

Esirem Informatique & Réseaux

ITC-315 : CR Développement Web

DAW-FullStack

Auteur :
BOUQUILLON Erwan

Professeur référent :
MEUNIER Charles



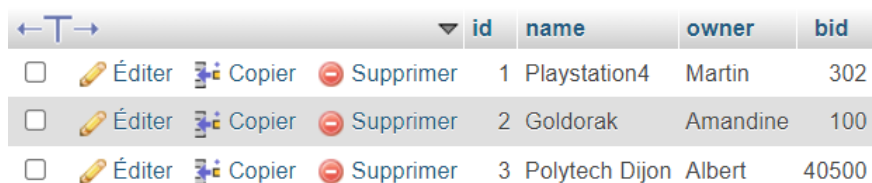
POLYTECH®
DIJON

1 DAW-FullStack

A travers cet exercice, on va créer une application de vente aux enchères dans une version wish de eBay. Parmi les points abordés :

- Créer un backend Java qui fournira une API Rest permettant d'interroger et de manipuler une base de données.
- Créer un frontend en HTML / CSS / JavaScript qui interagira avec votre API Rest dans une architecture micro-services.
- Mettre en place des Server Sent Events pour rendre votre application davantage réactive.

On commence donc par établir le service MySQL à l'aide de Xampp. Une fois les services misent en marche, on crée une nouvelle base de données poly_bay avec une table "product" qui contient un id, un nom (name), un propriétaire (owner) et une enchère (bid).



				id	name	owner	bid
<input type="checkbox"/>	Éditer	Copier	Supprimer	1	Playstation4	Martin	302
<input type="checkbox"/>	Éditer	Copier	Supprimer	2	Goldorak	Amandine	100
<input type="checkbox"/>	Éditer	Copier	Supprimer	3	Polytech Dijon	Albert	40500

FIGURE 1 – Ajout d'éléments à la base de donnée

1.1 Backend

Avant de se lancer dans la réalisation de ce TP, il nous faut encore créer le nouveau projet Java, récupérer le driver MySQL et récupérer le fichier MySQLDatabase.java pour le projet.

On va créer une classe PolyBayDatabase qui héritera de MySQLDatabase et qui initialisera les informations de connexion à la base de données. Pour vérifier son bon fonctionnement, on crée une nouvelle instance :

```
public class App {  
    Run | Debug  
    public static void main(String[] args) throws Exception {  
  
        PolyBayDatabase data = new PolyBayDatabase();  
    }  
}
```

FIGURE 2 – Nouvelle instance dans le main

En exécutant, on obtient 0 erreur, ce qui montre un bon déroulement de la classe instanciée.

```
PS C:\Users\verwan\Desktop\Fullstack\backend> cd "C:\Users\verwan\Desktop\Fullstack\backend"; & "C:\Program Files\Java\jdk-22\bin\java.exe" -gc:W  
sers\verwan\AppData\Local\Temp\cp_apege3c4c1hpjwtk1ybsep2qk.argfile" "App"  
PS C:\Users\verwan\Desktop\Fullstack\backend>
```

FIGURE 3 – Test de l'instance

1.1.1 Modèle

On vient créer un record Product qui permet de créer facilement des objets qui expriment une simple agrégation de valeurs. Voici ce que ça donne au moment de l'implémentation :

```

src > models > J Product.java > ...
1  package models;
2
3  public record Product(
4      int id,
5      String name,
6      String owner,
7      float bid
8  ){}
9

```

FIGURE 4 – Record

Les attributs d'un record sont immutables (ne peuvent pas être modifiés après avoir été initialisés) et les getters sont générés automatiquement par le compilateur Java.

