



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

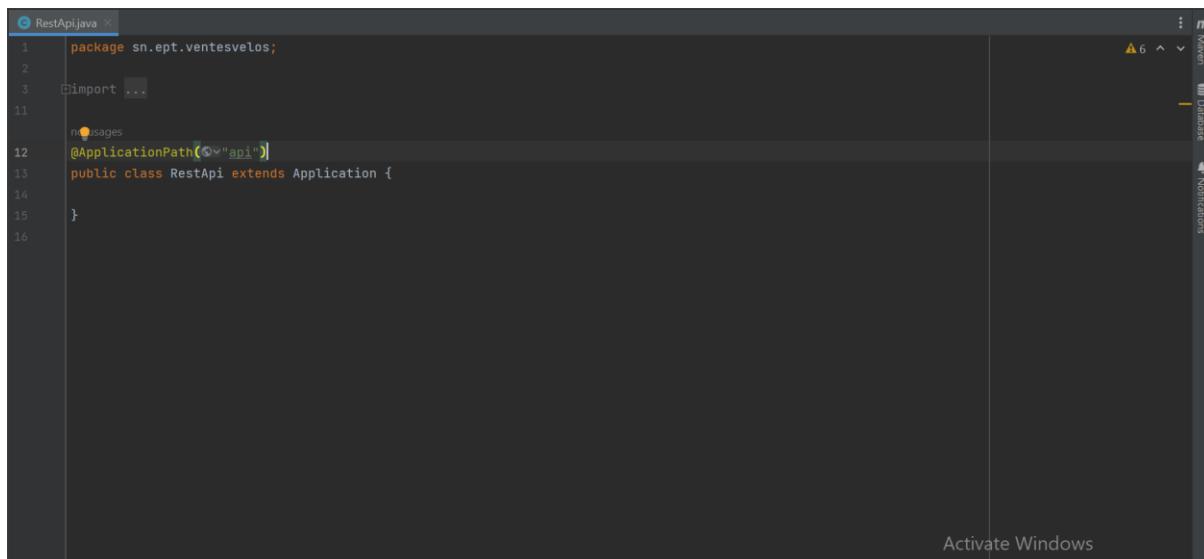
Pour cette ressource nous utilisons :

- 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".



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

Nous créons désormais nos ressources :

ArticleCommandeResource.java

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

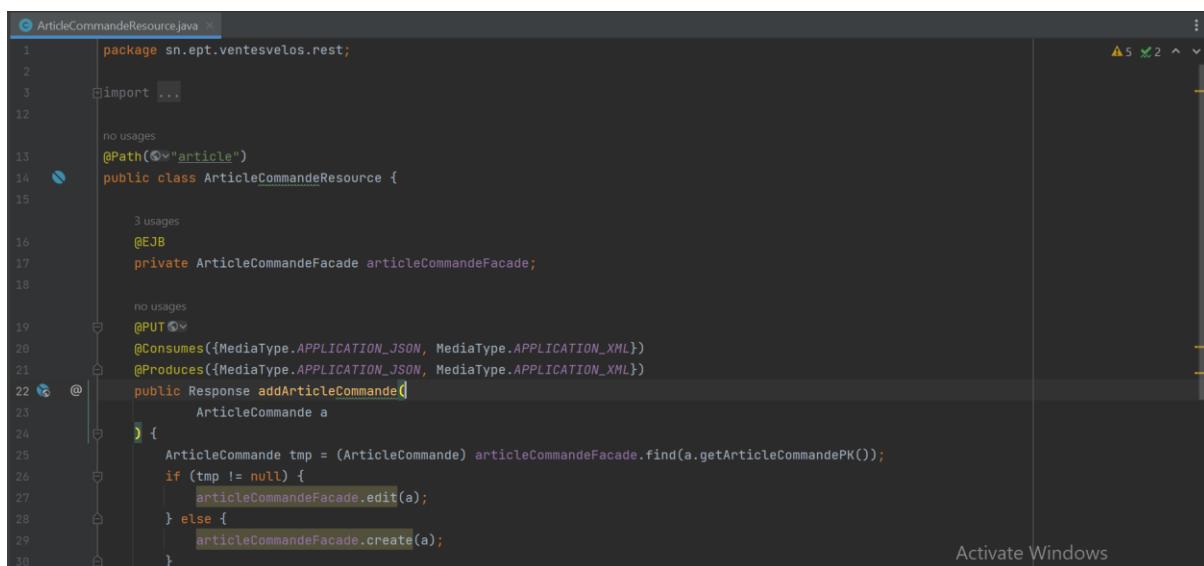
Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec `@GET`, `@PUT`, `@DELETE` suivant le type de requête devant permettre d'y accéder. En utilisant `@Consumes` (`{MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}`) ou `@Produces` (`{MediaType.APPLICATION_JSON,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

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

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



The screenshot shows a code editor window with the file `ArticleCommandeResource.java` open. The code defines a RESTful resource for managing article commands. It includes annotations for `@Path`, `@EJB`, `@PUT`, `@Consumes`, and `@Produces`. The `addArticleCommande` method handles the creation or update of an article command based on its primary key. The code uses a facade named `articleCommandeFacade` to perform the actual database operations. The code editor interface includes a status bar at the bottom right with the text "Activate Windows".

```
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`.

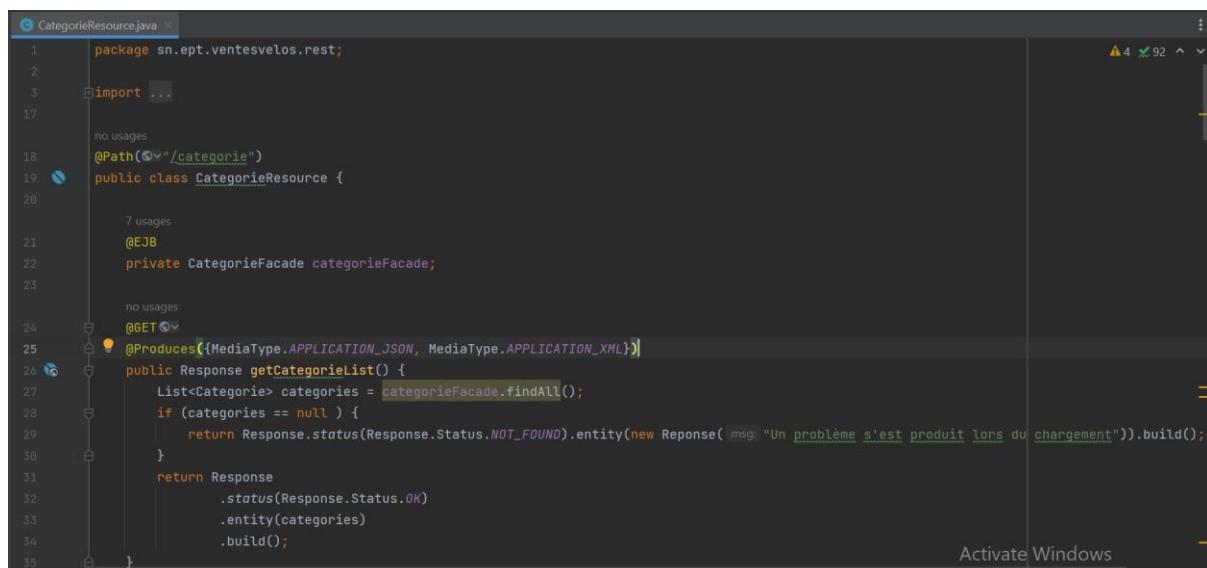
Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec `@GET`, `@PUT`, `@DELETE` suivant le type de requête devant permettre d'y accéder. En utilisant `@Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})` ou `@Produces ({MediaType.APPLICATION_JSON, 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 resource for categories. It includes annotations for `@Path`, `@EJB`, `@GET`, and `@Produces`. The `@GET` method returns a list of categories. The `@Produces` annotation specifies support for both JSON and XML formats. The code uses a private EJB facade to handle category operations.

```
1 package sn.ept.ventesvelos.rest;
2
3 import ...
4
5 no usages
6
7 @Path("/categorie")
8 public class CategorieResource {
9
10    7 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("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
28 }
```

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,`

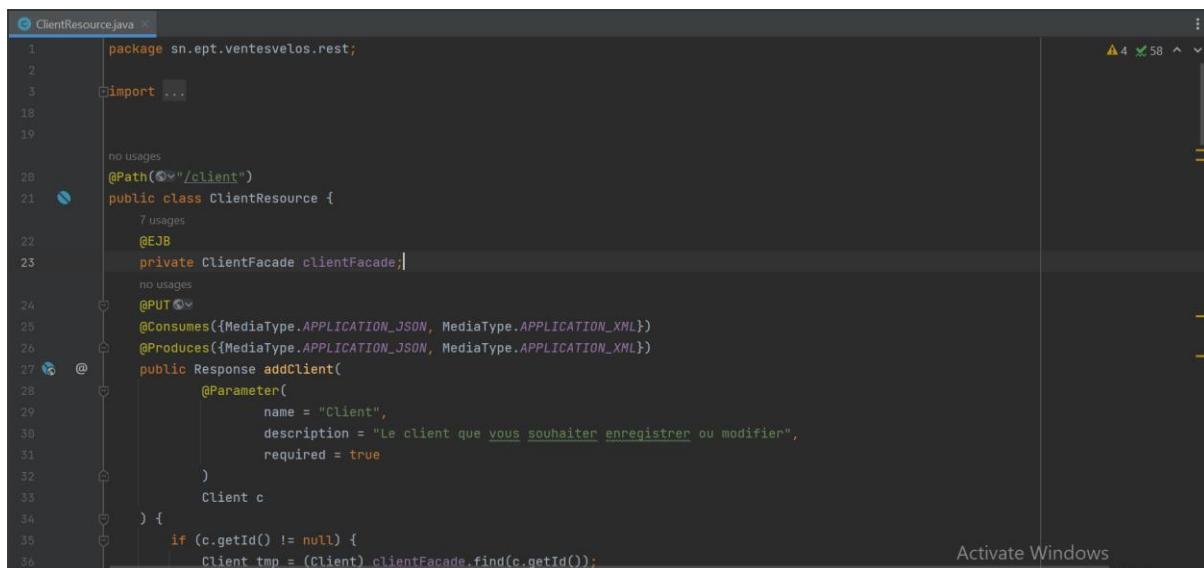
`MediaType.APPLICATION_XML})` pour spécifier qu'elles acceptent ou reçoivent des messages au format JSON et XML ou les deux annotations suivant le type de requête

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`.



The screenshot shows a code editor window for a Java file named `ClientResource.java`. The code defines a class `ClientResource` with a single method `addClient`. The `addClient` method is annotated with `@PUT`, `@Consumes(MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML)`, and `@Produces(MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML)`. It takes a parameter `c` of type `Client` with a description indicating it's for registering or modifying a client. The code then checks if the client exists in the database and updates it if necessary. The code editor interface includes a status bar at the bottom right with the text "Activate Windows".

