

**Institut National des Postes et Télécommunications**

# **Rapport de Travaux Pratiques**

TP1 : Les Servlets HTTP et Architecture JEE

**Réalisé par :**

Youssef Dadda  
Akram El Mamoun  
Asmaa Bagri

20 novembre 2025

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Mise en place et Configuration</b>	<b>2</b>
<b>3</b>	<b>Extension 1 : Technologie JSP (Vue)</b>	<b>2</b>
<b>4</b>	<b>Extension 2 : Filtrage et Sécurité</b>	<b>3</b>
<b>5</b>	<b>Extension 3 : Persistance avec JDBC et Docker</b>	<b>4</b>
5.1	Architecture Mise en Place . . . . .	4
5.2	Implémentation et Résultats . . . . .	5
<b>6</b>	<b>Conclusion</b>	<b>6</b>

## 1 Introduction

L'objectif de ce TP est de manipuler le protocole HTTP en utilisant la plateforme JEE, spécifiquement les Servlets. Nous avons commencé par la configuration de l'environnement (Eclipse, Tomcat), pour ensuite développer une application web dynamique. Enfin, nous avons réalisé plusieurs extensions techniques majeures : la séparation Vue/Contrôleur avec JSP, la sécurisation via des Filtres, et la persistance des données via JDBC et Docker.

## 2 Mise en place et Configuration

La première étape a consisté à configurer un projet "Dynamic Web Project" sous Eclipse avec le serveur Apache Tomcat 9.0. Nous avons rencontré et résolu des problèmes liés au *Deployment Assembly* pour assurer que les classes Java soient bien déployées dans le dossier WEB-INF/classes.

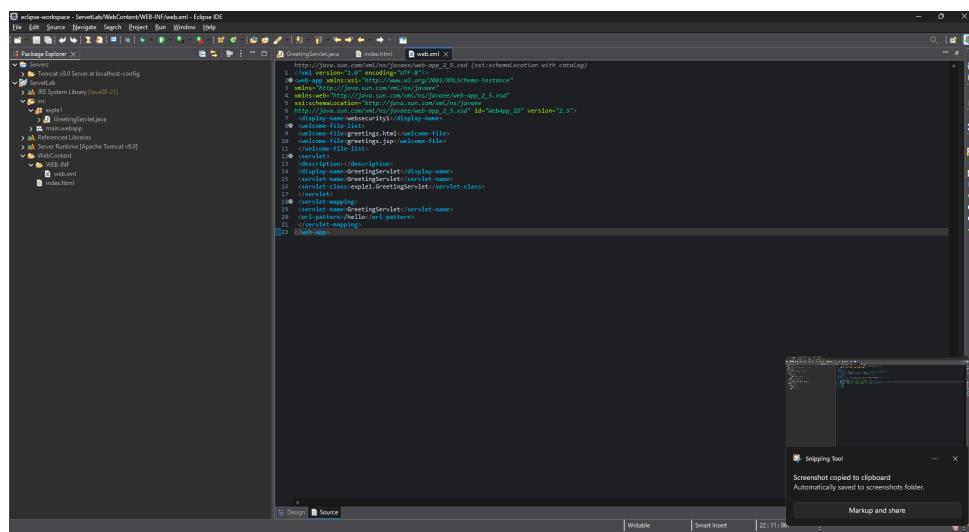


FIGURE 1 – Configuration du descripteur de déploiement web.xml (Version 2.5)

### 3 Extension 1 : Technologie JSP (Vue)

Pour respecter le modèle MVC (Modèle-Vue-Contrôleur), nous avons remplacé la génération de HTML via `out.println` dans la Servlet par une redirection vers une page JSP.

Nous avons dû assurer la correspondance exacte des clés entre `request.setAttribute` dans la Servlet et `request.getAttribute` dans la JSP. La figure ci-dessous montre une phase de débogage où les clés ne correspondaient pas (valeur `null`), corrigée par la suite.