1.1.2 DAO

On implémente la classe ProductsDAO avec la méthode findAll() qu'on a implémenté lors du TP précédent :

```

public ProductsDAO(){
    try {
        this.database = new PolyBayDatabase();
    }
    catch (SQLException e) {
        System.err.println(e.getMessage());
    }
}

public ArrayList<Product> findAll(){
    ArrayList<Product> produits = new ArrayList<Product>();
    String query = "SELECT * FROM product ORDER BY name ";

    try {
        PreparedStatement myStatement = database.prepareStatement(query);
        ResultSet results = myStatement.executeQuery();
        while(results.next()){
            final String owner = results.getString(columnLabel:"owner");
            final String name = results.getString(columnLabel:"name");
            final int id = results.getInt(columnLabel:"id");
            final int bid = results.getInt(columnLabel:"bid");

            Product produit = new Product(id, name, owner, bid);

            produits.add(produit);
        }
        return produits;
    }
    catch (SQLException e) {
        System.err.println(e.getMessage());
        return null;
    }
}

```

FIGURE 5 – Implémentation de ProductsDAO

On retourne dans le main, on supprime l'instance qu'on a créé précédemment et on le remplace par une instance de ProductsDAO.

```

public static void main(String[] args) throws Exception {

    //PolyBayDatabase data = new PolyBayDatabase();

    /* Test ProductsDAO findAll() */
    ProductsDAO dao = new ProductsDAO();
    System.out.println(dao.findAll());

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

```

FIGURE 6 – Ajout de la nouvelle instance

On exécute ensuite et on obtient :

```

PS C:\Users\erwan\Desktop\Fullstack\backend> cd "C:\Users\erwan\Desktop\Fullstack\backend"; & 'C:\Program Files\Java\jdk-22\bin\java.exe' -gc:\U
sers\erwan\AppData\Local\Temp\cp_clgrxet2e23asw136t0covvr-argfile' 'App'
[Product{id=2, name=Goldorak, owner=Amandine, bid=100.0}, Product{id=1, name=Playstation4, owner=Martin, bid=302.0}, Product{id=3, name=PolytechBijon
, owner=Albert, bid=49500.0}]

```

FIGURE 7 – Test de la méthode findAll()

La classe WebServer fournie encapsule la classe HTTPServer de Java pour simplifier la mise en place d'API et la gestion des Server Sent Event (SSE). Lorsqu'on ajoute une instance dans le main (voir ci-dessus), on obtient :

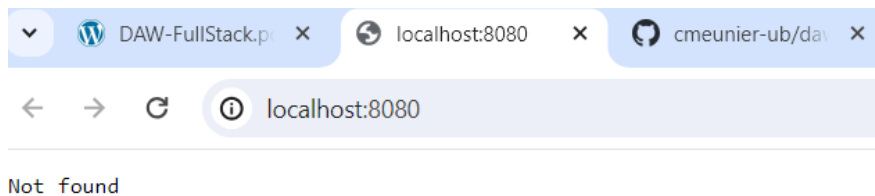


FIGURE 8 – Test de WebServer

1.1.3 Contrôleur

On implémente maintenant la classe ProductsController qui contient la méthode findAll. La méthode findAll appelle la méthode ok de WebServerResponse avec le message "Tous les produits".

```

src > controllers > ProductsController.java > ...
1  package controllers;
2
3  import java.util.ArrayList;
4
5  import dao.ProductsDAO;
6  import models.Product;
7  import webserver.WebServerContext;
8
9  public class ProductsController {
10
11      public static ArrayList<Product> findAll(WebServerContext serverContext){
12          ProductsDAO dao = new ProductsDAO();
13          serverContext.getResponse().ok(message:"Tous les produits :");
14          //serverContext.getResponse().json(dao.findAll());
15          return dao.findAll();
16      }
17  }
18

```

FIGURE 9 – Class ProductControlers

On obtient alors ;

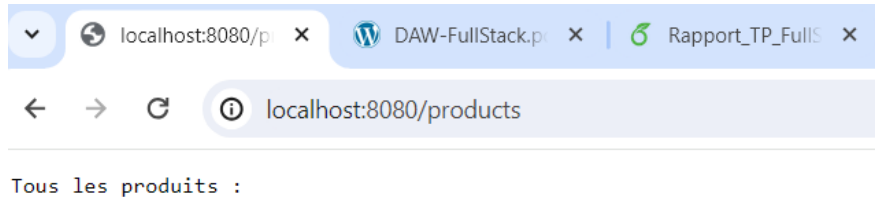


FIGURE 10 – Test du findAll()

1.1.4 Première route

La construction d'une route pour l'API nécessite trois éléments :

- l'URL de la route (/product).
- la méthode HTTP permettant d'y accéder (GET).
- la méthode à appeler pour réaliser.