```
1 package sn.upt.ventesvelos.rest;
2
3 import ...
4
5
6 no usages
7
8 @Path("/client")
9 public class ClientResource {
10
11     @EJB
12     private ClientFacade clientFacade;
13
14     @PUT
15     @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
16     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
17     public Response addClient(
18         @Parameter(
19             name = "Client",
20             description = "Le client que vous souhaitez enregistrer ou modifier",
21             required = true
22         )
23         Client c
24     ) {
25
26         if (c.getId() != null) {
27             Client tmp = (Client) clientFacade.find(c.getId());
28
29             if (tmp != null) {
30                 tmp.setName(c.getName());
31                 tmp.setDescription(c.getDescription());
32                 tmp.setAddress(c.getAddress());
33                 tmp.setPhone(c.getPhone());
34
35                 clientFacade.update(tmp);
36             }
37         }
38     }
39 }
```

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.

```
18 no usages
19 @Path("/commande")
20 public class CommandeResource {
21
22     7 usages
23     @JB
24     private CommandeFacade commandeFacade;
25
26     no usages
27     @PUT
28     @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
29     @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
30     public Response addCommande(
31         @Parameter(
32             name = "Commande",
33             description = "La commande que vous souhaitez ajouter",
34             required = true
35         )
36         Commande c
37     ) {
38
39         if (c.getNumero() != null) {
40             Commande tmp = (Commande) commandeFacade.find(c.getNumero());
41             if (tmp != null) {
42                 commandeFacade.edit(c);
43             }
44         }
45     }
46 }
```

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.

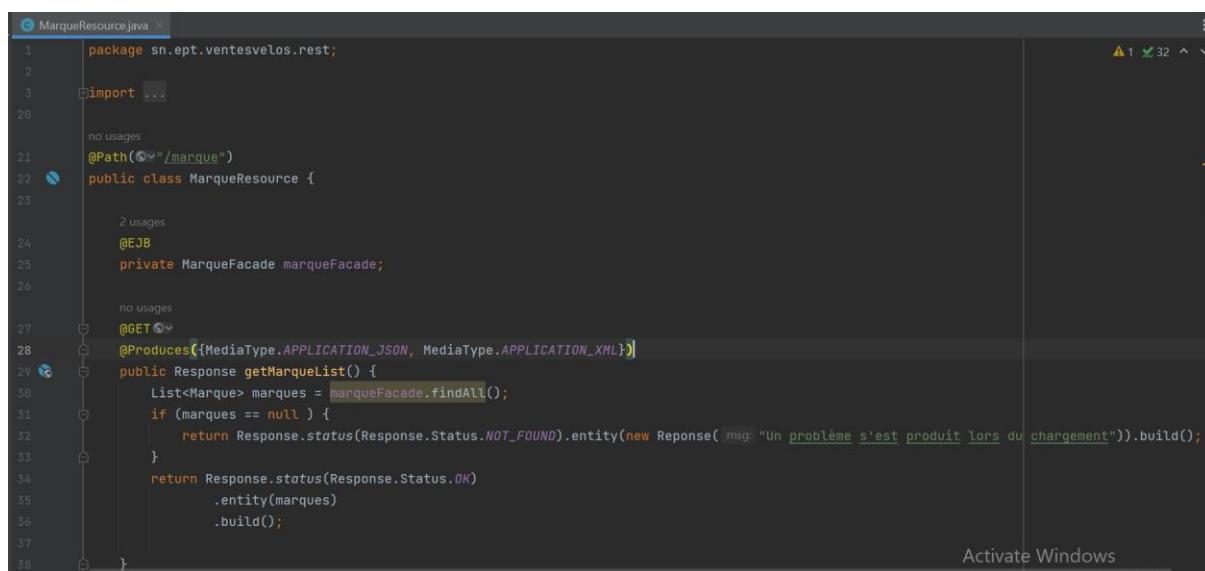
Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec @GET, @PUT, @DELETE suivant le type de requête devant permettre d'y accéder. En utilisant @Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}) ou @Produces ({MediaType.APPLICATION_JSON, 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 a code editor window for a Java file named MarqueResource.java. The code defines a class MarqueResource with a single method getMarqueList(). The method uses annotations @Path("/marque") and @GET, and specifies @Produces(MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML) to indicate it returns JSON or XML responses. The body of the method retrieves all Marque objects from a facade and returns them as a Response object. A tooltip for the Response class is visible at the bottom right of the code editor.

```
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     no usages
28     @PUT
29     @Consumes(MediaType.APPLICATION_JSON)
30     public Response addMarque(@PathParam("id") Integer id, Marque marque) {
31         if (marque == null) {
32             return Response.status(Response.Status.BAD_REQUEST).entity("La marque ne peut pas être null").build();
33         }
34         MarqueFacade.create(marque);
35         return Response.status(Response.Status.CREATED).entity("La marque a été ajoutée").build();
36     }
37
38 }
```

ProduitResource.java

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

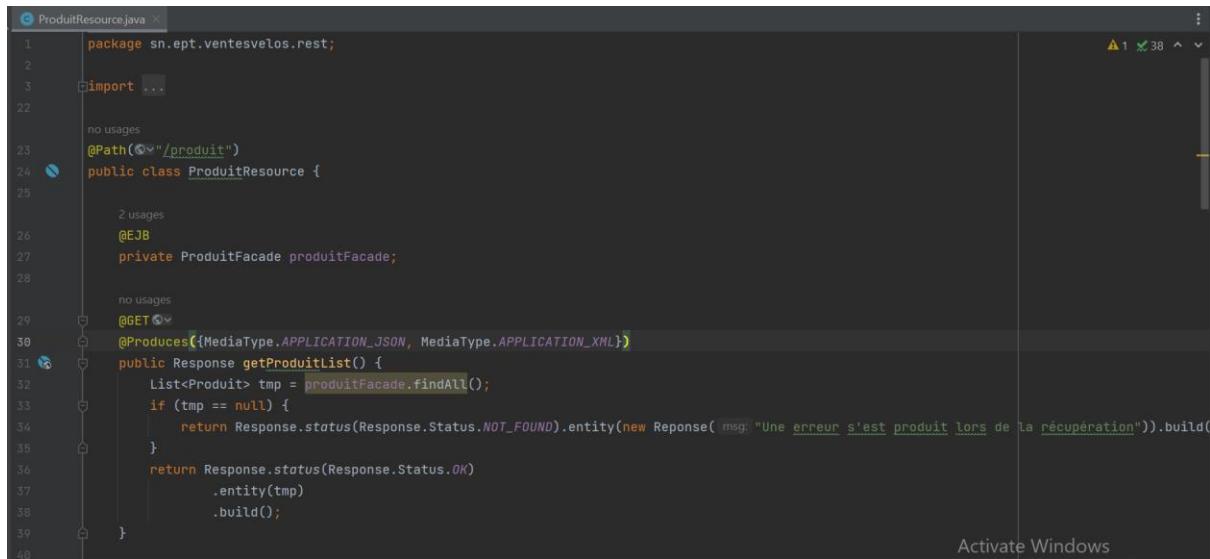
Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec `@GET`, `@PUT`, `@DELETE` suivant le type de requête devant permettre d'y accéder. En utilisant `@Consumes` (`{MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML}`) ou `@Produces` (`{MediaType.APPLICATION_JSON,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

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

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 faîte 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.

```

19  @Path("/empLoyer")
20  public class EmployeResource {
21      ...
22      private EmployeFacade employeFacade;
23
24      @PUT
25      @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
26      @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
27      public Response addEmploye(
28          @Parameter(
29              name = "Employe",
30              description = "L'employé que vous souhaitez enregistrer ou modifier",
31              required = true
32          )
33          Employe e
34      ) {
35          if (e.getId() != null) {
36              Employe tmp = (Employe) employeFacade.find(e.getId());
37              if (tmp != null) {
38                  employeFacade.edit(e);
39              }
39          } else {
40              employeFacade.create(e);
41          }
42      }
43  }

```

Activate Windows
Go to Settings to activate Windows

MagasinResource.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`.

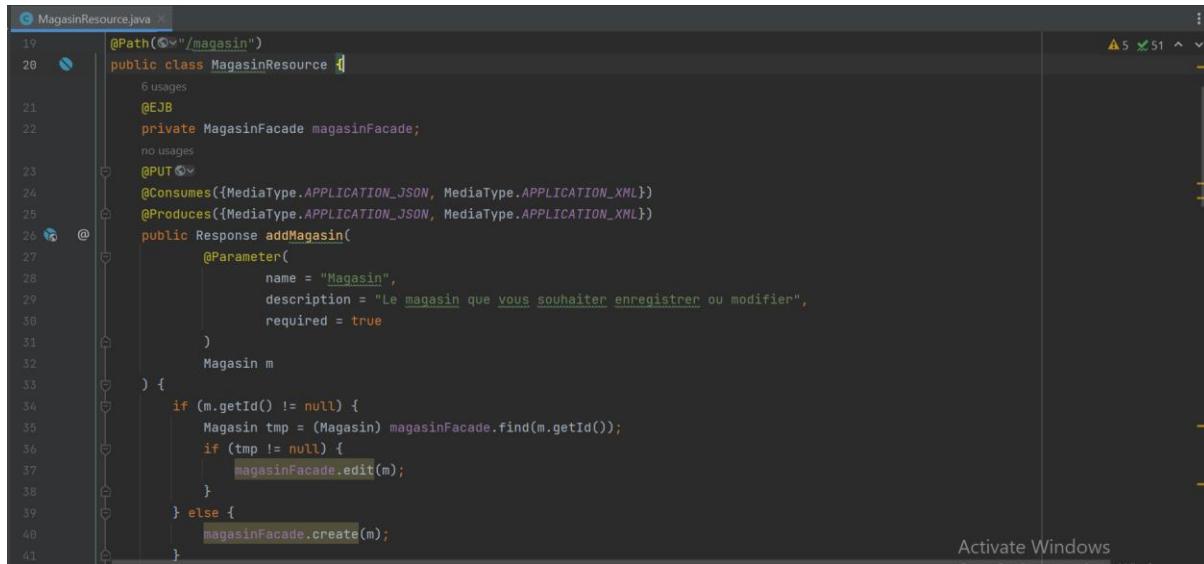
Ensuite pour chacune des actions possibles nous créons les méthodes permettant de les effectuer en les annotant avec `@GET`, `@PUT`, `@DELETE` suivant le type de requête devant permettre d'y accéder. En utilisant `@Consumes ({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})` ou `@Produces ({MediaType.APPLICATION_JSON, 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 `deleteMagasin()` dans laquelle nous supprimons un magasin spécifique dont l'id est passé en paramètre à la requête qui sera envoyé comme réponse. 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.



