BTS SIO — Option SLAM

RÉFACTORING ET SÉCURISATION D'UN APPLICATIF LEGACY

Analyse corrective de l'application AppliFrais en PHP brut et transition vers architecture moderne

Contexte Étude	Laboratoire GSB — Application AppliFrais v1
Période	Décembre 2025
Technologies	PHP Structuré (Legacy) / MySQLi / Python (Cryptographie / MD5)
Candidat	COUR Cyrille

COUR Cyrille – cyrillecour11@gmail.com – Aristée 2025/2027

SOMMAIRE

1. Contexte Applicatif et Problématique de l'Existant (Code Legacy)

Lors de ma première année de BTS SIO option SLAM, j'ai été amené à manipuler l'application AppliFrais de l'entreprise fictive GSB (Galaxy Swiss Bourdin). Cette application, conçue initialement pour centraliser les notes de frais des visiteurs médicaux, m'a été livrée dans un état non fonctionnel.

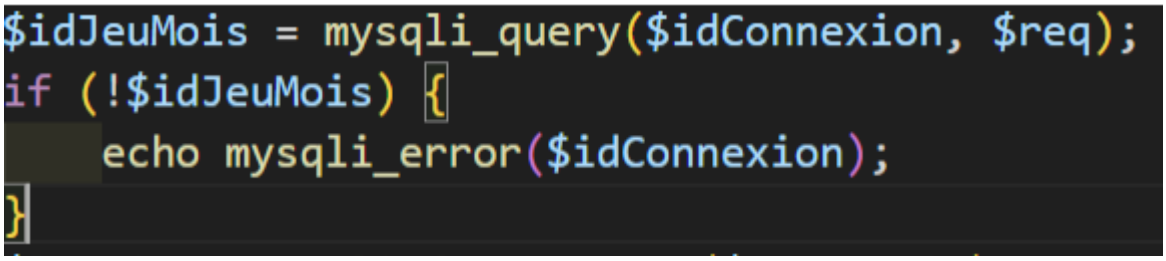
Le code source d'origine reposait sur du PHP impératif (ou brut), sans patron d'architecture logicielle, rendant la maintenance extrêmement complexe. Ma première mission a consisté en une maintenance corrective critique : analyser le code existant, comprendre les causes de blocage et rétablir un état opérationnel minimal (remise en marche).

2. Résolution des Ruptures de Compatibilité Système (Migration PHP 7+)

Le principal point de blocage de l'application v1 résidait dans l'obsolescence des fonctions natives de communication avec la base de données. Le code utilisait l'extension obsolète `mysql_query()`, totalement supprimée depuis les versions modernes de PHP (PHP 7.3+).

Afin de corriger cette rupture de compatibilité, j'ai mené un travail fastidieux de réécriture de la couche d'accès aux données :

- Remplacement systématique de l'ancienne extension par son alternative sécurisée : mysqli_query().
- Inversion rigoureuse des arguments de connexion (l'instance de connexion devant désormais être injectée en premier paramètre de la fonction).
- Implémentation d'une page de routage d'accueil (index.php) exploitant la fonction header('Location: cAccueil.php') pour éviter des failles d'arborescence à l'indexation.



3. Analyse Algorithmique et Sécurisation de l'Authentification (MD5)

Une fois l'application capable de communiquer de nouveau avec le serveur de base de données SQL, j'ai audité la brique d'authentification. L'application intégrait un chiffrement de type `md5($mdp)` sur les requêtes d'identification.

Pour mettre en évidence la vulnérabilité critique de ce protocole face aux attaques par dictionnaire ou par force brute, j'ai écrit un script de test algorithmique en Python exploitant la bibliothèque native `hashlib`.

Ce programme prend en entrée le hash MD5 généré lors d'une soumission de formulaire et tente de casser l'empreinte en testant de manière combinatoire des chaînes de caractères de test. En moins de 17 secondes, l'algorithme retrouve le mot de passe en clair, démontrant ainsi le manque de robustesse du système.

 Pattern de test cryptographique implanté en Python : hashed_password = hashlib.md5(password.encode()).hexdigest() if hashed_password == target_hash: return password

```python
# Programme de force brute pour trouver un mot de passe de 5 caractères hashé en MD5.
import hashlib

def brute_force_md5(target_hash):
    characters = 'abcdefghijklmnopqrstuvwxyz0123456789'  # Caractères possibles dans le mot de passe
    for a in characters:
        for b in characters:
            for c in characters:
                for d in characters:
                    for e in characters:
                        password = a + b + c + d + e
                        hashed_password = hashlib.md5(password.encode()).hexdigest() # Hashage du mo
                        if hashed_password == target_hash:
                            return password
    return None # Retourne None si aucun mot de passe n'est trouvé
# Exemple d'utilisation
if __name__ == "__main__":
    target_hash = "d07551a62e76546ef39e4c85399f096c" # Hash MD5 du mot de passe "kjx4c
    found_password = brute_force_md5(target_hash) # Appel de la fonction de force brute pour trouver
    if found_password:
        print(f"Mot de passe trouvé: {found_password}")
    else:
        print("Aucun mot de passe trouvé.")
```

```
PS C:\Users\courc\Desktop\BTS SIO SLAM\Code\Aristee\Python> & C:/Users/courc/AppData/Local/Programs/Python/Python313-arm64/python.exe "c:/Users
/courc/Desktop/BTS SIO SLAM/Code/Aristee/Python/brutforce.py"
Mot de passe trouvé: kjx4c
```

4. Limites de l'Architecture Native et Constat d'Obsolescence

Malgré les quelques ajustements ergonomiques apportés et la correction des bugs bloquants majeurs, cette phase de maintenance corrective m'a permis de dresser un constat sans appel : l'application en PHP brut n'était plus viable.

