

Polytech Dijon 3A options :
Informatique & Réseaux
Module : ITC315

TP Web DAW FullStack

Auteur :
HUBERT Matéo

Professeur référent :
MEUNIER Charles



POLYTECH[®]
DIJON

Table des matières

1	BACKEND - BASE DE DONNEES	3
1.1	POLYBAYDATABASE	3
1.2	MODELE / DAO	4
2	BACKEND - API	5
2.1	SERVEUR WEB	5
2.2	PREMIERE ROUTE	6
2.3	LISTE DES PRODUITS	6
3	FRONTEND	7
3.1	FETCH	7
3.2	VIEW	9
4	ENCHERES	10
4.1	TEST D'UNE API SANS FRONT	10
4.2	ROUTE PARAMETREE	13
4.3	ASSOCIER LE FRONT	14
4.4	SANS RECHARGEMENT COMPLET	15
5	SERVER SENT EVENT	16
5.1	FIN	16

Table des figures

1	PolyBayDatabase.java	3
2	App.java	3
3	Test PolyBayDatabase.java	3
4	Product.java	4
5	ProductDAO.java	4
6	App.java	5
7	Test ProductDAO.java	5
8	App.java	5
9	Test WebServer	5
10	ProductsController.java	6
11	App.java	6
12	Test ProductsController.java	6
13	ProductsController.java	7
14	Test ProductsController.java	7
15	Index.html	7
16	Products-service.js	8
17	Main.js	8
18	Test products-service.js	8
19	Index.html	9
20	Products-view.js	9
21	Main.js	10
22	Test products-view.js	10
23	ProductsController.java	10
24	ProductsDAO.java partie 1	11
25	ProductsDAO.java partie 2	12
26	ProductsDAO.java partie 3	13
27	App.java	13
28	Test api sans front	13
29	Test api sans front	13
30	Test route parametree	14
31	Méthode bid de products-service.js	14
32	Event Listener	14
33	Modification ProductsDAO	15
34	Modification ProductsController	15
35	Modification products-service	15
36	Modification products-view	15
37	Modification ProductsController	16
38	Ajout du dataset sur le bid	16
39	Méthode updateBid	16
40	Modification main	16

1 BACKEND - BASE DE DONNEES

L'ensemble du code est disponible sur ce [repo](#)

1.1 POLYBAYDATABASE

```
src > database > PolyBayDatabase.java > PolyBayDatabase
1  package database;
2
3  import java.sql.SQLException;
4
5  public class PolyBayDatabase extends MySQLDatabase{
6      public PolyBayDatabase() throws SQLException{
7          super(host:"localhost", port:3307, databaseName:"poly_bay", user:"mateo", password:"esirem");
8      }
9  }
```

FIGURE 1 – PolyBayDatabase.java

Pour initialiser la connexion via une instance de PolyBayDatabase, il suffit d'appeler le constructeur de MySQLDatabase en donnant directement les informations de connexion.

```
src > App.java > App
1  import java.sql.SQLException;
2
3  import database.PolyBayDatabase;
4
5  public class App {
6      Run | Debug
7      public static void main(String[] args) throws Exception {
8          try {
9              PolyBayDatabase poly_bay = new PolyBayDatabase();
10             } catch (SQLException e) {
11                 System.err.println(e.getMessage());
12             }
13     }
```

FIGURE 2 – App.java

Dans App.java il suffit ensuite de créer une instance de PolyBayDatabase sans oublier de l'importer.

```
(azymut@hp) - [~/Desktop/temp/TP_Web_DAW_FullStack/backend]
• $ /usr/bin/env /usr/lib/jvm/java-21-openjdk-amd64/bin/java @/tmp/cp_pg6n382f3zttfdqswnzgcpyl.argfile App
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
(azymut@hp) - [~/Desktop/temp/TP_Web_DAW_FullStack/backend]
o $
```

FIGURE 3 – Test PolyBayDatabase.java

Comme on peut le voir sur le test ci-dessus, la connexion fonctionne normalement et ne crée pas d'erreur.