Ici, la notation `() ->` permet de créer une fonction anonyme (plus exactement une expression lambda) comme on le ferait en JavaScript avec la notation `() =>`.

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

FIGURE 11 – Mise en place de la route

On obtient maintenant le même résultat que précédemment :

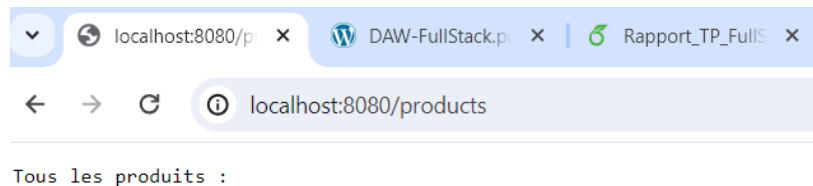


FIGURE 12 – Test de la route

1.1.5 Liste des produits

On modifie la méthode `findAll` pour qu'elle appelle la méthode `json` de `WebServerResponse` en lui passant la liste des produits présents dans la base de données, on obtient le code suivant :

```
src > controllers > ProductsController.java > ...
1  package controllers;
2
3  import java.util.ArrayList;
4
5  import dao.ProductsDAO;
6  import models.Product;
7  import webserver.WebServerContext;
8
9  public class ProductsController {
10
11     public static ArrayList<Product> findAll(WebServerContext serverContext){
12         ProductsDAO dao = new ProductsDAO();
13         //serverContext.getResponse().ok("Tous les produits :");
14         serverContext.getResponse().json(dao.findAll());
15         return dao.findAll();
16     }
17 }
18
```

FIGURE 13 – Class ProductControlers

Après exécution, on obtient :

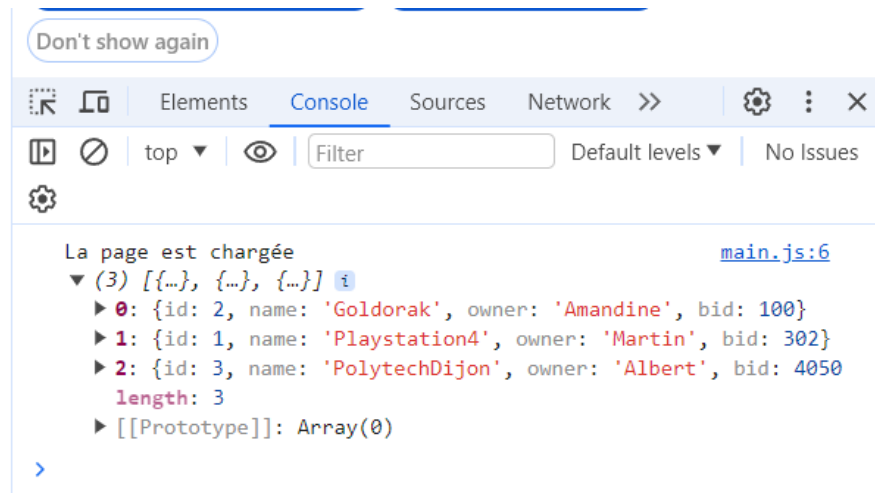


FIGURE 14 – Affichage de la console



FIGURE 15 – Test du listage des produits avec JSON

1.2 Frontend

De la même manière que précédemment, on cherche à créer un nouveau projet pour accueillir la partie frontend du ebay de wish.

