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

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`.

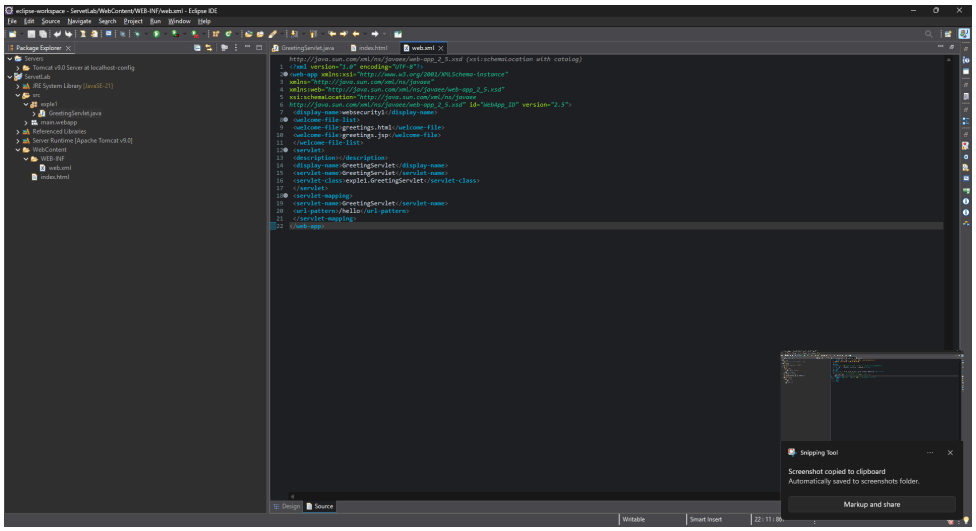

FIGURE 1 – Configuration du descripteur de déploiement web.xml (Version 2.5)

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.

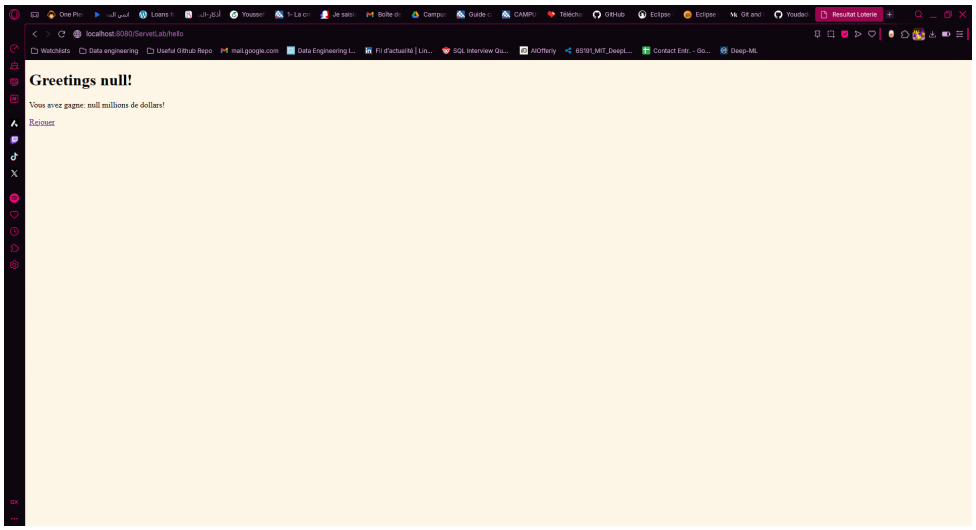

FIGURE 2 – Débogage de la transmission des attributs vers la JSP

Une fois corrigé, l'affichage dynamique fonctionne parfaitement :

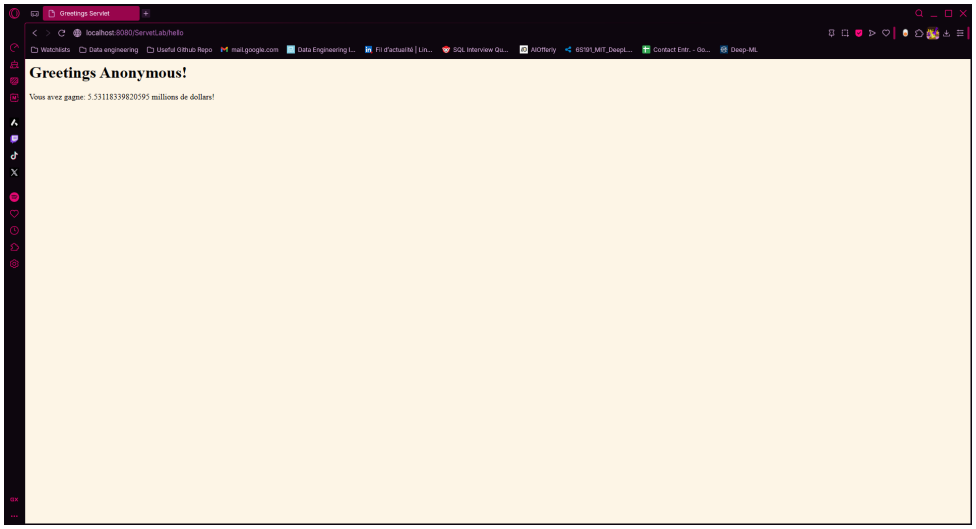

FIGURE 3 – Affichage réussi du résultat via la JSP

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.