1.2 MODELE / DAO

```
src > models > Product.java > ...
1  package models;
2
3  public record Product(
4      int id,
5      String name,
6      String owner,
7      float bid
8  );
```

FIGURE 4 – Product.java

Voici la structure du record Product qui possède un id, un name, un owner ainsi qu'un bid.

```
src > dao > ProductsDAO.java > ...
1  package dao;
2  import java.sql.PreparedStatement;
3  import java.sql.ResultSet;
4  import java.sql.SQLException;
5  import java.util.ArrayList;
6  import database.PolyBayDatabase;
7  import models.Product;
8  public class ProductsDAO {
9      public ProductsDAO(){
10
11      }
12      public ArrayList<Product> findAll(){
13          PolyBayDatabase poly_bay;
14          try {
15              poly_bay = new PolyBayDatabase();
16          } catch (SQLException e) {
17              System.err.println(e.getMessage());
18              poly_bay = null;
19          }
20          ArrayList<Product> products = new ArrayList<Product>();
21          String sqlquery = "SELECT * FROM `product` ORDER BY `name` ASC;";
22          PreparedStatement myPreparedStatement;
23          try {
24              myPreparedStatement = poly_bay.prepareStatement(sqlquery);
25          } catch (SQLException e) {
26              System.err.println("Impossible de préparer la requête:");
27              System.err.println(e.getMessage());
28              myPreparedStatement = null;
29          }
30          try {
31              ResultSet myResults = myPreparedStatement.executeQuery();
32              while(myResults.next()){
33                  int id = myResults.getInt("id");
34                  String name = myResults.getString("name");
35                  String owner = myResults.getString("owner");
36                  float bid = myResults.getFloat("bid");
37                  Product product = new Product(id, name, owner, bid);
38                  products.add(product);
39              }
40          } catch (SQLException e) {
41              System.err.println(e.getMessage());
42          }
43          return products;
44      }
45  }
```

FIGURE 5 – ProductDAO.java

La classe ProductDAO possède une méthode findAll qui va récupérer tous les produits présents dans la base de données et les triés par nom croissant. Pour cela, on crée une instance de PolyBayDatabase afin de pouvoir initialiser la connexion. On utilise un PreparedStatement qui prend notre requête en paramètre puis on récupère le résultat de la requête dans un ResultSet. Ensuite, pour chaque produit, on crée un Product que l'on ajoute au tableau des Products qui seront retournés par la fonction.

```

src > App.java > ...
1  import java.sql.SQLException;
2  import java.util.ArrayList;
3
4  import dao.ProductsDAO;
5  import models.Product;
6
7  public class App {
8      public static void main(String[] args) throws Exception {
9          ProductsDAO productDAO = new ProductsDAO();
10         ArrayList<Product> allProduct = productDAO.findAll();
11         System.out.println("Voici tous les produits de la BDD:");
12         for(int i = 0; i < allProduct.size(); i++){
13             System.out.println("Le produit ayant pour id: " + allProduct.get(i).id() + " est: " + allProduct.get(i).name() + " possédé par: "
14                               + allProduct.get(i).owner() + " et coute: " + allProduct.get(i).bid() + "€");
15         }
16     }
17 }

```

FIGURE 6 – App.java

Dans la classe App, on crée une instance de productDAO. On appelle la méthode findAll et on récupère le résultat dans le tableau allProduct. Pour chaque Product du tableau, on affiche les différentes informations et on obtient la sortie suivante :

```

(azymut@hp) - [~/Desktop/temp/TP_Web_DAW_FullStack/backend]
$ cd /home/azymut/Desktop/temp/TP_Web_DAW_FullStack/backend ; /usr/bin/env /usr/lib/jvm/
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Voici tous les produits de la BDD:
Le produit ayant pour id: 1 est: Bâton de Merlin possédé par: Francis et coute: 12450.0€
Le produit ayant pour id: 2 est: Capsule magique possédé par: Michel et coute: 13212.5€
Le produit ayant pour id: 3 est: RXT 4090 possédé par: Nicole et coute: 14793.6€
Le produit ayant pour id: 4 est: Transistor possédé par: Denis et coute: 13701.5€

```

FIGURE 7 – Test ProductDAO.java

On voit que l'on obtient bien toutes les informations sur chaque produit et que ceux-ci sont bien triés par nom croissant.