```
19  @Path("/magasin")
20  public class MagasinResource {
21      6 usages
22      @EJB
23      private MagasinFacade magasinFacade;
24      no usages
25      @PUT
26      @Consumes({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
27      @Produces({MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML})
28      public Response addMagasin(
29          @Parameter(
30              name = "Magasin",
31              description = "Le magasin que vous souhaitez enregistrer ou modifier",
32              required = true
33          )
34          Magasin m
35      ) {
36          if (m.getId() != null) {
37              Magasin tmp = (Magasin) magasinFacade.find(m.getId());
38              if (tmp != null) {
39                  magasinFacade.edit(m);
40              } else {
41                  magasinFacade.create(m);
42              }
43          }
44      }
45  }
```

StockResource.java

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

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

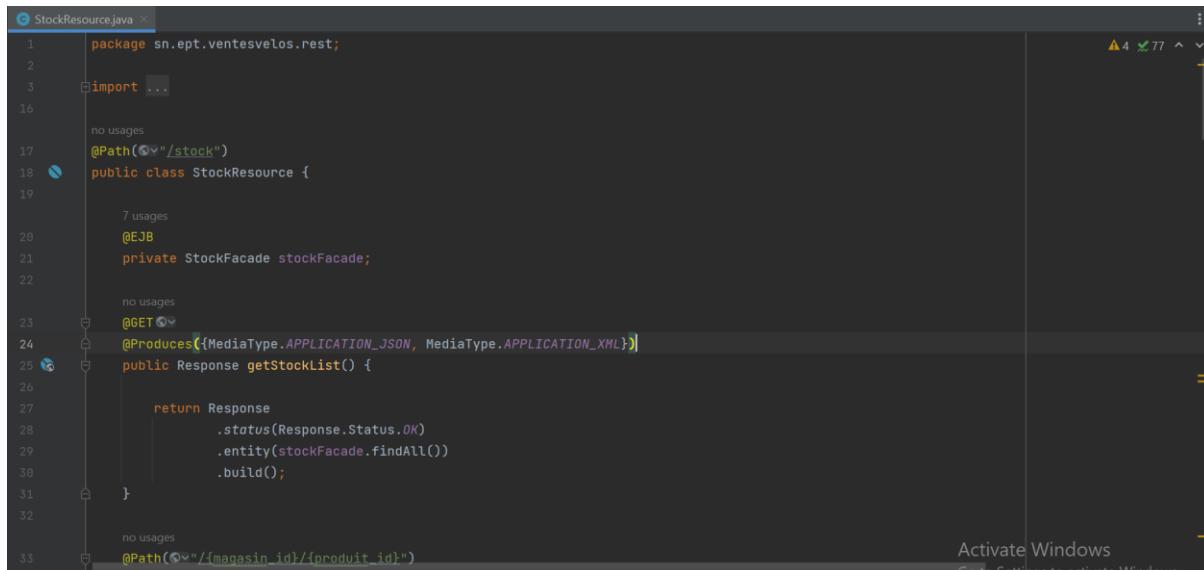
Ici il s'agit donc de créer une méthode getStockList() dans laquelle nous récupérons l'ensemble des stocks présents dans la base de données que nous envoyons comme réponse. On spécifie l'annotation @GET pour spécifier que le type de requête permettant d'y accéder est GET ainsi que l'annotation @Produces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

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

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

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

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



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

A cela nous ajoutons deux classes supplémentaires

Reponse.java

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

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

Activate Windows
Go to Settings to activate Windows

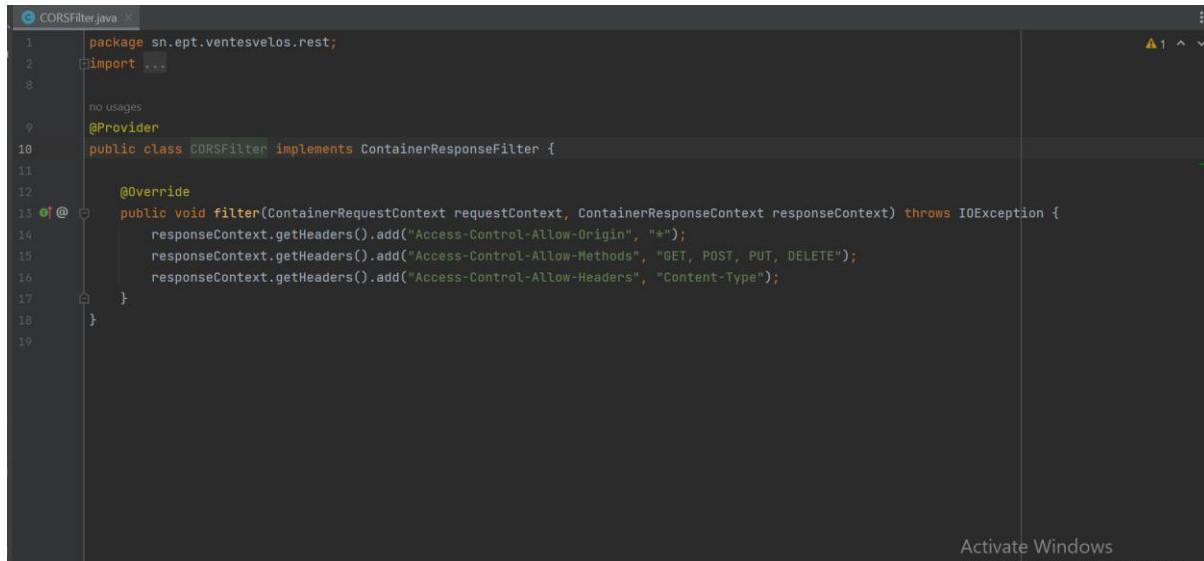
CORSFilter.java

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

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

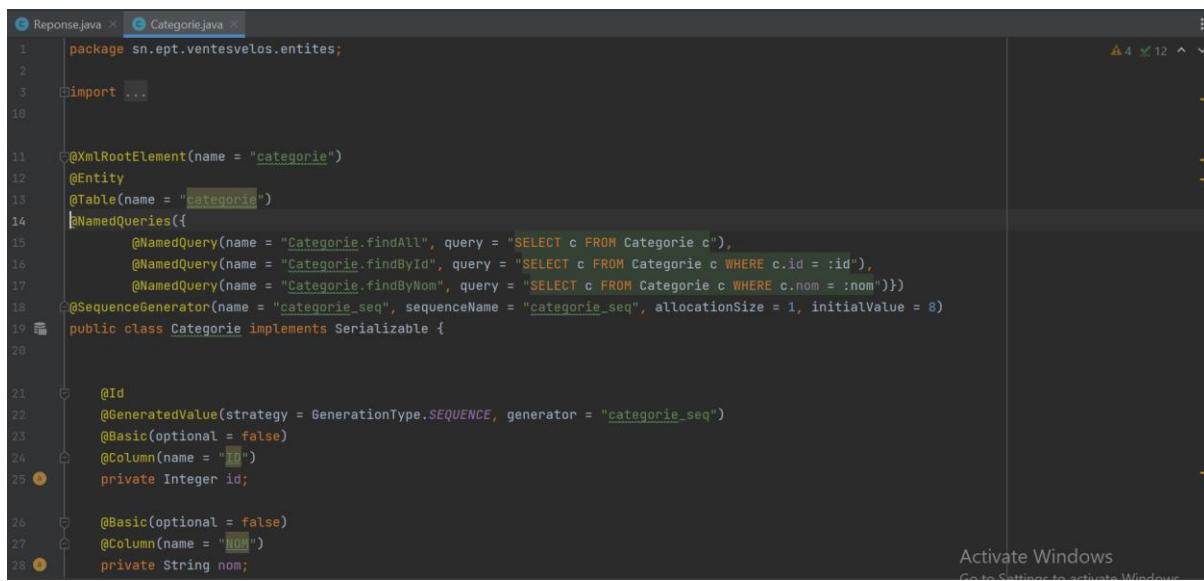
L'en-tête "Access-Control-Allow-Methods" spécifie les méthodes HTTP autorisées pour les requêtes cross-origin.

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



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

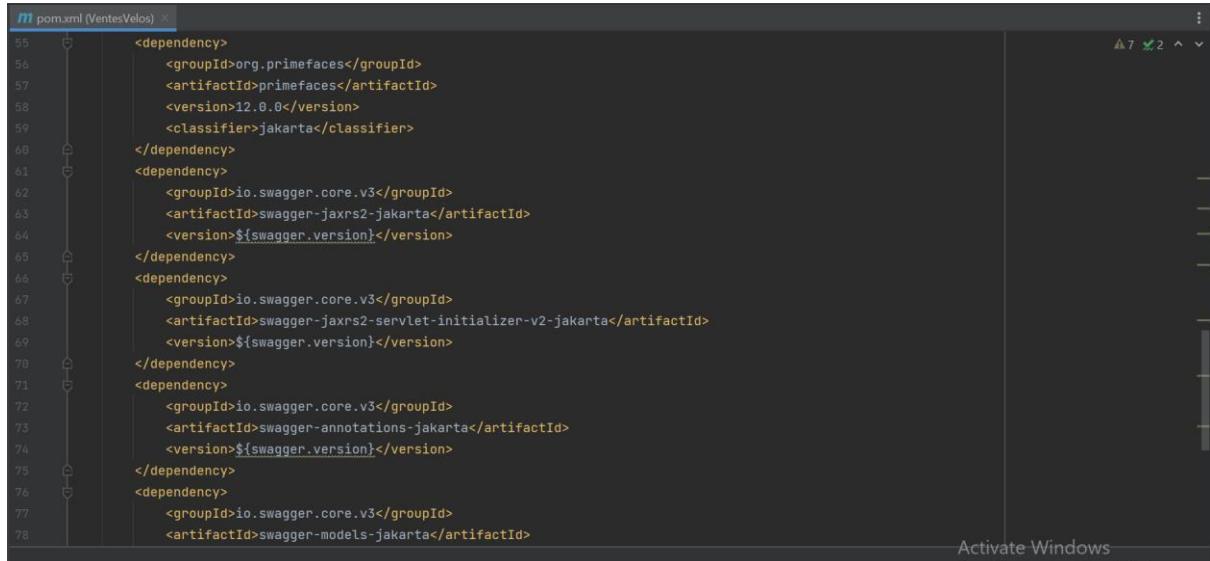
Pour les besoins de la communication au format XML on ajoute également l'annotation @XmlRootElement à nos entités. Comme on peut le voir dans cet exemple avec Catégorie.



