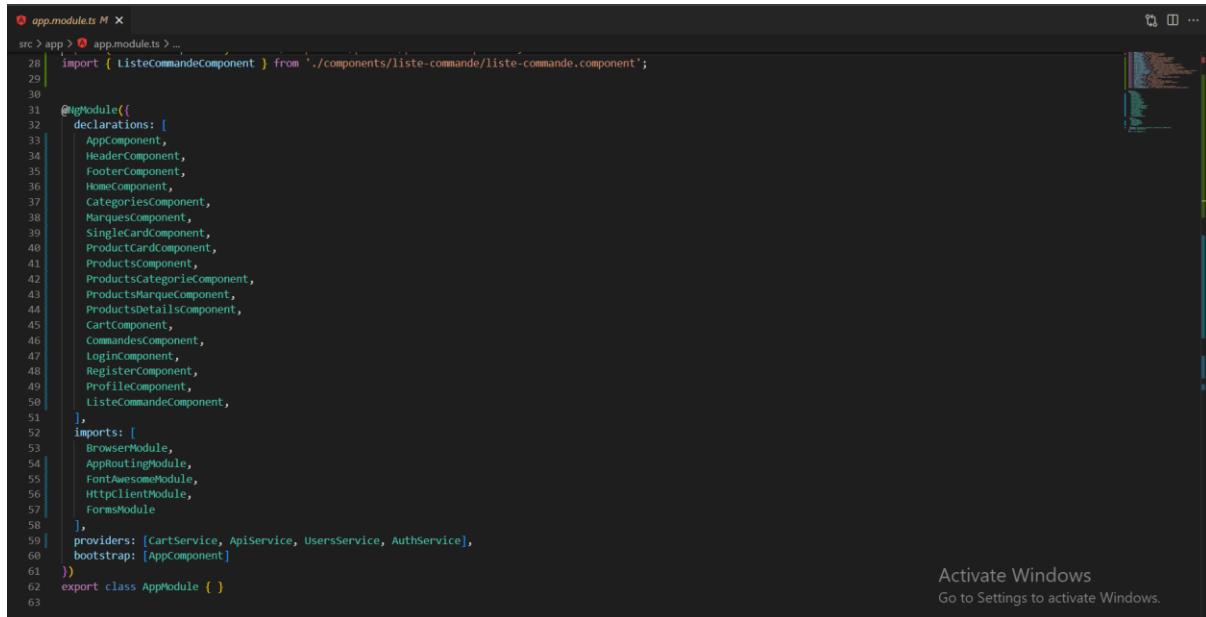
```

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

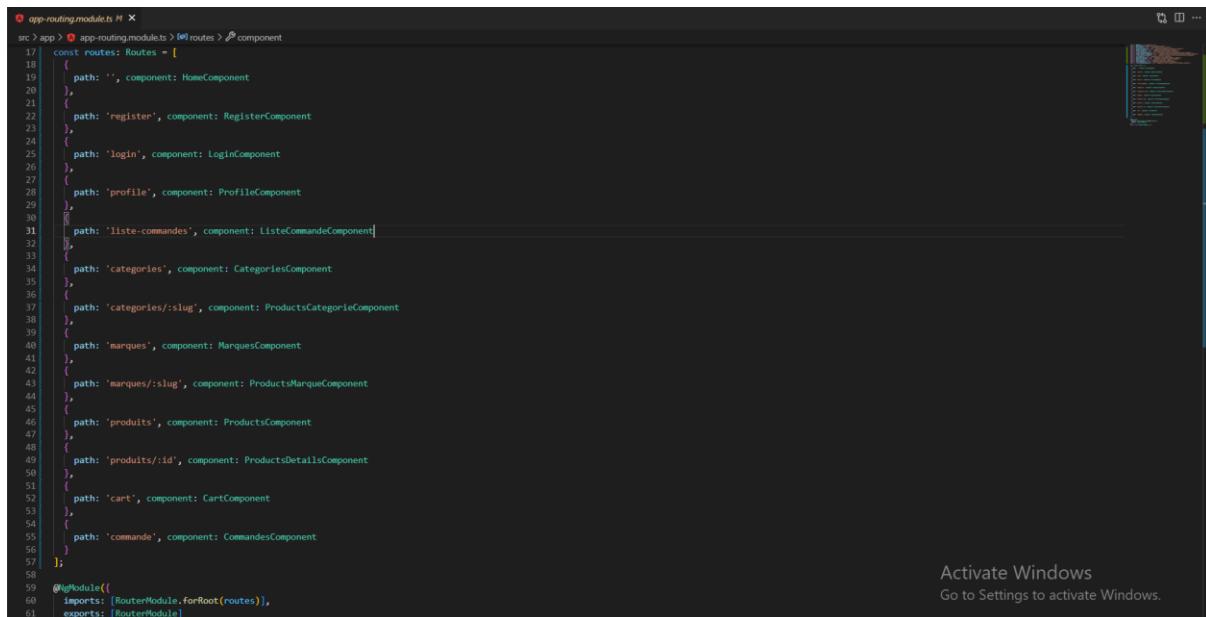


Nous importons les modules HttpClient, FormsModule, FontAwesomeModule, AppRoutingModules et nous ajoutons nos services au en tant que *providers*



```
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 { }
```

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 export class AppRoutingModule { }
```

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

- Page Accueil

BikeShop

localhost:4200

Accueil Tout Categories Marques Connexion Sign in

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 Google Chrome A propos de nous Mon compte

Activate Windows
Go to Settings to activate Windows

➤ 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



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



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 BikeShop logo, a navigation menu with links to '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 for 'Email' and 'Mot de passe', followed by a red-bordered 'Connexion' button. Below the buttons is the text 'Pas encore de compte? [Inscrivez-vous](#)'. At the bottom of the page, there are four main sections: 'A PROPOS DE NOUS', 'SUPPORT', 'INFORMATIONS', and 'SERVICES'. Each section lists various links such as 'FAQ', 'Knowledge base', 'Garantie', 'Nouveauté', 'A propos de nous', 'Contactez nous', 'Politique de confidentialité', 'Termes & Conditions', 'Mon compte', 'Voir panier', 'Aide', and 'Activate Windows'.