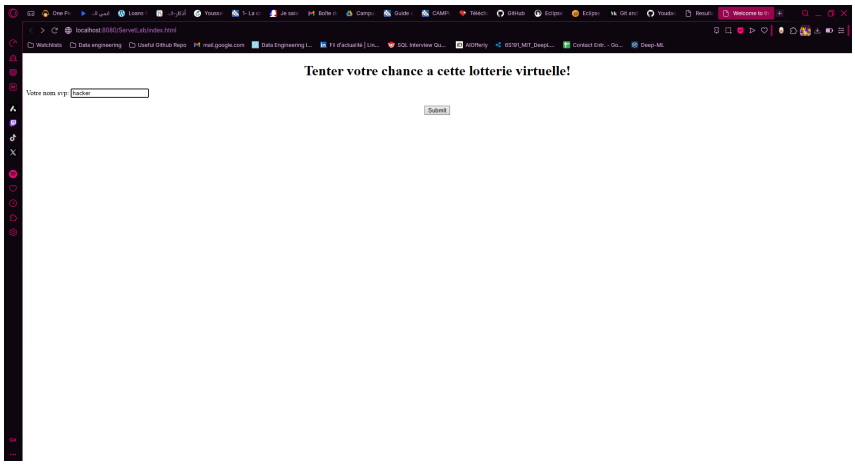

FIGURE 4 – Tentative d'intrusion avec le nom "hacker"

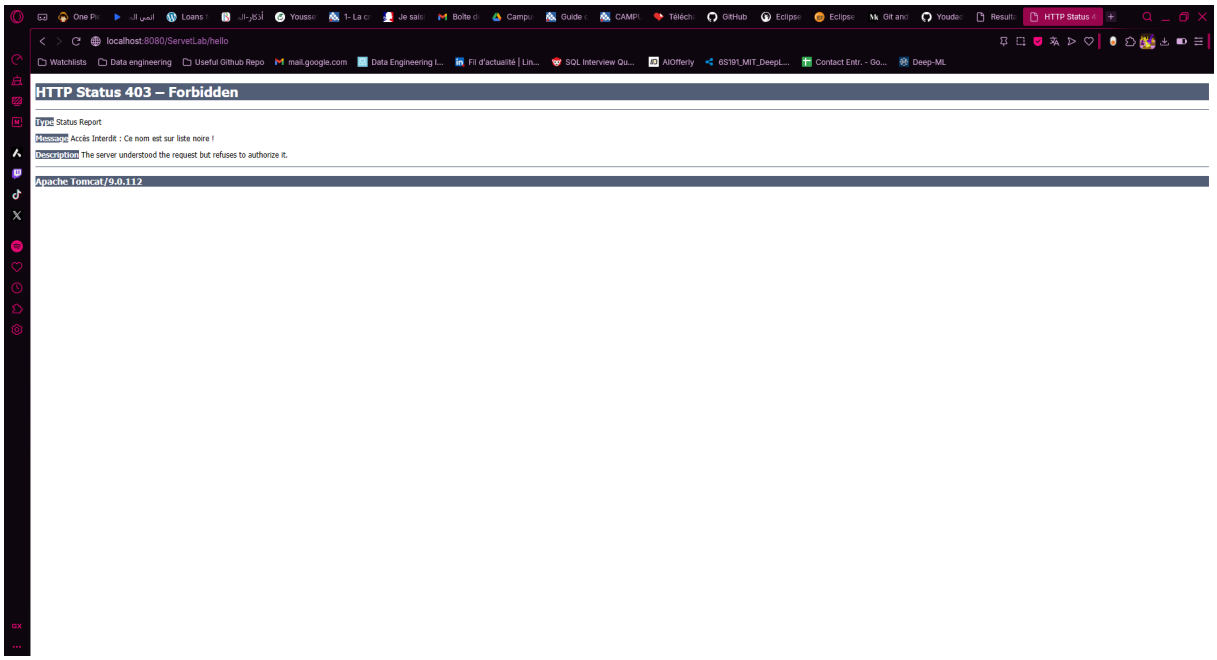

FIGURE 5 – Interception par le Filtre et renvoi d'une erreur 403 Forbidden

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.