```
1 package sn.ept.ventesvelos.entites;
2 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;
```

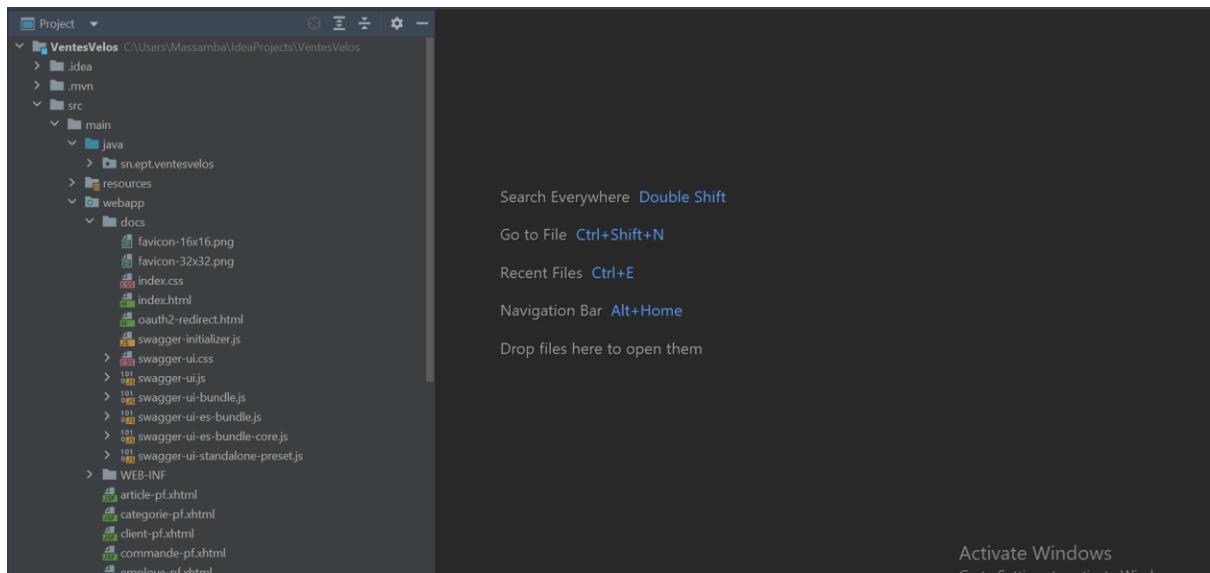
2. Documentation des services web

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.

```

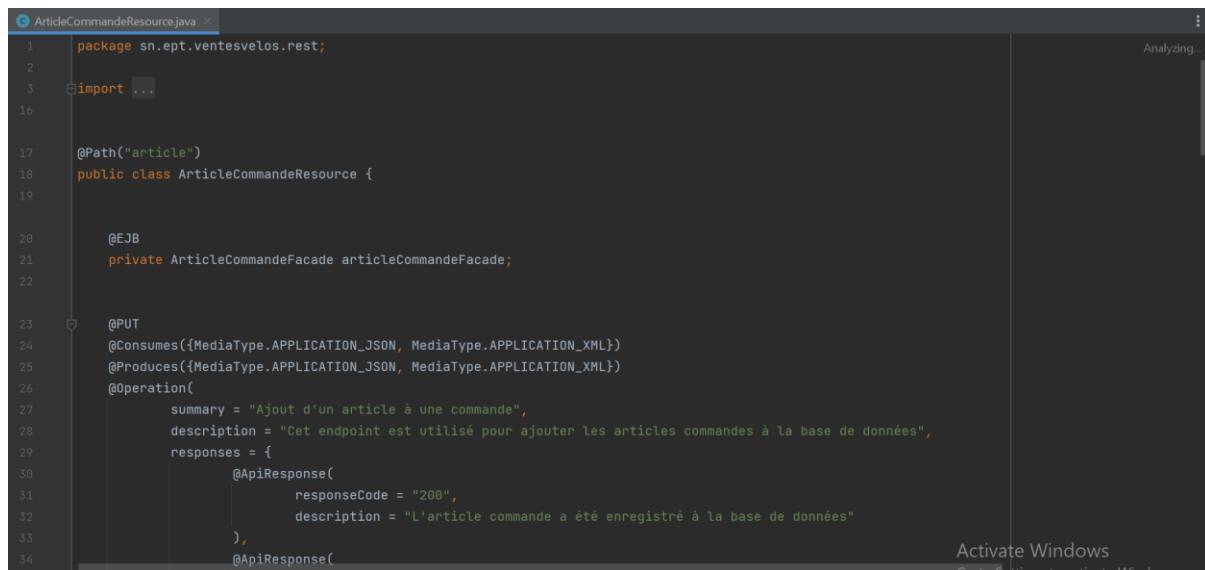
13  @OpenAPIDefinition(
14      info = @Info(
15          title = "API POUR LA BOUTIQUE DE VENTES DE VELOS",
16          description = "Ceci est une documentation Swagger définissant les endpoints pour les différentes ressources nécessaire aux applications",
17          contact = @Contact(
18       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

```

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

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

Comme nous pouvons le voir dans cet exemple :



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

```
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 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 API endpoints under the 'default' category. Each endpoint is represented by a row with its method (e.g., GET, PUT, DELETE), path, and a brief description. The rows are color-coded: blue for GET, orange for PUT, red for DELETE, and light blue for other methods like LIST.

Method	Path	Description
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
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
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

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.

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

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

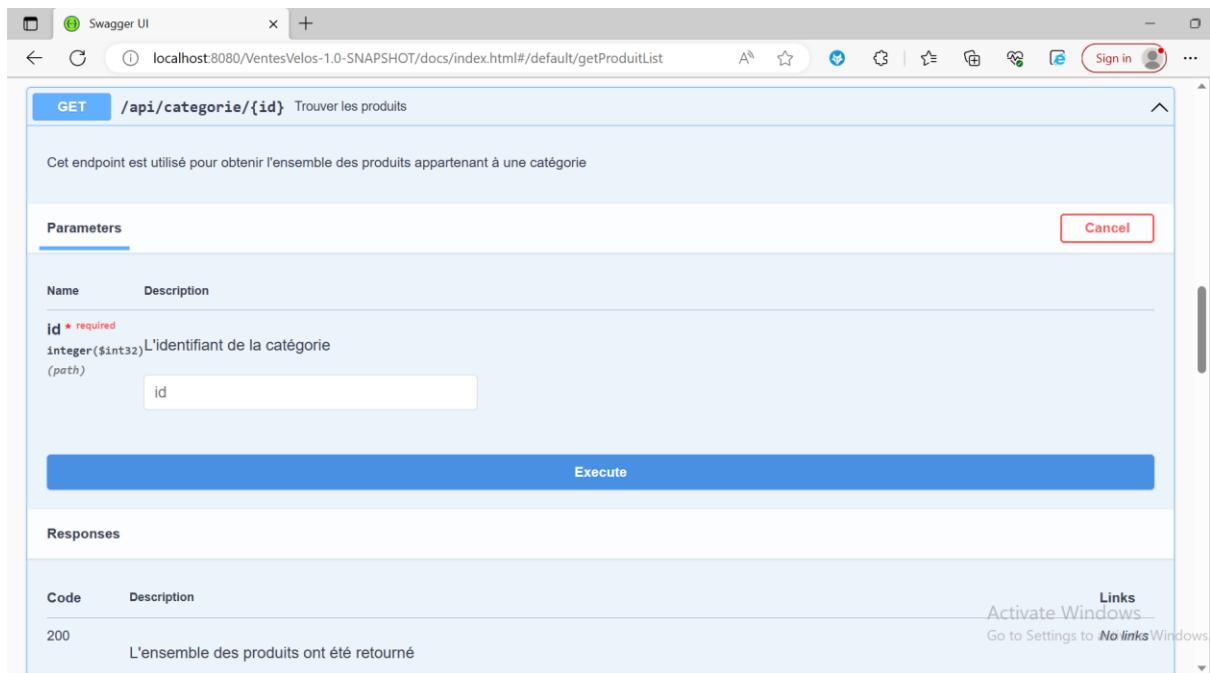
Responses

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 IntelliJ IDEA interface. At the top, the 'New Project' dialog is open, showing the configuration for creating a new Java project named 'ResourceTest'. The 'Language' is set to Java, 'Build system' to Maven, and 'JDK' to 'corretto-17 java version "17.0.2"'. The 'Advanced Settings' section includes 'GroupId: sn.ept' and 'ArtifactId: ResourceTest'. Below the dialog, the 'pom.xml' file for the 'ResourceTest' project is displayed, showing the XML code for dependencies, including JUnit and RestAssured.

```
<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>
</dependency>
```

```

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.

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

- File Bar:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help.
- Project Bar:** ResourceTest > src > main > java > sn > ept > ArticleTest > testGetArticleList
- Code Editor:** ArticleTest.java (Line 19: @Test)
- Run Tab:** ArticleTest selected, Tests passed: 7 of 7 tests - 9 sec 383 ms.
- Toolbars:** Version Control, Run, Terminal, Problems, Services, Profiler, Build, Dependencies.
- Status Bar:** Tests passed: 7 (moments ago), 123:36 CRLF UTF-8 4 spaces.

The code editor displays the following Java test code:

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

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.

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ResourceTest - CategorieTest.java

ResourceTest src main java sn ept CategorieTest testGetProduitListJSON

CategorieTest.java