2 BACKEND - API

2.1 SERVEUR WEB

```

WebServer webserver = new WebServer();
webserver.listen(listeningPort:8080);

```

FIGURE 8 – App.java

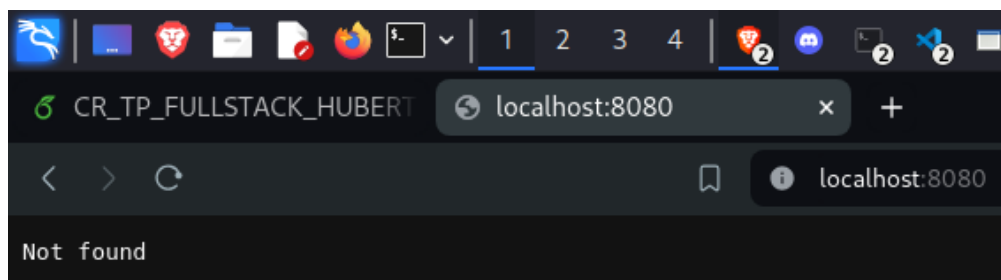


FIGURE 9 – Test WebServer

Grâce aux deux lignes précédentes, on a maintenant un serveur web sur le port 8080 fonctionnel qui retourne pour le moment Not found.

2.2 PREMIERE ROUTE

```
src > controllers > ProductsController.java > ProductsController
1  package controllers;
2  import java.util.ArrayList;
3  import models.Product;
4  import webserver.WebServerContext;
5  import webserver.WebServerResponse;
6
7  public class ProductsController {
8      public static ArrayList<Product> findAll(WebServerContext context){
9          WebServerResponse response = context.getResponse();
10         response.ok(message:"Tous les produits");
11         return null;
12     }
13 }
```

FIGURE 10 – ProductsController.java

Voici la méthode `findAll` de `ProductsController`. On récupère la réponse de `context` pour répondre à la request que tout s'est bien passé avec comme message "Tous les produits".

```
webserver.getRouter().get(path:"/products", (WebServerContext context) -> { ProductsController.findAll(context);});
```

FIGURE 11 – App.java

On crée maintenant notre première route. L'URL de notre route est `/products`. Si cette URL est contactée via la méthode GET alors la méthode à appeler est `findAll` de la classe `ProductsController` en donnant comme paramètre le `context` de contact. Si l'on se rend sur cette URL on obtient le résultat suivant :

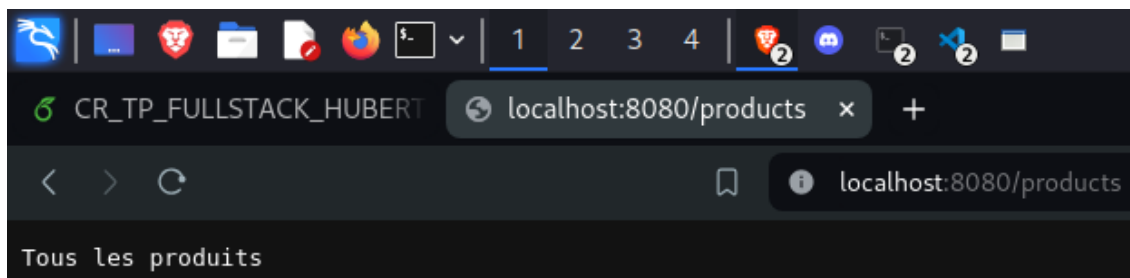


FIGURE 12 – Test ProductsController.java

Comme on peut le voir, la méthode `findAll` fonctionne correctement et on arrive à répondre aux demandes sur la route `/product`.