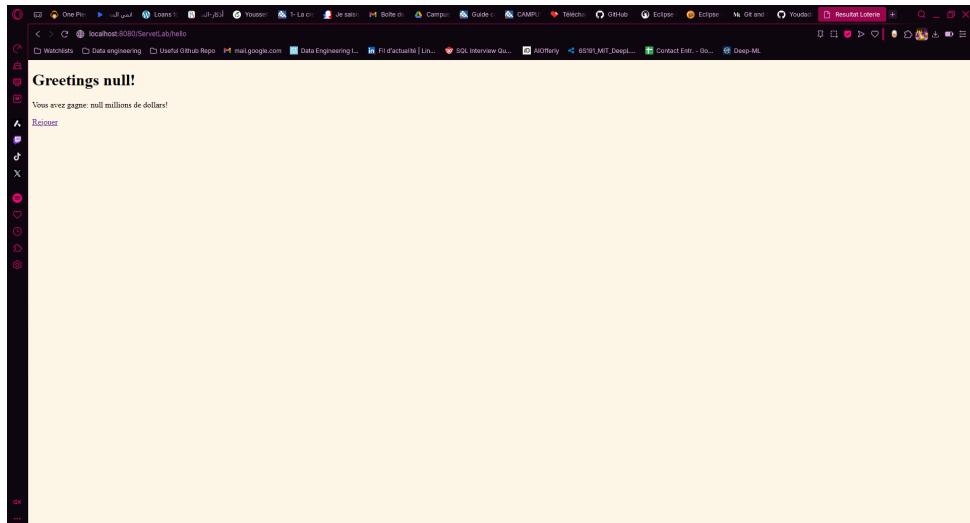


FIGURE 2 – Débogage de la transmission des attributs vers la JSP

Une fois corrigé, l'affichage dynamique fonctionne parfaitement :

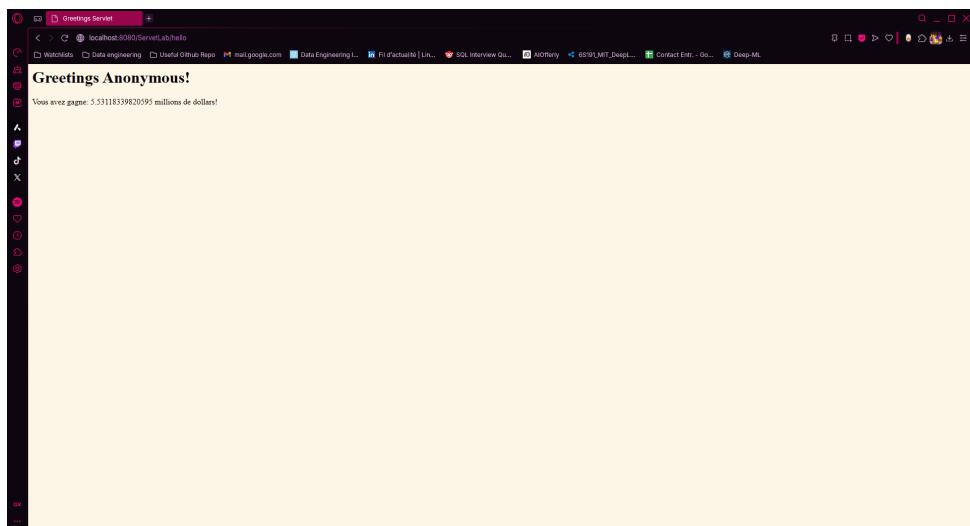


FIGURE 3 – Affichage réussi du résultat via la JSP

## 4 Extension 2 : Filtrage et Sécurité

Nous avons implémenté un `javax.servlet.Filter` pour intercepter les requêtes avant qu'elles n'atteignent la Servlet. L'objectif était de mettre en place une "Blacklist". Si un utilisateur tente de se connecter avec le nom "hacker", l'accès lui est refusé.



FIGURE 4 – Tentative d'intrusion avec le nom "hacker"

