

**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

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

1 BACKEND - BASE DE DONNEES

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, dbName:"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     public static void main(String[] args) throws Exception {
7         try {
8             PolyBayDatabase poly bay = new PolyBayDatabase();
9         } catch (SQLException e) {
10             System.err.println(e.getMessage());
11         }
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]
$
```

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() + " posséd   par: "
14                               + allProduct.get(i).owner() + " et coute: " + allProduct.get(i).bid() + "  ");
15         }
16     }
17 }

```

FIGURE 6 – App.java

Dans la classe App, on cr  e une instance de productDAO. On appelle la m  thode findAll et on r  cup  re le r  sultat dans le tableau allProduct. Pour chaque Product du tableau, on affiche les diff  rentes 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: B  ton de Merlin poss  d   par: Francis et coute: 12450.0  
Le produit ayant pour id: 2 est: Capsule magique poss  d   par: Michel et coute: 13212.5  
Le produit ayant pour id: 3 est: RXT 4090 poss  d   par: Nicole et coute: 14793.6  
Le produit ayant pour id: 4 est: Transistor poss  d   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 tri  s par nom croissant.

2 BACKEND - API

2.1 SERVEUR WEB

```

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

```

FIGURE 8 – App.java

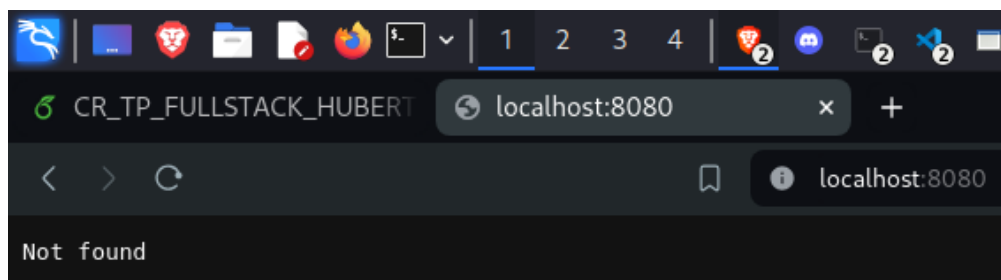


FIGURE 9 – Test WebServer

Gr  ce aux deux lignes pr  c  dentes, 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 :

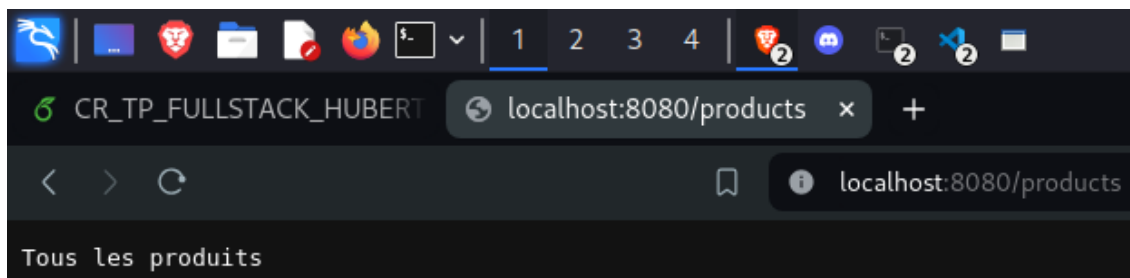


FIGURE 12 – Test ProductsController.java

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 :

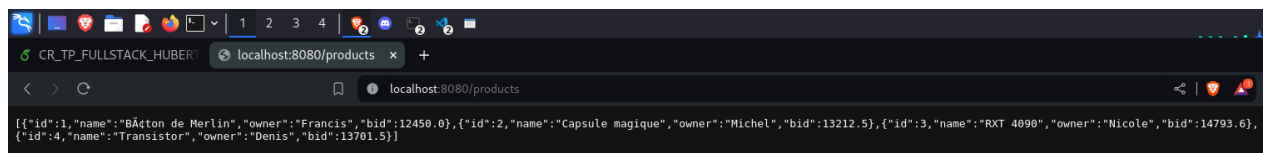
```

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