```
public class CategorieTest {  
    public static void setup() { RestAssured.baseURI = "http://localhost:8880/VentesVélos-1.0-SNAPSHOT/api"; }  
  
    public void testGetCategorieList() {  
        given()  
            .header("Accept", MediaType.APPLICATION_JSON)  
        .when()  
            .get("/categorie")  
        .then()  
            .validatableResponse  
    }  
}
```

Run: CategorieTest

Tests passed: 9 of 9 tests – 4 sec 927 ms

CategorieTest (sn.ept) 4 sec 927 ms
testDeleteCategorie 3 sec 70 ms
testAddCategorieX 1 sec 311 ms
testGetProduitListJSON 262 ms
testAddCategorieJSON 90 ms
testDeleteCategorieNotF 31 ms
testDeleteCategorieNotF 31 ms
testGetProduitListNotF 24 ms
testDeleteCategorieXML 78 ms
testGetCategorieList 30 ms

"C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" ...
Process finished with exit code 0

Activate Windows
Go to Settings to activate Windows.

Version Control Run TODO Problems Terminal Services Profiler Build Dependencies

Tests passed: 9 (2 minutes ago)

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 with the following details:

- Code Editor:** The file `ClientTest.java` is open, showing Java code using RestAssured for API testing. It includes a `@BeforeAll` setup method and several `@Test` methods for client operations.
- Run Tool Window:** The `ClientTest` run configuration is selected. The results show 10 tests passed in 5 seconds, with individual test times listed.
- Status Bar:** Shows "Tests passed: 10 (moments ago)" and the system status: "Activate Windows Go to Settings to activate Windows".

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.

```

11  public class CommandeTest {
12
13      no usages
14      @BeforeAll
15      public static void setup() { RestAssured.baseURI = "http://localhost:8880/VentesVelos-1.0-SNAPSHOT/api"; }
16
17      no usages
18      @Test
19      public void testAddCommandeJSON() {
20          String requestBody = "{\"date_commande\": \"2023-06-09\", \"date_livraison_voulee\": \"2023-06-30\", \"status\": 3, "
21          + "\"clientId\": {\"id\": 136}, \"vendeurId\": {\"id\": 1453}, \"magasinId\": {\"id\": 3}}";
22
23          given().RequestSpecification()

```

Run: CommandeTest

- ✓ CommandeTest (sn.ept) 6 sec 496 ms "C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" ...
- ✓ testDeleteCommande 1 sec 870 ms
- ✓ testDeleteCommandeJSON 152 ms
- ✓ testGetCommandeList 3 sec 468 ms
- ✓ testAddCommandeJSON 839 ms
- ✓ testGetArticleCommandeList 26 ms
- ✓ testDeleteCommandeXML 29 ms
- ✓ testGetArticleCommandeList 24 ms
- ✓ testGetArticleCommandeList 69 ms
- ✓ testDeleteCommandeNotFound 19 ms

Process finished with exit code 0

Activate Windows
Go to Settings to activate Windows.

Tests passed: 9 (moments ago)

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.

```

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

```

Run: MarqueTest

- ✓ MarqueTest (sn.ept) 3 sec 241 ms "C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" ...
- ✓ testGetProductListJSON 1 sec 383 ms
- ✓ testGetMarqueListJSON 32 ms
- ✓ testDeleteMarqueNotFound 37 ms
- ✓ testDeleteMarqueNotFound 26 ms
- ✓ testDeleteMarqueXML 935 ms
- ✓ testDeleteMarqueJSON 134 ms
- ✓ testAddMarqueJSON 671 ms
- ✓ testGetProductListNotFound 23 ms

Process finished with exit code 0

Activate Windows
Go to Settings to activate Windows.

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:

- Project:** ResourceTest
- Code Editor:** Shows the `ProductTest.java` file containing Java code for RestAssured tests.
- Run Tool Window:** Shows the results of the test run:
 - Tests passed: 8 of 8 tests - 2 sec 417 ms
 - ProductTest (sn.ept) 2 sec 417 ms "C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" ...
 - testGetProductListISO 1 sec 573 ms
 - testDeleteProductXML() 54 ms
 - testDeleteProductNotFound() 24 ms
 - testGetStockListNotFound() 23 ms
 - testDeleteProductNotFound() 19 ms
 - testDeleteProductJSON() 79 ms
 - testAddProduct() 586 ms
 - testGetProductJSON() 59 ms
- Status Bar:** Activate Windows, Go to Settings to activate Windows, 50:28, CRLF, UTF-8, 4 spaces

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.

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

- Project Bar:** ResourceTest > src > main > java > sn > ept > EmployeeTest
- Code Editor:** EmployeeTest.java


```
public class EmployeeTest {
    ...
    @BeforeAll
    public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVelos-1.0-SNAPSHOT/api"; }

    ...
    @Test
    public void testDeleteEmployeeNotFound() {
        int id=1601;
        given()
            .header("Accept", MediaType.APPLICATION_XML)
            .pathParam("id", id)
    }
}
```
- Run Tool Window:**
 - Shows the test results for EmployeeTest (sn.ept): 2 sec 307 ms.
 - Test cases listed: testGetEmployeeList(), testAddEmployee(), testDeleteEmployee(), testDeleteEmployeeNotFound(). All passed.
 - Output: "Process finished with exit code 0".
- Bottom Status Bar:** Tests passed: 4 (moments ago) | 10:1 CRLF UTF-8 4 spaces

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.

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

- Project Bar:** ResourceTest > src > main > java > sn > ept > MagasinTest > testAddMagasin
- Code Editor:** MagasinTest.java


```
public class MagasinTest {
    ...
    @BeforeAll
    public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVelos-1.0-SNAPSHOT/api"; }

    ...
    @Test
    public void testDeleteMagasinNotFound() {
        int id=1601;
        given()
            .header("Accept", MediaType.APPLICATION_XML)
    }
}
```
- Run Tool Window:**
 - Shows the test results for MagasinTest (sn.ept): 2 sec 600 ms.
 - Test cases listed: testDeleteMagasin(), testDeleteMagasinNotFound(), testGetMagasinList(), testAddMagasin(). All passed.
 - Output: "Process finished with exit code 0".
- Bottom Status Bar:** Tests passed: 4 (moments ago) | 47:32 CRLF UTF-8 4 spaces

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 contains a `setup()` method setting the base URI to `"http://localhost:8080/VentesVelos-1.0-SNAPSHOT/api"`, and an `@Test` 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 ms. The command used was `"C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" ...`.