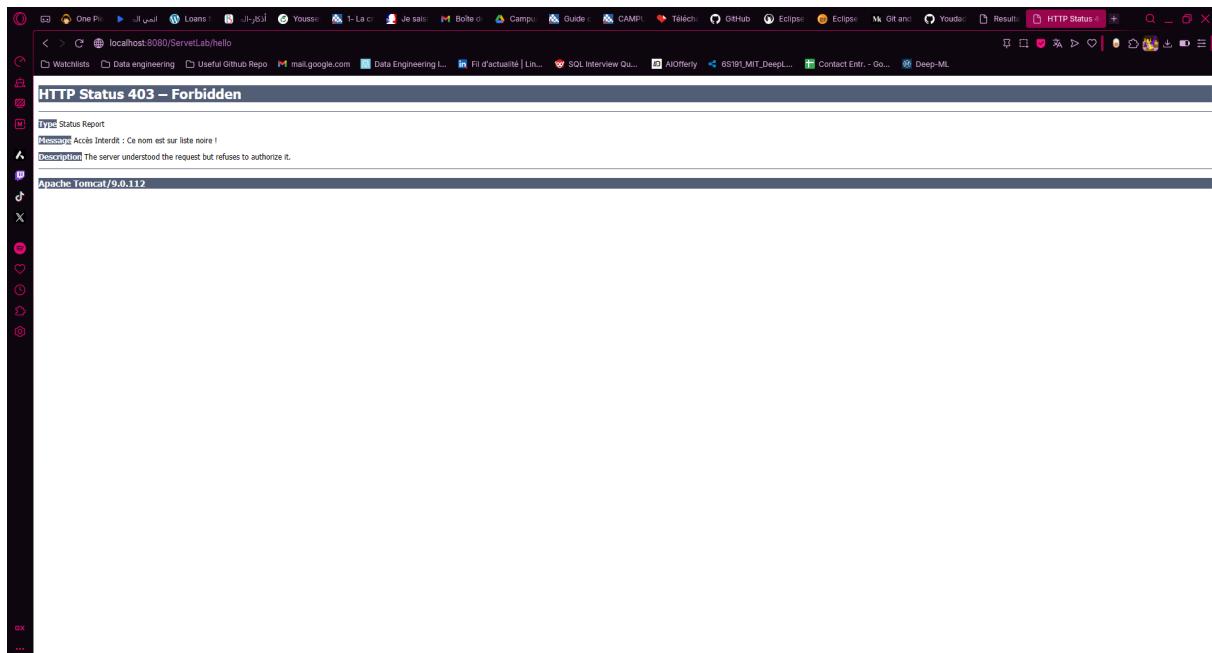


FIGURE 5 – Interception par le Filtre et renvoi d'une erreur 403 Forbidden

## 5 Extension 3 : Persistance avec JDBC et Docker

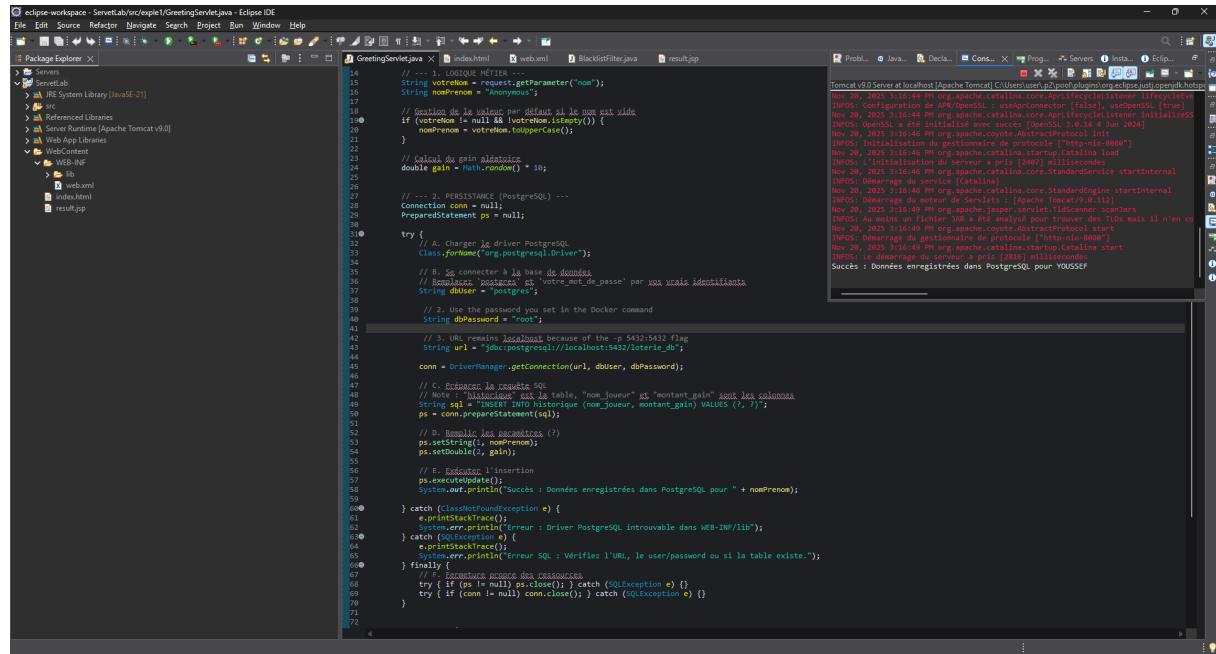
Dans le cadre de notre spécialisation Data Engineering, nous avons choisi d'utiliser \*\*PostgreSQL\*\* conteneurisé via \*\*Docker\*\* au lieu d'une installation locale MySQL standard.