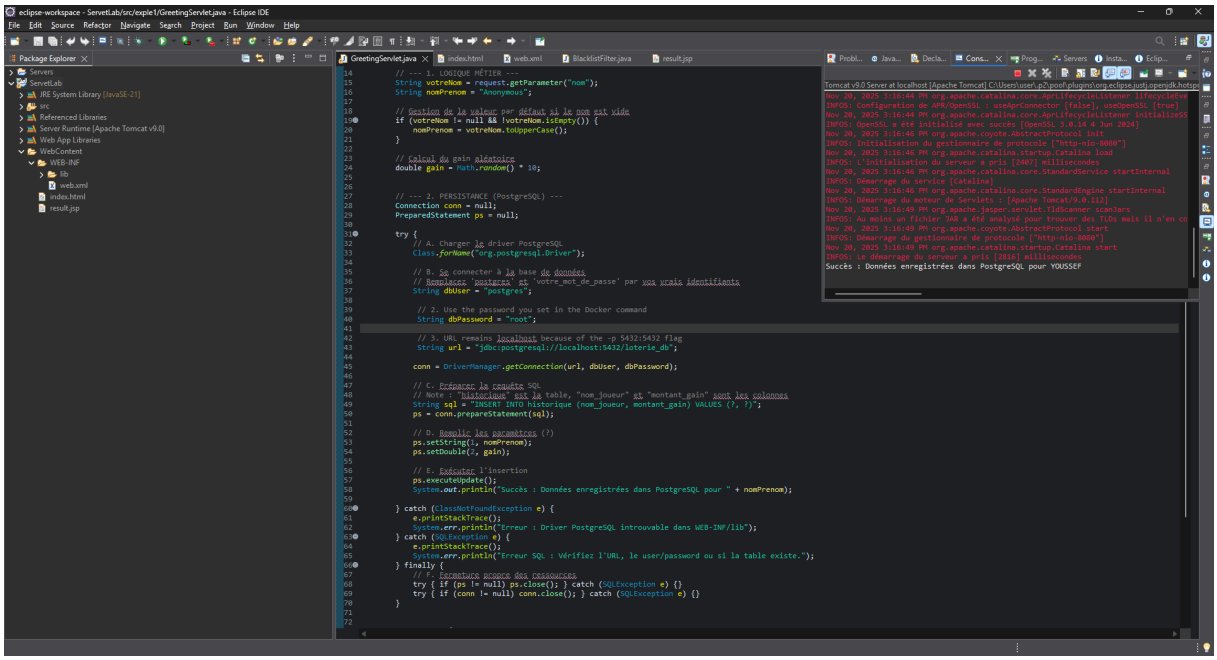

FIGURE 6 – Code JDBC final et logs console confirmant l'insertion

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

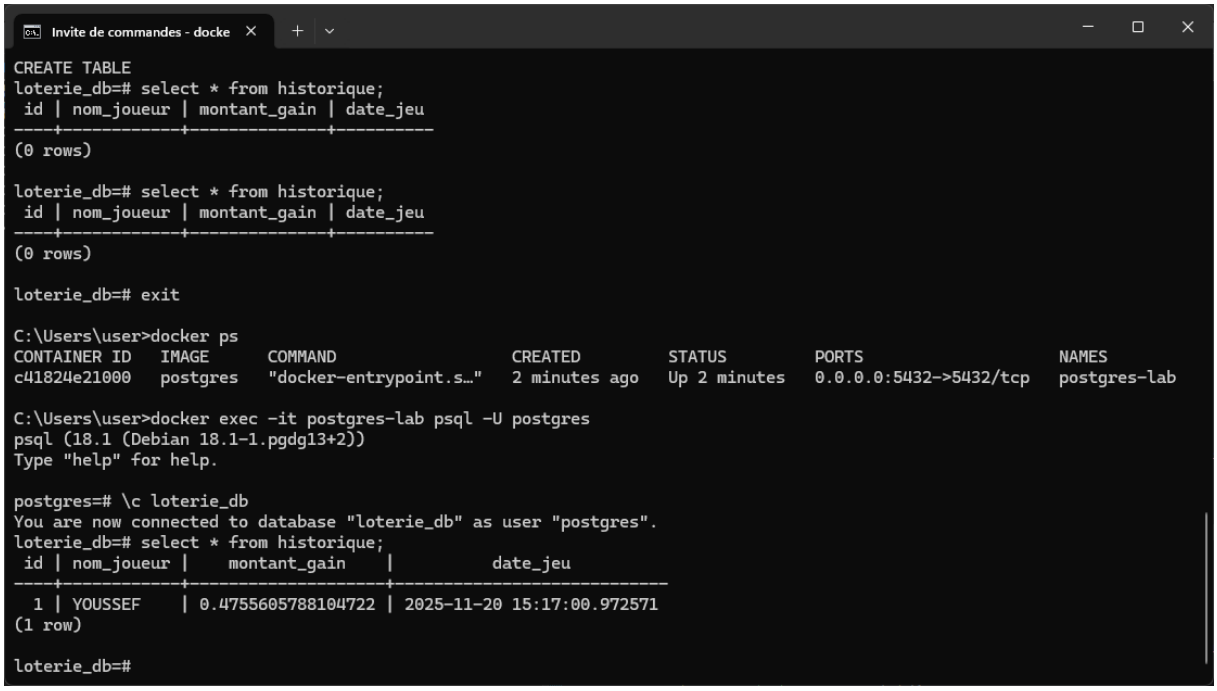

FIGURE 7 – Vérification des données dans PostgreSQL via CLI Docker

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.