```
14     public static void setup() { RestAssured.baseURI = "http://localhost:8080/VentesVelos-1.0-SNAPSHOT/api"; }
17
18     no usages
19     @Test
20     public void testGetStockListJSON() {
21         given()
22             .header( "Accept", MediaType.APPLICATION_JSON)
23             .when()
24                 .get( "/stock" ) Response
```

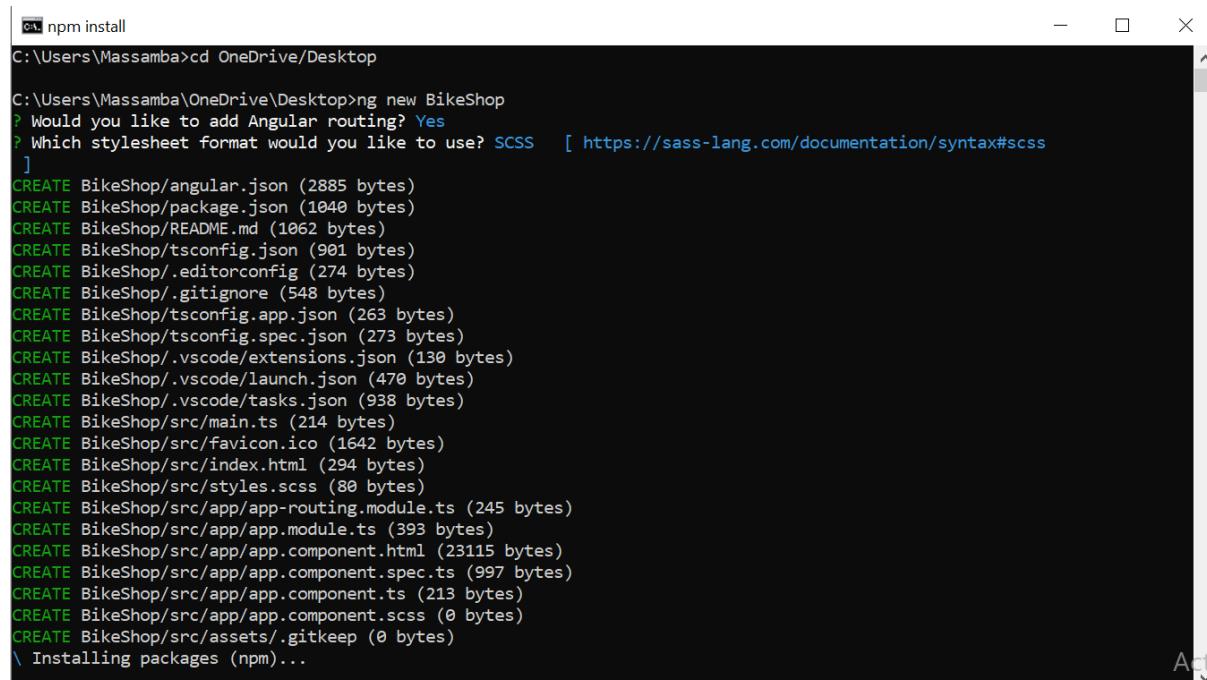
Run: StockTest ×

StockTest (suite) 5 sec 128 ms "C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe" ...
Process finished with exit code 0

Activate Windows

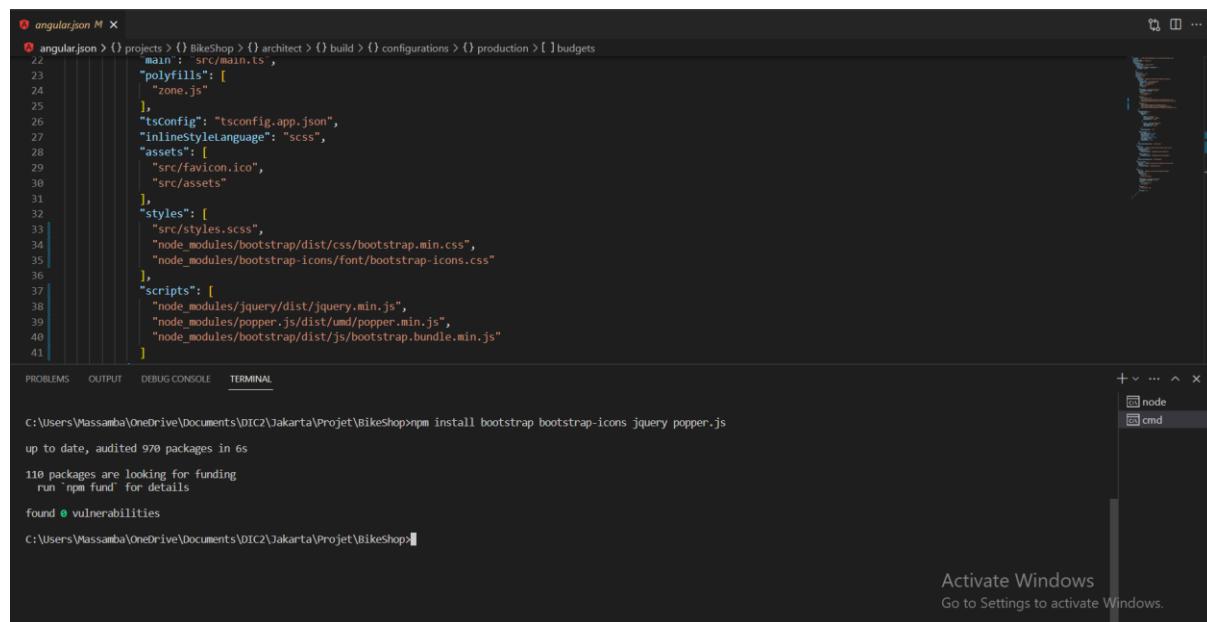
IV. Client Angular+bootstrap

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 fourni et on ajoute les chemins dans le fichier *angular.json*

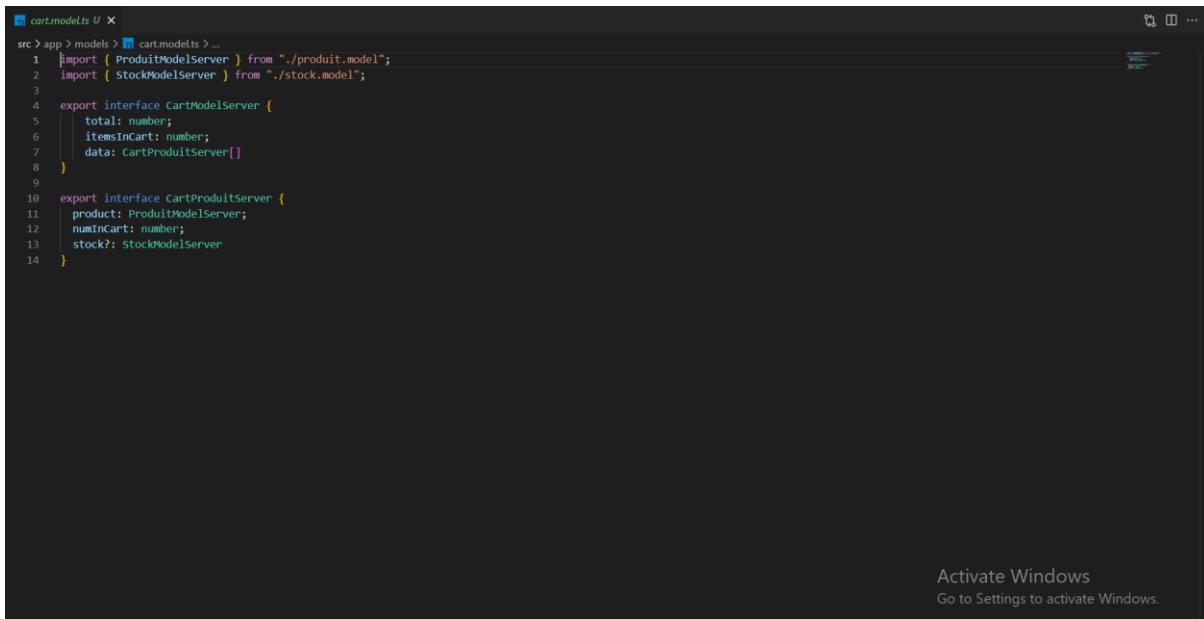


```
angular.json M
...
21   "main": "src/main.ts",
22   "polyfills": [
23     "zone.js"
24   ],
25   "tsConfig": "tsconfig.app.json",
26   "inlineStyleLanguage": "scss",
27   "assets": [
28     "src/favicon.ico",
29     "src/assets"
30   ],
31   "styles": [
32     "src/styles.scss",
33     "node_modules/bootstrap/dist/css/bootstrap.min.css",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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

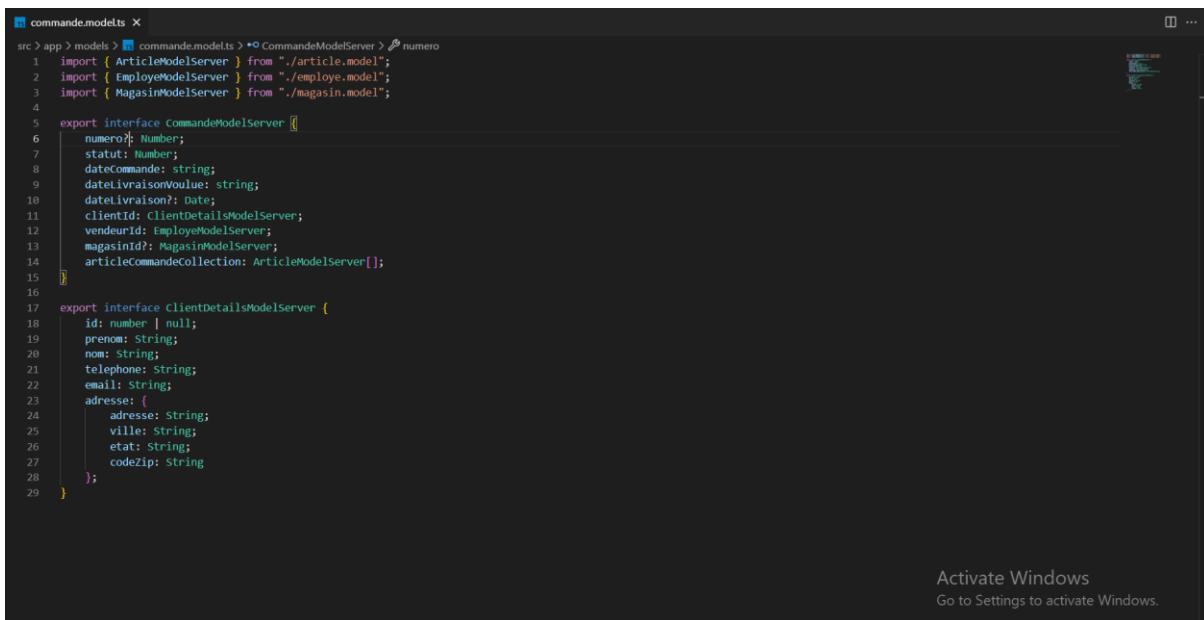
C:\Users\Massamba\OneDrive\Documents\JDK\Projet\BikeShop>npm install bootstrap bootstrap-icons jquery popper.js
up to date, audited 970 packages in 6s
110 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities
C:\Users\Massamba\OneDrive\Documents\JDK\Projet\BikeShop>
```

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



