Module : Compétence 2 - Optimisation des applications
Projet : Suivi de Colis IUT

Rapport Optimisation et Sécurité Algorithme

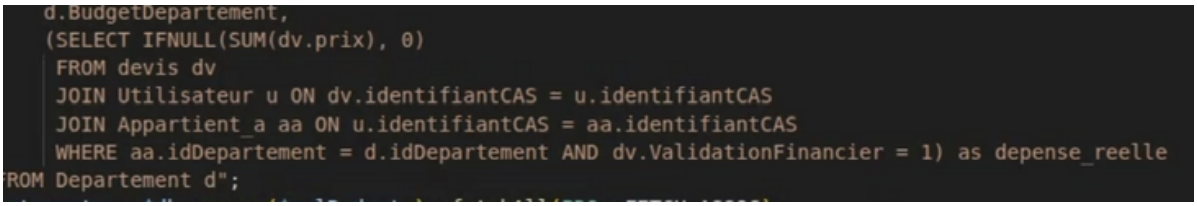
OMAR MSA , JASPER PETIT , LUKA JEVTIC , RIYAD LAROUB , HELA ADDAR

Introduction

Notre projet consiste en la réalisation d'une application web complète de suivi de colis au sein de l'IUT. Afin de répondre aux besoins spécifiques de chaque acteur, nous avons divisé l'application en quatre modules interconnectés : le Service Postal (expédition), le Service Financier (validation budgétaire), le Demandeur (départements/commandes) et l'Administrateur (gestion globale). L'objectif de cet audit est de justifier nos choix algorithmiques visant à garantir une application performante, capable de traiter un grand volume de données tout en restant robuste face aux failles de sécurité.

Choix d'Optimisation des Algorithmes

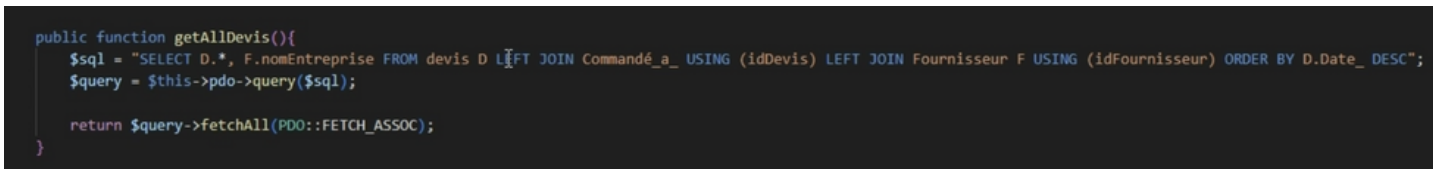
Notre défi principal était de traiter efficacement les historiques de commandes et les calculs budgétaires sans ralentir le serveur. Nous avons appliqué une stratégie consistant à déléguer le traitement lourd à la base de données. Pour le module Financier, plutôt que d'utiliser des boucles PHP coûteuses en mémoire pour additionner les dépenses, le calcul des budgets restants est effectué directement via la fonction d'agrégation SUM() dans la requête SQL. Ainsi, c'est le moteur de base de données qui effectue le calcul mathématique, PHP ne recevant que le résultat final.



```
    d.BudgetDepartement,
    (SELECT IFNULL(SUM(dv.prix), 0)
     FROM devis dv
     JOIN Utilisateur u ON dv.identifiantCAS = u.identifiantCAS
     JOIN Appartient_a aa ON u.identifiantCAS = aa.identifiantCAS
     WHERE aa.idDepartement = d.idDepartement AND dv.ValidationFinancier = 1) as depense_reelle
FROM Departement d";
```

Optimisation des calculs via agrégation SQL.

Dans cette même logique de réduction de la charge serveur, nous avons optimisé la gestion des données relationnelles sur l'ensemble des modules. Nous avons banni les requêtes multiples à l'intérieur de boucles et privilégions l'utilisation de LEFT JOIN. Cette technique nous permet de récupérer en une seule requête toutes les données liées, comme la commande, le client et le fournisseur, évitant ainsi de multiplier les allers-retours réseau. Bien que cela créer des requetes sql plus grandes et augmente ainsi le temps de traitement d'une requete, cela nous fait gagner du temps au global.



```
public function getAllDevis(){
    $sql = "SELECT D.*, F.nomEntreprise FROM devis D LEFT JOIN Commandé_a_ USING (idDevis) LEFT JOIN Fournisseur F USING (idFournisseur) ORDER BY D.Date_ DESC";
    $query = $this->pdo->query($sql);

    return $query->fetchAll(PDO::FETCH_ASSOC);
}
```

Utilisation de LEFT JOIN pour récupérer toutes les données en une seule requête.