2.3 LISTE DES PRODUITS

On va maintenant essayer de modifier la méthode `findAll` pour afficher la liste de tous les produits présents sur la BDD. Pour cela, on va utiliser la méthode `json` de `WebServerResponse` comme suit :

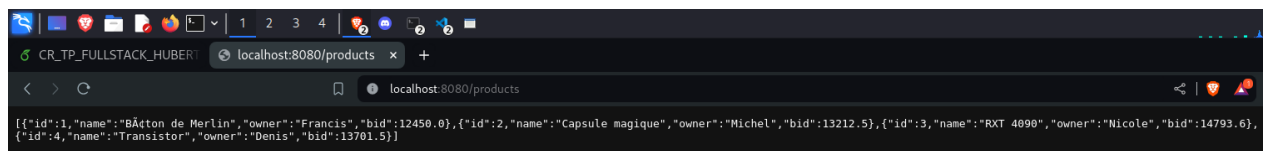
```

public class ProductsController {
    public static ArrayList<Product> findAll(WebServerContext context){
        WebServerResponse response = context.getResponse();
        ProductsDAO productDAO = new ProductsDAO();
        response.json(productDAO.findAll());
        return null;
    }
}

```

FIGURE 13 – ProductsController.java

Il suffit de créer un ProductsDAO puis d'appeler la méthode json de response en donnant en paramètre l'ensemble des produits récupérés dans la BDD via la méthode findAll de la classe ProductsDAO et on obtient ceci :



```

[{"id":1,"name":"Bâton de Merlin","owner":"Francis","bid":12450.0}, {"id":2,"name":"Capsule magique","owner":"Michel","bid":13212.5}, {"id":3,"name":"RXT 4090","owner":"Nicole","bid":14793.6}, {"id":4,"name":"Transistor","owner":"Denis","bid":13701.5}]

```

FIGURE 14 – Test ProductsController.java

3 FRONTEND

3.1 FETCH

```

index.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset="UTF-8">
5  |   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  |   <title>Document</title>
7  </head>
8  <body>
9  |   <script src="main.js" type="module"></script>
10 </body>
11 </html>

```

FIGURE 15 – Index.html

Voici le code de index.html généré via le snippet html :5.


```

services > JS products-service.js > ProductService
1  export class ProductService {
2      async findAll() {
3          const response = await fetch("http://localhost:8080/products");
4          if (response.status === 200) {
5              const data = await response.json();
6              return data;
7          }
8      }
9  }

```

FIGURE 16 – Products-service.js

Voici le code de la classe ProductService. La méthode findAll est asynchrone on peut donc utiliser await. Si la réponse reçue est de code 200, alors on récupère les données.

```

JS main.js > ...
1  import { ProductService } from "../services/products-service.js";
2
3  function run(){
4      const productService = new ProductService();
5      productService.findAll().then((data) => {
6          if (data) {
7              console.log("Tous les produits:", data);
8          }
9      }).catch(error => {
10         console.log("Aucun produit trouvé", error);
11     });
12 }
13
14 window.addEventListener("load", run);

```

FIGURE 17 – Main.js

Dans main.js on crée une fonction run qui crée une instance productService sur laquelle on appelle la méthode findAll. Si on obtient un résultat alors on affiche les données dans la console. Sinon, on affiche un message d'erreur. La fonction run est appelée lorsque la page est chargée. On obtient le résultat suivant :

```

Tous les produits: ▼ (4) [{...}, {...}, {...}, {...}] ⓘ
  ▶ 0: {id: 1, name: 'Bâton de Merlin', owner: 'Francis', bid: 12450}
  ▶ 1: {id: 2, name: 'Capsule magique', owner: 'Michel', bid: 13212.5}
  ▶ 2: {id: 3, name: 'RXT 4090', owner: 'Nicole', bid: 14793.6}
  ▶ 3: {id: 4, name: 'Transistor', owner: 'Denis', bid: 13701.5}
  length: 4
  ▶ [[Prototype]]: Array(0)

```

FIGURE 18 – Test products-service.js

On a bien tous les produits sous forme d'objets.