```
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.



```
src > app > models > commande.models > ...
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   dateLivraison?: string;
10  dateLivraisonVoulee: string;
11  clientID: ClientDetailsModelServer;
12  vendeurID: EmployeModelServer;
13  magasinID: MagasinModelServer;
14  articleCommandeCollection: ArticleModelServer[];
15 }
16
17 export interface ClientDetailsModelServer {
18   id: number | null;
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.

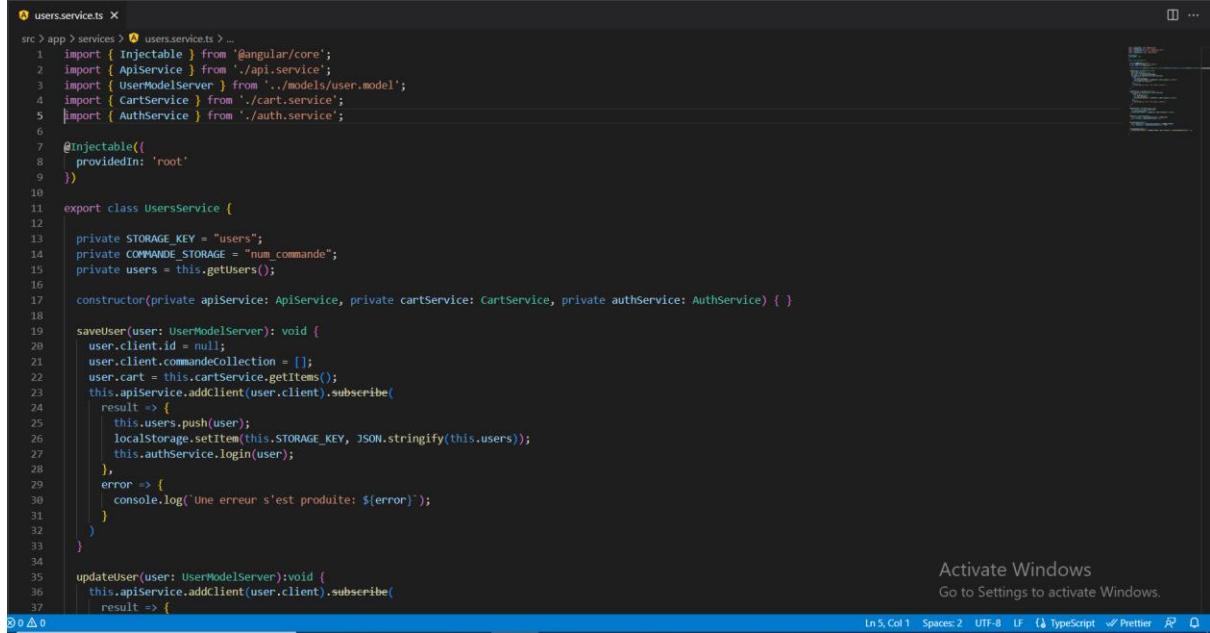
Nous créons également les services pour communiquer avec les services Web Restful (api), le panier (cart), les utilisateurs de l'application web (users) et l'authentification (auth)

```
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.



```

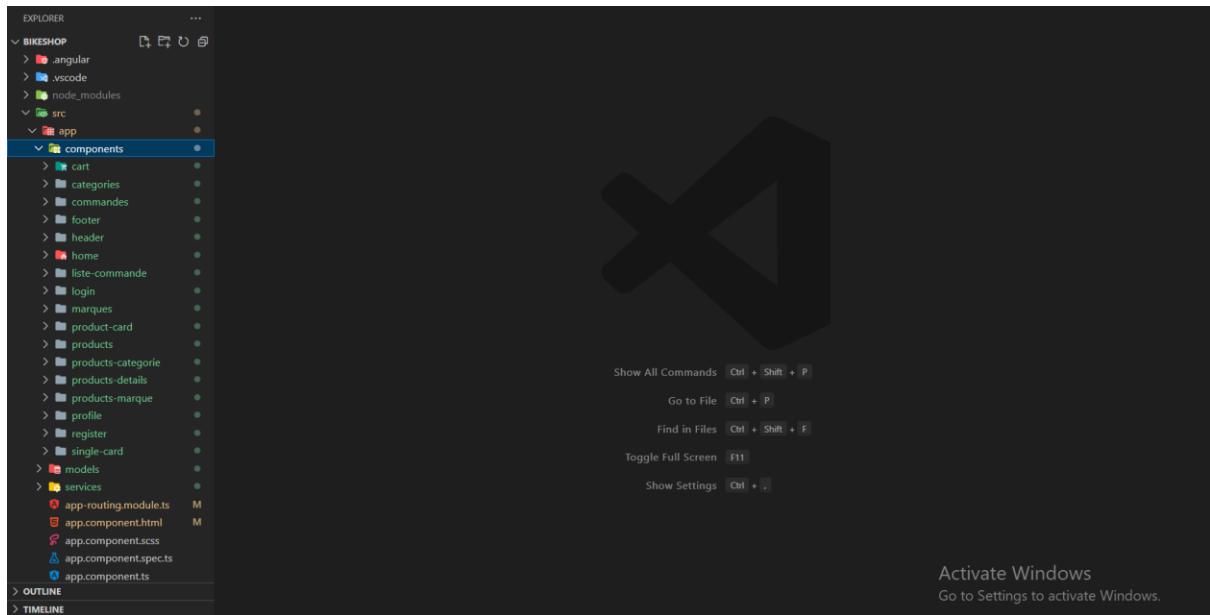
1 import { Injectable } from '@angular/core';
2 import { ApiService } from './api.service';
3 import { UserModelServer } from './models/user.model';
4 import { CartService } from './cart.service';
5 import { AuthService } from './auth.service';
6
7 @Injectable({
8   providedIn: 'root'
9 })
10
11 export class UserService {
12
13   private STORAGE_KEY = "users";
14   private COMMANDE_STORAGE = "num_commande";
15   private users = this.getUsers();
16
17   constructor(private apiService: ApiService, private cartService: CartService, private authService: AuthService) { }
18
19   saveUser(user: UserModelServer): void {
20     user.client.id = null;
21     user.client.commandeCollection = [];
22     user.cart = this.cartService.getItems();
23     this.apiService.addClient(user.client).subscribe(
24       result => {
25         this.users.push(user);
26         localStorage.setItem(this.STORAGE_KEY, JSON.stringify(this.users));
27         this.authService.login(user);
28       },
29       error => {
30         console.log(`Une erreur s'est produite: ${error}`);
31       }
32     );
33   }
34
35   updateUser(user: UserModelServer):void {
36     this.apiService.addClient(user.client).subscribe(
37       result => {

```

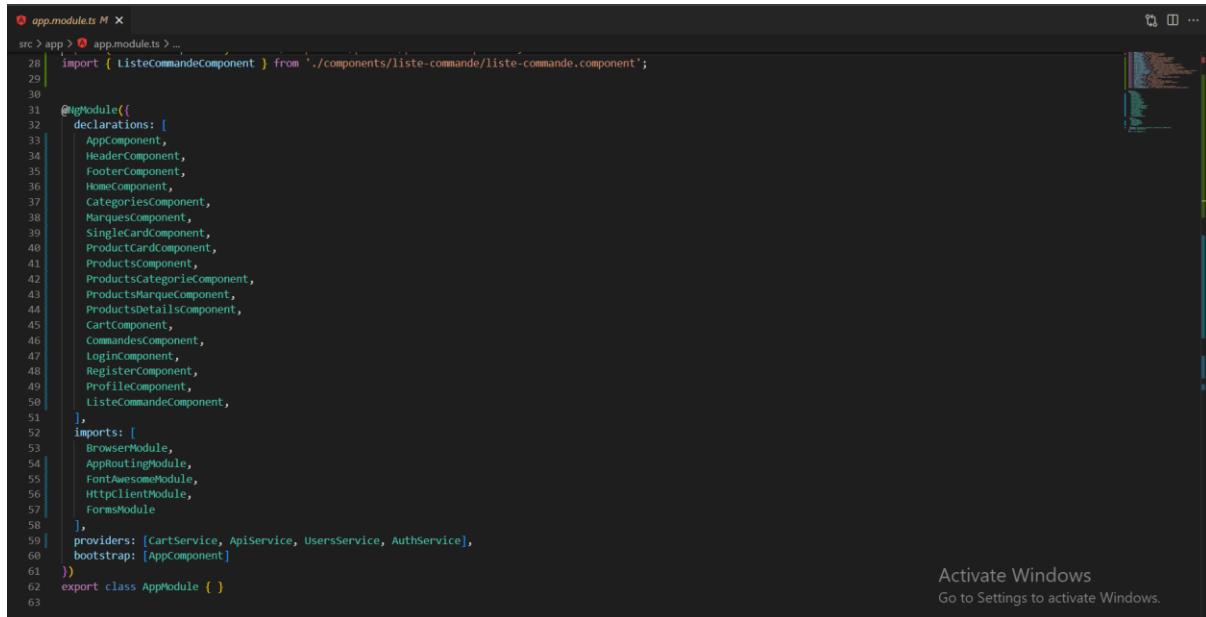
Activate Windows
Go to Settings to activate Windows.

Ln 5, Col 1 Spaces: 2 UTF-8 LF 🔍 typeScript 🖐️ Prettier ⚡

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



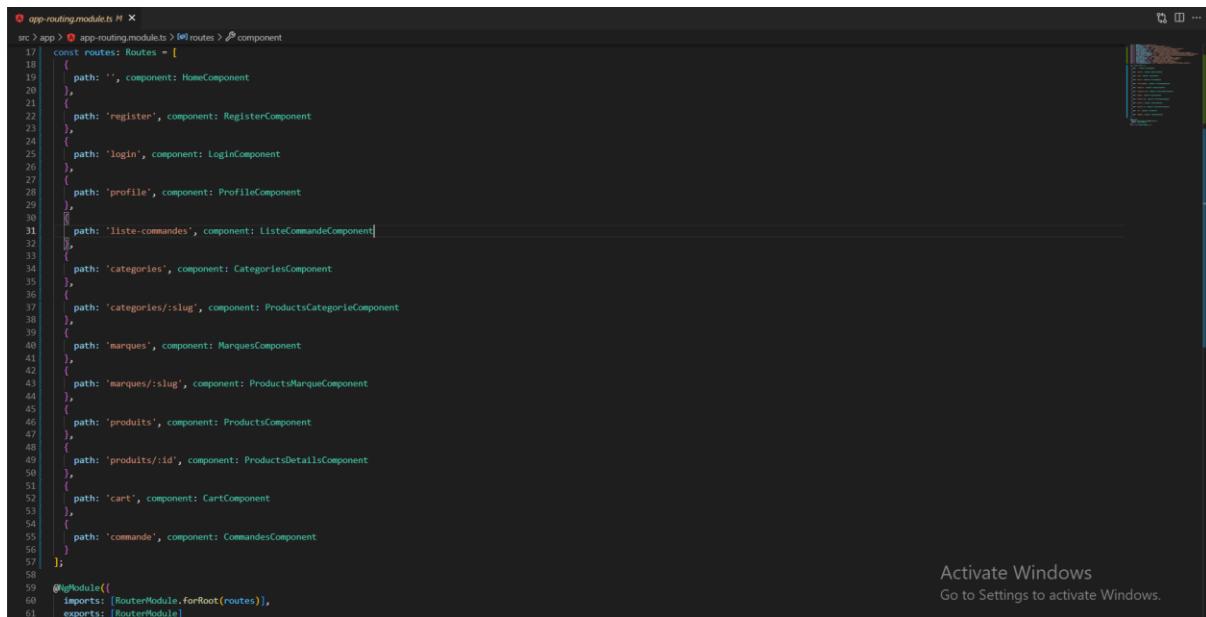
Nous importons les modules HttpClient, FormsModule, FontAwesomeModule, AppRoutingModule et nous ajoutons nos services au en tant que *providers*



```
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 { }
```

Activate Windows
Go to Settings to activate Windows.