Concernant la recherche de colis, nous avons également mis en œuvre une optimisation côté client pour améliorer l'expérience utilisateur et alléger le travaille du serveur. Pour éviter de recharger la page à chaque filtre, nous utilisons JavaScript. Les métadonnées des commandes sont stockées dans des attributs HTML5 data-*, ce qui permet à un script de filtrer et trier le DOM instantanément sans solliciter le serveur.

Filtrage instantané côté client via JavaScript pour réduire la charge serveur.

```
<script>
    function filtrerCommandes() {
        const inputRecherche = document.getElementById("searchBar");
        const filtreRecherche = inputRecherche.value.toUpperCase();

        const filtreDate = document.getElementById("filtreDate").value;
        const filtreStatut = document.getElementById("filtreStatut").value;
        const filtreConfirmation = document.getElementById("filtreConfirmation").value;

        const container = document.getElementById("listeCommandes");
        const cards = Array.from(container.getElementsByClassName("commande-card"));

        // Étape 1 : Filtrage
        const cardsVisibles = cards.filter(function(card) {
            const h3 = card.getElementsByTagName("h3")[0];
            const txtValue = h3.textContent || h3.innerText;
            const matchRecherche = txtValue.toUpperCase().indexOf(filtreRecherche) > -1;

            const statut = card.getAttribute("data-statut");
            const matchStatut = !filtreStatut || statut === filtreStatut;

            const confirmation = card.getAttribute("data-confirmation");
            const matchConfirmation = !filtreConfirmation || confirmation === filtreConfirmation;

            return matchRecherche && matchStatut && matchConfirmation;
        });
```

Audit de sécurité

La sécurité étant une priorité absolue, notamment pour les modules financiers, nous avons mis en place une défense en profondeur. Le risque critique d'injection SQL est neutralisé par l'utilisation systématique de l'objet PDO ainsi que des requêtes préparées sur tous les modules. Nous renforçons cette protection par un typage strict des données ; par exemple, dans le module Postal, l'identifiant client est explicitement typé en SQLITE3_INTEGER, ce qui bloque automatiquement toute tentative d'injection de texte malveillant

```
$preparation = $this->connexion_bdd->prepare($requete_sql);

$preparation->bindValue(':cas', $identifiant_cas, SQLITE3_INTEGER);
```

Typage strict des paramètres SQL (SQLITE3_INTEGER) pour bloquer les injections.

De plus, nous protégeons également l'affichage. Nous utilisons la fonction htmlspecialchars() partout. Ça empêche les assaillant d'injecter du code JavaScript (faille XSS) dans nos pages en le traitant comme des chaines de caractères. Et pour finir, nous avons bien configuré la base de données pour qu'on ne puisse pas créer de données "fantômes" (grâce aux clés étrangères) en forcant un colis a etre ratyaché a une commande, au quel cas une erreur s'affichera..

```
<p><i class="fas fa-envelope"></i> <?= htmlspecialchars($fournisseur['Mail']) ?></p>
<p><i class="fas fa-phone"></i> <?= htmlspecialchars($fournisseur['NumeroTelephone']) ?></p>
<p><i class="fas fa-map-marker-alt"></i> <?= htmlspecialchars($fournisseur['adresse'] ?? 'Adresse non renseignée') ?></p>
<p><i class="fa-solid fa-box"></i> <?= $fournisseur['nb_commandes'] ?> commandes passées</p>
```

Protection contre les failles XSS : on filtre tout ce qui s'affiche avec htmlspecialchars().

Pour conclure notre application est fiable, respectant plusieurs règles de sécurité essentiel tel que l'injection SQL et XSS. Nous avons également réfléchi a plusieurs moyen d'optimiser notre site afin d'avoir la meilleur expérience utilisateur et performance que l'on pouvait produire.