3.2 VIEW

```
<div class="products"></div>
```

FIGURE 19 – Index.html

```
views > products-view.js > ProductsView
1 import { ProductService } from "../services/products-service.js";
2
3 export class ProductsView {
4   constructor() {
5
6   }
7   displayProducts() {
8     const productService = new ProductService();
9     productService.findAll().then((data) => {
10       if (data) {
11         this.#displayProduct(data);
12       }
13     }).catch(error => {
14       console.log("Aucun produit trouvé", error);
15     });
16   }
17   #displayProduct(data) {
18     const products = document.querySelector(".products");
19     let name_div = document.createElement("div");
20     let owner_div = document.createElement("div");
21     let bid_div = document.createElement("div");
22     let button_div = document.createElement("div");
23     products.style = "display: flex; justify-content : space-between;";
24     let temp_enfant_name = document.createElement("div");
25     temp_enfant_name.style = "align-items: center;";
26     let temp_enfant_owner = document.createElement("div");
27     temp_enfant_owner.style = "align-items: center;";
28     let temp_enfant_bid = document.createElement("div");
29     temp_enfant_bid.style = "align-items: center;";
30     let temp_enfant_button = document.createElement("div");
31     temp_enfant_button.style = "display: grid;";
32     for(let i = 0; i < data.length; i++){
33       let temp_name = document.createElement("div");
34       temp_name.innerHTML = data[i].name;
35       let temp_owner = document.createElement("div");
36       temp_owner.innerHTML = data[i].owner;
37       let temp_bid = document.createElement("div");
38       temp_bid.innerHTML = data[i].bid;
39       let temp_button = document.createElement("button");
40       temp_button.innerHTML = "enchérir";
41       temp_enfant_name.appendChild(temp_name);
42       temp_enfant_owner.appendChild(temp_owner);
43       temp_enfant_bid.appendChild(temp_bid);
44       temp_enfant_button.appendChild(temp_button);
45     }
46     products.appendChild(temp_enfant_name);
47     products.appendChild(temp_enfant_owner);
48     products.appendChild(temp_enfant_bid);
49     products.appendChild(temp_enfant_button);
50   }
51 }
```

FIGURE 20 – Products-view.js

La méthode `displayProduct` récupère tous les produits de la base de données en paramètre. Pour chaque produit, on crée les différents "div" ou "button" nécessaire.

```

js main.js > ...
1  import { ProductService } from "../services/products-service.js";
2  import { ProductsView } from "../views/products-view.js";
3
4  function run(){
5      const productView = new ProductsView;
6      productView.displayProducts();
7  }
8
9  window.addEventListener("load", run);

```

FIGURE 21 – Main.js

Voici le fichier main.js modifié. On obtient ainsi l’affichage suivant :

Bâton de Merlin	Francis	12450	encherir
Capsule magique	Michel	13212.5	encherir
RXT 4090	Nicole	14793.6	encherir
Transistor	Denis	13701.5	encherir

FIGURE 22 – Test products-view.js

4 ENCHERES

4.1 TEST D’UNE API SANS FRONT

```

public static void bid(int id, WebServerContext context){
    ProductsDAO productDAO = new ProductsDAO();
    boolean result;
    try {
        result = productDAO.bid(id);
    } catch (Exception e) {
        System.err.println(e.getMessage());
        result = false;
    }
    WebServerResponse response = context.getResponse();
    if(result){
        response.ok(message:"Le prix à bien été incrémenté");
    }
    else{
        response.serverError(message:"Impossible de modifier le prix");
    }
}