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



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

Activate Windows
Go to Settings to activate Windows.

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

- Page Accueil

The screenshot shows a web browser window with the title bar "BikeShop" and the URL "localhost:4200". The page content includes a header with the BikeShop logo, navigation links for "Accueil", "Tout", "Categories", and "Marques", a "Connexion" button, and a shopping cart icon. Below the header is a section titled "Recherche, Achat, Livraison" with a descriptive paragraph about the website's purpose. A large image of a bike shop interior with many bicycles on display follows. At the bottom of the page is a footer with links for "A PROPOS DE NOUS", "SUPPORT", "INFORMATIONS", and "SERVICES", along with social media icons and a "Sign in" button.

➤ Page Inscription

BikeShop

Connexion 

Création de compte

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

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

[S'inscrire](#)

A PROPOS DE NOUS

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

SUPPORT

FAQ
Knowledge base

INFORMATIONS

A propos de nous
Contactez nous

SERVICES

Mon compte
Voir panier

Activate Windows
Go to Settings to activate Windows



BikeShop

Accueil Tout Categories Marques

Connexion 

Création de compte

Prenom
John

Nom
Doe

Telephone
791030018

Adresse
Thies Route VCN Nord

Etat
CA

Code Zip
A10

Ville
Thies

Email
test@example.com

Mot de passe

Confirmer mot de passe

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

[S'inscrire](#)

A PROPOS DE NOUS

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

SUPPORT

FAQ
Knowledge base

INFORMATIONS

A propos de nous
Contactez nous

SERVICES

Mon compte
Voir panier

Activate Windows
Go to Settings to activate Windows



BikeShop

Accueil Tout Categories Marques

Mon Compte 

Recherche, Achat, Livraison

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



A PROPOS DE NOUS

Nous sommes une boutique de ventes de

SUPPORT

FAQ

INFORMATIONS

A propos de nous

SERVICES

Mon compte

Activate Windows
Go to Settings to activate Windows

➤ Page Connexion

The screenshot shows the BikeShop login page. At the top, there is a header with the logo "BikeShop", a navigation bar with links "Accueil", "Tout", "Categories", "Marques", a "Connexion" button, and a shopping cart icon showing "0". Below the header, a section titled "Connectez Vous" contains two input fields: "Email" and "Mot de passe". A red-bordered "Connexion" button is centered below the fields. Below the button, a link "Pas encore de compte? Inscrivez-vous" is visible. At the bottom of the page, there is a footer menu with four 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 shows the same BikeShop login page as above, but with user input. The "Email" field now contains "test@example.com" and the "Mot de passe" field contains a series of asterisks ("*****"). The rest of the page structure, including the "Connexion" button and the footer menu, remains identical to the first screenshot.

 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

The screenshot shows the 'Modification de compte' (Account Modification) page. It features a form with fields for First Name (John), Last Name (Doe), Phone Number (791030018), Address (Thies Route VCN Nord), State (CA), Zip Code (A10), City (Thies), Email (test@example.com), and Password (*****). Below the form are 'Modifier' (Modify) and 'Supprimer' (Delete) buttons. At the top right, there is a 'Mon Compte' (My Account) button and a shopping cart icon with a '0'.

Commandes

The screenshot shows the 'Aucune Commande' (No Orders) page. It includes a 'Retourner' (Return) button. Below it is a footer with links: A PROPOS DE NOUS, SUPPORT, INFORMATIONS, and SERVICES. The SUPPORT section includes FAQ, Knowledge base, Garantie, and Nouveauté. The INFORMATIONS section includes A propos de nous, Contactez nous, Politique de confidentialité, and Termes & Conditions. The SERVICES section includes Mon compte, Voir panier, Aide, and Activate Windows (Go to Settings to activate Windows).

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

TREK 820 - 2016... \$380 Trek

MOUNTAIN BIKES

RITCHIE TIMBERWOLF F... \$750 Ritchey

MOUNTAIN BIKES

SURLY WEDNESDAY FRAM... \$1000 Surly

MOUNTAIN BIKES

TREK FUEL EX 8 29 ... \$2900 Trek

Activate Windows
Go to Settings to activate Windows

Sélectionner un produit spécifique



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

Qty
AJOUTER AU PANIER

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

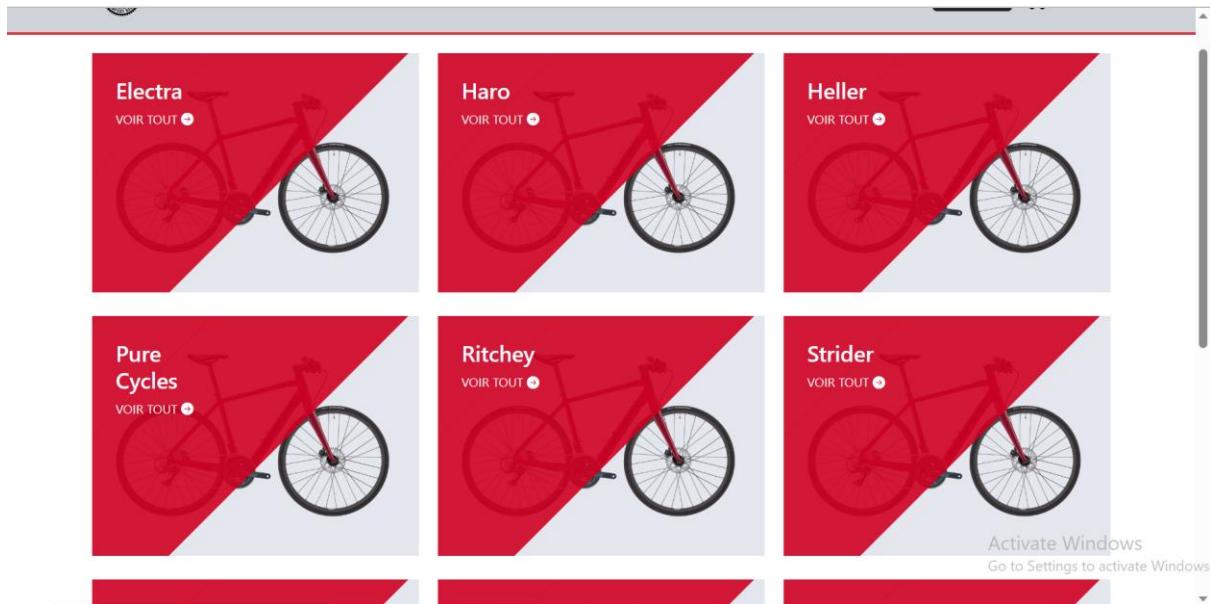
Santa Cruz Bikes
3700 Portola Drive, Santa Cruz, CA, 95060
(831) 476-4321
santacruz@bikes.shop
10

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 with its name and price: HARO FLIGHTLINE ONE ... \$380, HARO FLIGHTLINE TWO ... \$550, HARO SHIFT R3 - 2017... \$1470, and HARO SR 1.1 - 2017... \$540. The 'HARO' brand name is centered above the grid. A watermark for 'Activate Windows Go to Settings to activate Windows' is visible in the bottom right corner.

Selectionner un produit

➤ Ajout de produits au panier

The screenshot shows a shopping cart summary for a Haro Flightline Two 26 Plus - 2017 bicycle. The cart contains one item at \$550.00, totaling \$3,300.00. The cart icon indicates 1 item.

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

The screenshot shows the 'Création de compte' (Account Creation) form. It includes fields for First Name (John), Last Name (Dieng), Phone Number (791030018), Address (Thies Route VCN Nord), State (CA), Zip Code (A10), City (Thies), Email (email@example.com), and Password (*****). A confirmation field for the password is also present. A link to log in is provided at the bottom left.

Création de compte

Prenom: John Nom: Dieng Telephone: 791030018

Adresse: Thies Route VCN Nord

Etat: CA Code Zip: A10 Ville: Thies

Email: email@example.com

Mot de passe: ***** Confirmer mot de passe: *****

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

[S'inscrire](#)

A PROPOS DE

SUPPORT

INFORMATIONS

SERVICES

Activate Windows
Go to Settings to activate Windows

BikeShop

Accueil Tout Categories Marques

Mon Compte

1

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

BikeShop

Accueil Tout Categories Marques

Mon Compte

1

DETAILS LIVRAISON

Prenom: Dieng
Nom: Dieng
Telephone: 791030018

Adresse Mail: email@example.com

Adresse: Thies Route VCN Nord

Etat: CA
Code Zip: A10
Ville: Thies

Livraison Voulue: 06/29/2023

VOTRE COMMANDE

PRODUIT	SOUS TOTAL
6x Haro Flightline Two 26 Plus - 2017	\$3,300.00
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 single item in 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. On the left, there is a large image of the red bicycle, a quantity selector set to "1", and a red "AJOUTER AU PANIER" button. To the right, there is detailed product information: "HARO FLIGHTLINE TWO 26 PLUS - 2017", "\$550.00", "CATEGORY: MOUNTAIN BIKES", "MARQUE: HARO", and "ANNÉE: 2017". Below this, there are two boxes: "Santa Cruz Bikes" with address, phone, email, and a count of 7; and "Baldwin Bikes" with address, phone, email, and a count of 18. A "Activate Windows" message is also visible.

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". The "client" table has columns "id", "type", "email", and "adresse". The "personne" table has columns "nom", "prenom", and "telephone". The "WHERE" clause is set to "ORDER BY id DESC". The data is as follows:

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

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

➤ Nous pouvons également voir la commande créée

commande

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

➤ Nous pouvons voir l'article commande

article_commande

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

➤ La modification du stock

console_3

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

Services

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

Output

	quantite	produit_id	magasin_id
1	7	38	1