### 5.1 Architecture Mise en Place

- **SGBD :** PostgreSQL (Image Docker officielle)
- **Connecteur :** Pilote JDBC PostgreSQL ajouté au WEB-INF/lib
- **Code :** Utilisation de PreparedStatement pour éviter les injections SQL.

## 5.2 Implémentation et Résultats

Le code de la Servlet a été modifié pour ouvrir une connexion JDBC, insérer les données (Nom et Gain), puis fermer la connexion avant de rediriger vers la JSP.



The screenshot shows the Eclipse IDE interface with the GreetingServlet.java file open in the editor. The code implements a servlet that connects to a PostgreSQL database to insert data into a 'historique' table. The Tomcat 9.0 Server log window is also visible, showing successful deployment and startup messages, including the insertion of a new record into the database.

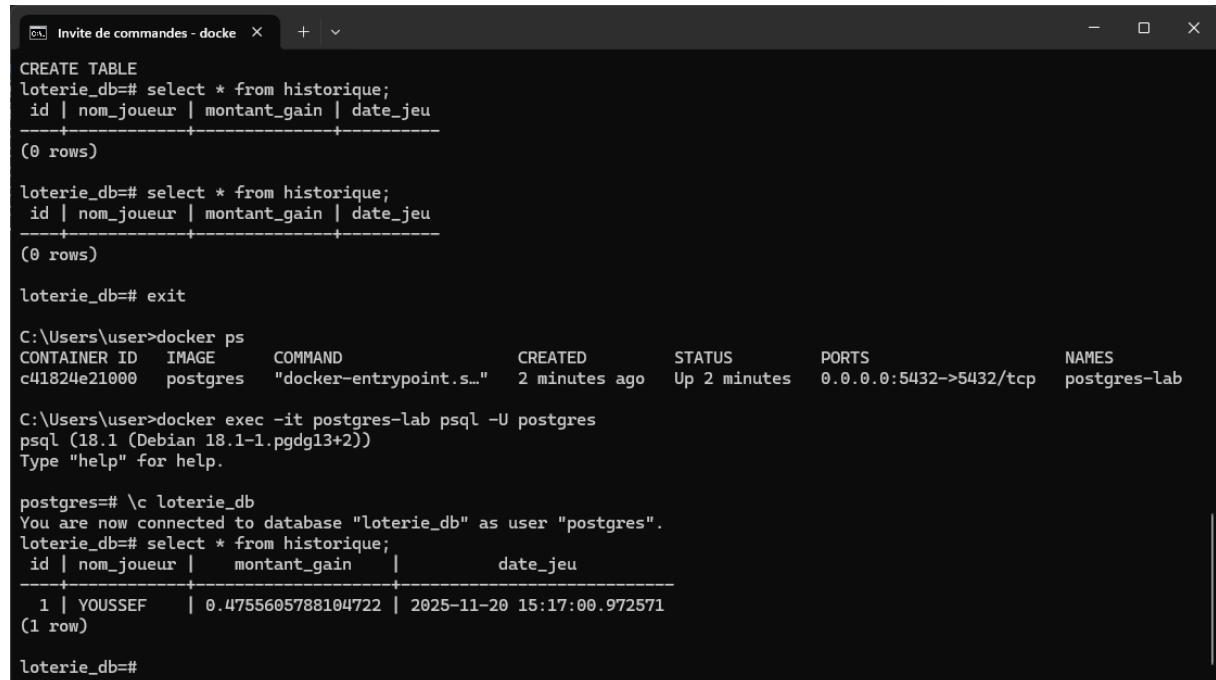
```

14     // --- L. LOTERIE NÉTIER ---
15     String votrenom = request.getParameter("nom");
16     String nomPrenom = "Anonymous";
17
18     // Gestion de la valeur par défaut si le nom est vide
19     if (votrenom == null && votrenom.isEmpty()) {
20         nomPrenom = votrenom.toUpperCase();
21     }
22
23     // Calcul du gain aléatoire
24     double gain = Math.random() * 10;
25
26     // --- 2. PERSISTANCE (PostgreSQL) ---
27     Connection conn = null;
28     PreparedStatement ps = null;
29
30     try {
31         // A. Charger le driver PostgreSQL
32         Class.forName("org.postgresql.Driver");
33
34         // B. Se connecter à la base de données
35         // Remplacez 'postgresql://votre_nom_de_passe' par vos vrais identifiants
36         String dbUser = "postgres";
37
38         // C. Utilisez le password que vous avez set dans le Docker command
39         String dbPassword = "root";
40
41         // D. URL remains localhost because of the -p 5432:5432 flag
42         String url = "jdbc:postgresql://localhost:5432/loterie_db";
43
44         conn = DriverManager.getConnection(url, dbUser, dbPassword);
45
46         // E. Exécutez la requête SQL
47         // Notez que l'insertion est dans la table "nom_joueur" et "montant_gain" sont les colonnes
48         String sql = "INSERT INTO historique (nom_joueur, montant_gain) VALUES (?, ?)";
49         ps = conn.prepareStatement(sql);
50
51         // F. Remplissez les paramètres ?
52         ps.setString(1, nomPrenom);
53         ps.setDouble(2, gain);
54
55         // G. Exécutez l'insertion
56         ps.executeUpdate();
57         System.out.println("Success : Données enregistrées dans PostgreSQL pour " + nomPrenom);
58
59     } catch (ClassNotFoundException e) {
60         e.printStackTrace();
61     } catch (SQLException e) {
62         System.err.println("Erreur : Driver PostgreSQL introuvable dans WEB-INF/lib/");
63     } catch (Exception e) {
64         e.printStackTrace();
65     } finally {
66         // H. Fermeture des ressources
67         try { if (ps != null) ps.close(); } catch (SQLException e) {}
68         try { if (conn != null) conn.close(); } catch (SQLException e) {}
69     }
70 }
71
72