This screenshot is identical to the one above, but it includes user input in the 'Email' field. The 'Email' field now contains the value 'test@example.com'. The rest of the page, including the 'Connexion' button and the footer navigation sections, remains the same.

 BikeShop

[Accueil](#) Tout Categories Marques

[Mon Compte](#)  0

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

 BikeShop

[Accueil](#) Tout Categories Marques

[Mon Compte](#)  0

[Commandes](#)
[Informations](#)
[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.



[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

BikeShop

Accueil Tout Categories Marques

Connexion

0

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

➤ 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

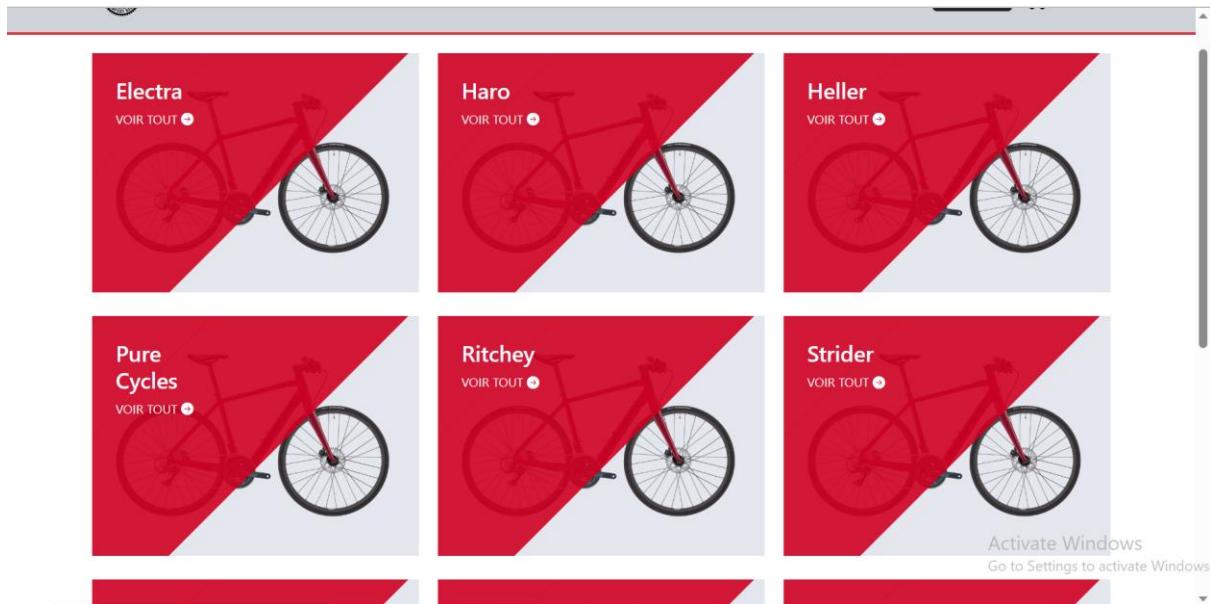
Santa Cruz Bikes
3700 Portola Drive, Santa Cruz, CA, 95060
(831) 476-4321
santacruz@bikes.shop
10

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 from a side-on perspective. Below each bike is a small description: 'MOUNTAIN BIKES' followed by the model name and price, and the brand name 'Haro'. The first three items have their prices (\$380, \$550, \$1470) displayed in red, while the fourth item's price (\$540) is displayed in grey. A vertical scrollbar is visible on the right side of the page.

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
			TOTAL	\$3,300.00	

[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

The screenshot shows a shopping cart summary for a Haro Flightline Two 26 Plus - 2017 bicycle. The cart contains 6 items at \$550.00 each, totaling \$3,300.00. The bicycle is listed as 'Santa Cruz Bikes'.

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

[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

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
TOTAL	\$3,300.00

[Placer Commande](#)

Activate Windows
Go to Settings to activate Windows

BikeShop

Accueil Tout Categories Marques

Mon Compte 0

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 0

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. The main title is "Liste de vos commandes" and the subtitle is "Numéro Commande : 1616". Below this, a table displays the details of the order:

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

At the bottom of the page, there are four sections: "A PROPOS DE NOUS", "SUPPORT", "INFORMATIONS", and "SERVICES".

➤ Modification du stock

The screenshot shows a product page for a Haro Flightline Two 26 Plus - 2017 bicycle. The product image is a red mountain bike. Below it is a quantity selector set to "1" and a red "AJOUTER AU PANIER" button. To the right of the product image, the product details are listed: "HARO FLIGHTLINE TWO 26 PLUS - 2017", "\$550.00", "CATEGORY: MOUNTAIN BIKES", "MARQUE: HARO", and "ANNÉE: 2017". There are also two boxes for "Santa Cruz Bikes" and "Baldwin Bikes" with their respective addresses, phone numbers, emails, and follower counts.

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 table data:

	id	dtype	email
1	1459	Client	email@example.com
2	1458	Client	test@example.com

personne table data:

	nom	prenom	telephone
1	Dieng	John	791030018
2	Doe	John	791030018

Below the tables, a WHERE clause is shown: "WHERE ORDER BY id DESC".

➤ 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