```

FIGURE 23 – ProductsController.java

La méthode bid prend en paramètre l’id du produit à incrémenter ainsi que le context. On appelle la méthode bid de ProductDAO en passant en paramètre l’id. Si la requête aboutie, alors on renvoie une réponse ok sinon on envoie un serverError.

```

public boolean bid(int id){
    PolyBayDatabase poly_bay;
    boolean result;
    try {
        poly_bay = new PolyBayDatabase();
    } catch (SQLException e) {
        System.err.println(e.getMessage());
        poly_bay = null;
    }
    String query = "SELECT `bid` FROM `product` WHERE `id` = ?;";
    PreparedStatement effectue;
    try {
        effectue = poly_bay.prepareStatement(query);
    } catch (SQLException e) {
        System.err.println(e.getMessage());
        effectue = null;
    }
    try {
        effectue.setInt(1, id);
    } catch (SQLException e) {
        System.err.println(e.getMessage());
    }
    ResultSet myResult;
    try {
        myResult = effectue.executeQuery();
    } catch (SQLException e) {
        System.err.println(e.getMessage());
        myResult = null;
    }
    float bid_precedent = 0;
    try {
        while(myResult.next()){
            bid_precedent = myResult.getFloat("bid");
        }
    } catch (SQLException e) {
        System.err.println(e.getMessage());
        bid_precedent = 0;
    }
}

```

FIGURE 24 – ProductsDAO.java partie 1

Dans cette première partie de la méthode, on crée une instance de PolyBayDatabase qui nous servira par la suite pour les prepareStatement. On crée une requête qui va récupérer le bid du produit l'id passé en paramètre. Le but est de récupérer le prix avant la requête de mettre à jour le prix et de récupérer le prix après la requête. Si celui-ci a changé, alors la requête a fonctionné.

```

String sqlquery = "UPDATE `product` SET `bid` = bid + 50 WHERE `id` = ?;";

PreparedStatement myPreparedStatement;
try {
    myPreparedStatement = poly_bay.prepareStatement(sqlquery);
} catch (SQLException e) {
    System.err.println("Impossible de préparer la requête:");
    System.err.println(e.getMessage());
    myPreparedStatement = null;
}

try {
    myPreparedStatement.setInt(1, id);
} catch (SQLException e) {
    System.err.println(e.getMessage());
}

try {
    result = myPreparedStatement.execute();
} catch (SQLException e) {
    System.err.println(e.getMessage());
}

String query_after = "SELECT `bid` FROM `product` WHERE `id` = ?;";
PreparedStatement effectue_after;
try {
    effectue_after = poly_bay.prepareStatement(query_after);
} catch (SQLException e) {
    System.err.println(e.getMessage());
    effectue_after = null;
}

try {
    effectue_after.setInt(1, id);
} catch (SQLException e) {
    System.err.println(e.getMessage());
}

ResultSet myResult_after;
try {
    myResult_after = effectue_after.executeQuery();
} catch (SQLException e) {
    System.err.println(e.getMessage());
    myResult_after = null;
}

```

FIGURE 25 – ProductsDAO.java partie 2

```

float bid_after = 0;
try {
    while(myResult_after.next()){
        bid_after = myResult_after.getFloat("bid");
    }
} catch (SQLException e) {
    System.err.println(e.getMessage());
    bid_after = 0;
}
float requete_valide = bid_after - bid_precedent;
if(requete_valide == 50){
    result = true;
}
else{
    result = false;
}
return result;

```

FIGURE 26 – ProductsDAO.java partie 3

```

webserver.getRouter().post(path:"/bid", (WebServerContext context) -> { ProductsController.bid(id:1, context);});

```

FIGURE 27 – App.java

```

(azymut@hp)-[~]
$ curl -X POST http://localhost:8080/bid
Le prix à bien été incrémenté

```

FIGURE 28 – Test api sans front

Comme on peut le voir, si on fait une requête POST sur l'URL, ça incrémente bien le prix de 50 euros.