```

FIGURE 6 – Code JDBC final et logs console confirmant l’insertion

Nous avons vérifié la persistance des données directement à l’intérieur du conteneur Docker via la commande `psql`.



The terminal window shows several commands run against a PostgreSQL database named 'loterie\_db'. It first creates a table 'historique' with columns id, nom\_joueur, montant\_gain, and date\_jeu. Then it runs two 'select \* from historique' queries, both of which return zero rows. Finally, it exits the session. Below the terminal, a list of running Docker containers is shown, and then the user runs 'docker exec -it postgres-lab psql -U postgres' to connect to the PostgreSQL container directly. Inside the container, a 'psql' session is started, and the user connects to the 'loterie\_db' database. They run the same 'select \* from historique' query again, and this time it returns a single row with the data inserted earlier.

```

CREATE TABLE
loterie_db=# select * from historique;
 id | nom_joueur | montant_gain | date_jeu
----+-----+-----+-----+
(0 rows)

loterie_db=# select * from historique;
 id | nom_joueur | montant_gain | date_jeu
----+-----+-----+-----+
(0 rows)

loterie_db=# exit

C:\Users\user>docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
c41824e21000        postgres            "docker-entrypoint.s..."   2 minutes ago      Up 2 minutes       0.0.0.0:5432->5432/tcp   postgres-lab

C:\Users\user>docker exec -it postgres-lab psql -U postgres
psql (18.1 (Debian 18.1-1.pgdg13+2))
Type "help" for help.

postgres=# \c loterie_db
You are now connected to database "loterie_db" as user "postgres".
loterie_db=# select * from historique;
 id | nom_joueur | montant_gain | date_jeu
----+-----+-----+-----+
 1 | YOUSSEF    | 0.4755605788104722 | 2025-11-20 15:17:00.972571
(1 row)

loterie_db=#

```

FIGURE 7 – Vérification des données dans PostgreSQL via CLI Docker

## 6 Conclusion

Ce TP nous a permis de maîtriser les fondamentaux du développement JEE. Nous avons appris à configurer un serveur d'application, à séparer la logique métier de la présentation (JSP), à sécuriser l'application via des filtres, et enfin à intégrer une base de données relationnelle moderne (PostgreSQL) dans un environnement conteneurisé, simulant une architecture de production réelle.