- Ergonomie / UX : L'interface graphique s'avérait austère, rigide et non adaptative aux écrans modernes (manque de frameworks CSS / JS).
- Sécurité : L'éparpillement du code SQL brut au milieu du code HTML multiplie les risques de failles d'injection SQL, et l'utilisation de MD5 est proscrite par les normes actuelles de sécurité (ANSSI).

- Maintenabilité : L'absence de structure type MVC (Modèle-Vue-Contrôleur) rend toute évolution future extrêmement chronophage.

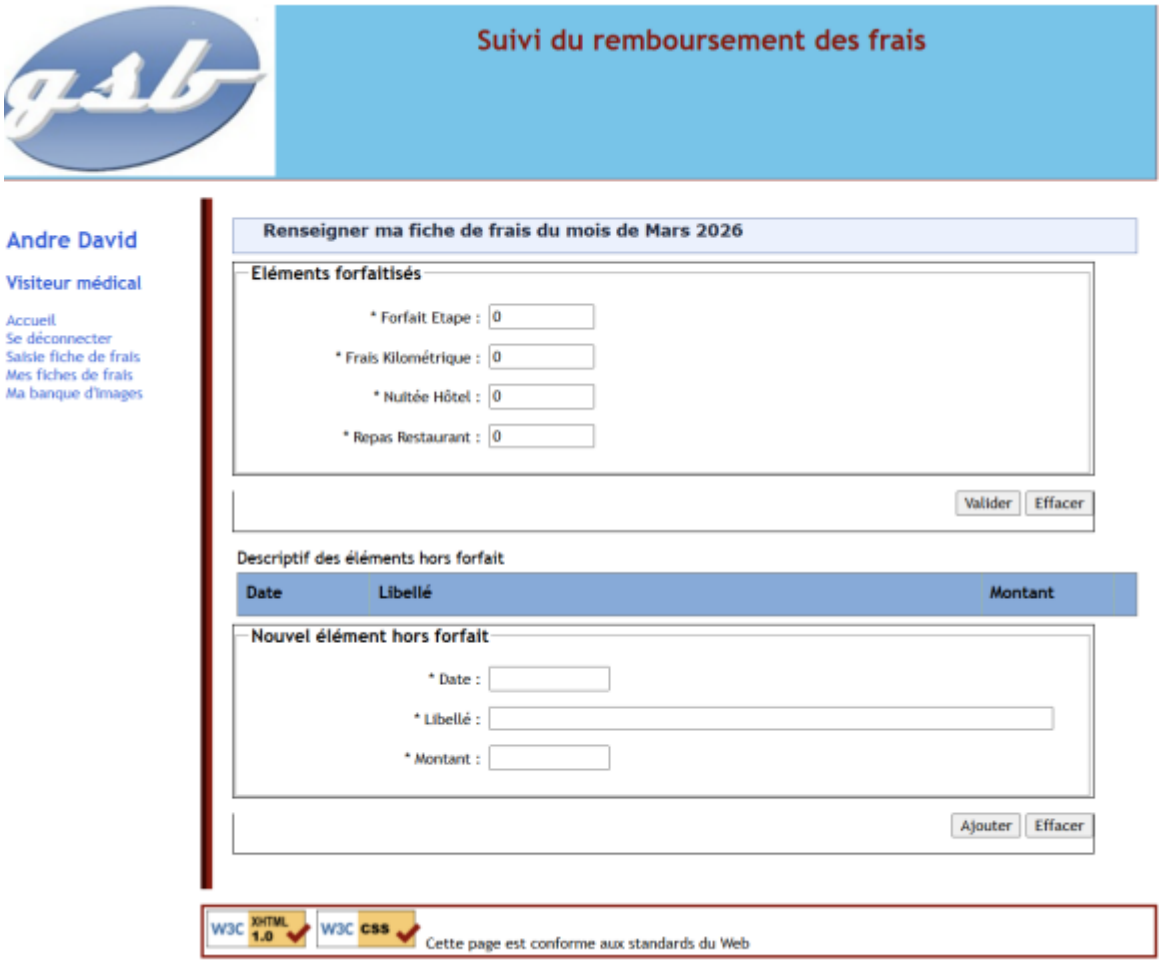
5. Bilan Évolutif : Migration d'Échelle vers le Framework Laravel

C'est précisément l'analyse de ces failles architecturales et structurelles qui a motivé la décision de refondre entièrement l'application AppliFrais en effectuant une migration complète vers le framework industriel **Laravel**.

Laravel répond nativement à l'ensemble des défauts constatés sur la version PHP brute :

- L'utilisation de son ORM Eloquent élimine tout risque d'injection SQL par l'encapsulation native des requêtes.
- Le système d'authentification de Laravel intègre par défaut un algorithme de hachage hautement sécurisé (Bcrypt ou Argon2id), résistant nativement aux scripts de force brute.
- L'architecture MVC claire sépare rigoureusement la logique métier, la gestion des données et les interfaces utilisateurs.

 Conclusion & Veille Technologique : Cette transition d'envergure est directement liée à ma démarche de veille technologique. C'est en recherchant des solutions d'industrialisation locales pour accélérer cette refonte que j'ai identifié et adopté l'outil Laragon. La mise en perspective de l'installation manuelle de Laravel (Composer / PHP natif) combinée à la vélocité offerte par Laragon valide l'ensemble de mes compétences en génie logiciel.



Design de l'application en PHP brut.