4.2 ROUTE PARAMETREE

```

WebServerRequest request = context.getRequest();
String parametre = request.getParam(key:"productId");
try {
    result = productDAO.bid(Integer.parseInt(parametre));
}

```

FIGURE 29 – Test api sans front

On modifie la fonction bid de ProductsController pour prendre le paramètre demandé en tant qu'id.

```

(azymut@hp)-[~]
$ curl -X POST http://localhost:8080/bid/6
Impossible de modifier le prix

(azymut@hp)-[~]
$ curl -X POST http://localhost:8080/bid/2
Le prix à bien été incrémenté

```

FIGURE 30 – Test route parametree

4.3 ASSOCIER LE FRONT

```

static async bid(id){
  const response = await fetch("http://localhost:8080/bid/"+id, {method: "POST"});
  if(response.status === 200){
    return true;
  }
  else{
    return false;
  }
}

```

FIGURE 31 – Méthode bid de products-service.js

```

temp_button.addEventListener("click", ()=>{
  const temp = document.querySelector("button");
  let id = temp_button.dataset.id;
  ProductsService.bid(id).then(()=>{
    location.reload();
  }).catch(error => {
    console.log("Vous ne pouvez pas enchérir");
  })
});

```

FIGURE 32 – Event Listener

Avec ces deux morceaux de code, lorsque l'on clique sur un bouton, la page se recharge et la valeur de bid correspondante se met à jour.

4.4 SANS RECHARGEMENT COMPLET

```
return Float.toString(bid_after);
```

FIGURE 33 – Modification ProductsDAO

On modifie la fonction méthode de ProductsDAO pour qu'elle retourne la valeur du bid sous la forme d'une string.

```
if(result != "null"){  
    String jsonString = "{\"bid\": \""+result+"\""}";  
    response.json(jsonString);  
    return true;  
}  
else{  
    response.serverError(message:"Impossible de modifier le prix");  
    return false;  
}
```

FIGURE 34 – Modification ProductsController

Dans la méthode bid de ProductsController, on crée une string qui prend la forme d'un objet JSON que l'on envoie en réponse.

```
const data2 = await response.json();  
return data2;
```

FIGURE 35 – Modification products-service

On modifie la méthode bid de products-service pour retourner les données reçues lors du contact de l'API.

```
temp_button.addEventListener("click", ()=>{  
    const id = temp_button.dataset.id;  
    ProductService.bid(id).then((data2)=>{  
        const current_temp_bid = temp_bid;  
        data2 = JSON.parse(data2);  
        current_temp_bid.innerHTML = data2.bid;  
    }).catch(error => {  
        console.log("Vous ne pouvez pas enchérir");  
    })  
});
```

FIGURE 36 – Modification products-view

On modifie la fonction appelée quand on clique sur un bouton pour récupérer les données et remplacer l'HTML de la div correspondante au bouton appuyer avec la nouvelle valeur du bid.

5 SERVER SENT EVENT

5.1 FIN

```
if(result != "null"){
    String jsonString = "{\"id\":\""+parametre+"\",\"bid\":\""+result+"\"}";
    context.getSSE().emit(channel:"bids", jsonString);
    response.ok(message:"Prix maj");
    return true;
}
```

FIGURE 37 – Modification ProductsController

On modifie la méthode bid de ProductsController pour renvoyer sur le channel l'id du prix modifié ainsi que le prix.

```
temp_bid.dataset.id = data[i].id;
```

FIGURE 38 – Ajout du dataset sur le bid

Pour pouvoir modifier le prix sur toutes les fenêtres sans souci, on ajoute un dataset similaire au bouton sur la div comportant le bid.

```
updateBid(data){
    const bid = document.querySelectorAll("div");
    for(let i = 0; i < bid.length; i++){
        if(bid[i].dataset.id == data.id){
            bid[i].innerHTML = data.bid + " €";
        }
    }
}
```

FIGURE 39 – Méthode updateBid

La méthode updateBid récupère toutes les divs et change le innerHTML uniquement si le dataset de la div correspond à l'id du produit modifié.

```
function run(){
    const productView = new ProductsView;
    const sseClient = new SSEClient("localhost:8080");
    sseClient.connect();
    sseClient.subscribe("bids", productView.updateBid);
    productView.displayProducts();
}
```

FIGURE 40 – Modification main

On modifie la fonction run de main.js pour utiliser les SSEClient et s'abonner au canal bids. Avec ce fonctionnement, il est possible de mettre à jour le prix sur toutes les fenêtres en même temps. Voici un lien pour une démonstration vidéo : [demo](#)