1.2.1 Fetch

On ajoute un fichier products-service.js dans lequel on implémente la classe suivante :

```
services > JS products-service.js > ...
1  export class ProductService{
2
3      static async findAll(){
4          const response = await fetch("http://localhost:8080/products");
5          if(response.status === 200){
6              const data = await response.json();
7              return data;
8          }
9      }
10
11  }
12 }
```

FIGURE 16 – Implémentation de products-service

Pour pouvoir vérifier le bon fonctionnement, on implémente une fonction run dans le main pour afficher les résultats de la méthode findAll().

```
JS main.js > ...
1  import { ProductService } from "../services/products-service.js"
2  import { ProductsView } from "../views/products-view.js";
3
4  function run(){
5      ProductService.findAll().then((data) => {
6          if(data){
7              console.log("La page est chargée", data)
8              const prod = new ProductsView() ;
9              prod.displayProducts();
10         }
11     }).catch(error => {
12         console.log("Bon courage !", error)
13     });
14 }
15
16 window.addEventListener("load", run);
17
```

FIGURE 17 – Implémentation de la fonction run()

1.2.2 View

Afin de gérer la vue de notre ebay, on ajoute dans le fichier index.html, la ligne suivante :

```
9  <body>
10     <div class="products">
11
12     </div>
13     <script type="module" src="main.js"></script>
14 </body>
15 </html>
```

FIGURE 18 – Ajout de div et script dans le main

On implémente ensuite la classe suivante :

```
views > JS products-view.js > ...
1 import { ProductService } from "../services/products-service.js";
2
3 export class ProductsView{
4
5     constructor(){
6
7     }
8
9     static async displayProducts(){
10         const products = await ProductService.findAll();
11         products.forEach((product) => this.#displayProduct(product));
12     }
13
14     static #displayProduct(product){
15         const productsContainer = document.querySelector(".products");
16         const productElement = document.createElement("div");
17         productElement.classList.add("product");
18         productElement.innerHTML = `
19         <div class="product-line">
20             <p>
21                 <span>${product.name}</span>
22                 <span>${product.owner}</span>
23                 <span>${product.bid}</span></span>
24                 <button class="bid-button" data-id="${product.id}">Enchérir</button>
25             </p>
26         </div>`;
27         productsContainer.appendChild(productElement);
28     }
29 }
```

FIGURE 19 – Implémentation de ProductsView

On obtient l’affichage suivant, il ne respecte pas l’affichage que l’on souhaite obtenir mais je ne parviens pas à insérer des espaces entre chaque catégorie :

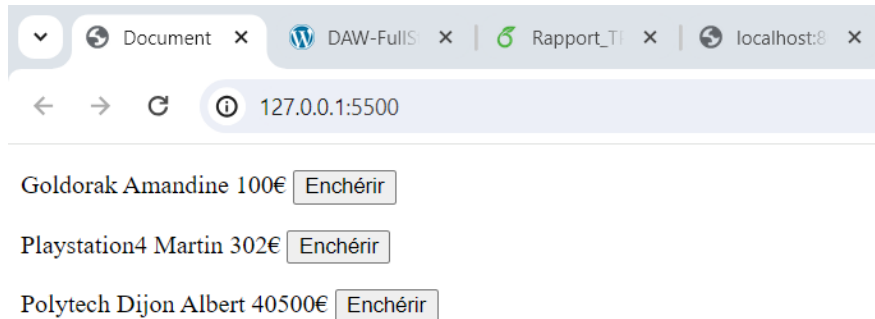


FIGURE 20 – Visualisation de l’affichage des produits

On retourne sur le backend de notre site web, on implémente la fonction bid qui permet d’ajouter 50 € à la somme initiale.

```
51 public boolean bid(int productId) throws SQLException {
52     String query = "UPDATE product SET bid = bid + 50 WHERE id = ?";
53     PreparedStatement preparedStatement = database.prepareStatement(query);
54     preparedStatement.setInt(parameterIndex:1, productId);
55     return preparedStatement.executeUpdate() > 0;
56 }
57 }
```

FIGURE 21 – Implémentation de bid dans ProductsDAO