BTS SIO — Option SLAM

Dossier de réalisation technique Applifrais

Mise en place d'une application de gestion des frais des visiteur médicaux pour le contexte GSB

Candidat	COUR Cyrille
Période	Septembre 2025 -> Juin 2026
Contexte d'étude	Laboratoire GSB (Aristée)
Technologies	Laragon, PHP 8.x, Eloquent ORM, Blade, Node.js

TABLE DES MATIÈRES

1. Contexte et analyse de l'application existante

L'application AppliFrais est l'outil central utilisé au sein du laboratoire GSB (Galaxy Swiss Bourdin) pour suivre et gérer le remboursement des frais professionnels engagés par les visiteurs médicaux lors de leurs déplacements professionnels.

1.1 Dysfonctionnements constatés sur la version PHP brut legacy

À ma prise en main de l'application initiale écrite en PHP brut, plusieurs dysfonctionnements majeurs empêchaient son bon fonctionnement :

- Incompatibilité de version : Le code utilisait l'extension obsolète mysql_query(), supprimée depuis PHP 7.x. Pour rendre l'application fonctionnelle à court terme, j'ai dû remplacer l'ensemble de ces appels par l'extension moderne mysqli_query() en inversant l'ordre des paramètres requis.
- Failles de sécurité : Les mots de passe étaient stockés en base de données à l'aide de l'algorithme MD5. À l'aide d'un script de test en Python utilisant le brute-force, j'ai mis en évidence qu'une empreinte MD5 simple se cassait en seulement 17 secondes, ce qui posait un problème de confidentialité critique.
- Maintenance complexe : Le code mélangeait les requêtes SQL, la logique applicative et le rendu HTML au sein des mêmes fichiers, rendant le débogage complexe.

1.2 Objectifs métiers et choix techniques du framework Laravel

Pour résoudre définitivement ces limites et moderniser l'infrastructure, j'ai pris la décision de migrer l'application vers le framework Laravel. Ce choix répond à trois objectifs : implémenter une architecture MVC claire, sécuriser les données via l'ORM Eloquent (requêtes préparées automatiques) et hacher les mots de passe avec Bcrypt. Le développement en local a été mené sous l'environnement Laragon.

2. Sécurisation et Back-End : infrastructure de la BDD

L'intégration de Laravel nécessite de faire le lien entre le code et la base de données existante fournie par le CFA, tout en adaptant les mécanismes de sécurité.

2.1 Passage au hachage Bcrypt et adaptation des types SQL

Pour sécuriser les accès conformément aux standards actuels, le système utilise désormais Bcrypt pour le traitement des mots de passe. Cet algorithme génère une chaîne de 60 caractères. La colonne d'origine 'mdp' de la table étant trop courte, elle coupait le hash et rendait l'authentification impossible.

J'ai modifié la structure directement dans phpMyAdmin pour passer le champ en type CHAR(255) afin d'accueillir correctement la clé de chiffrement.

2.2 Configuration non conventionnelle des modèles avec Eloquent

La base de données héritée de GSB ne suit pas les conventions de nommage de Laravel (les tables ne sont pas au pluriel anglais et ne contiennent pas de colonnes de dates automatiques). J'ai configuré les modèles de données pour écraser ces règles par défaut et les mapper correctement sur l'existant :

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class FicheFrais extends Model
{
    protected $table = 'fichefrais';
    // 1. Désactiver l'auto-incrément car il n'y a pas de colonne 'id'
    public $incrementing = false;
    // 2. Indiquer qu'il n'y a pas de clé primaire unique standard 'id'
    protected $primaryKey = null;
    // 3. Autoriser le remplissage de ces champs
    protected $fillable = [
        'idVisiteur',
        'mois',
        'nbJustificatifs',
        'montantValide',
        'dateModif',
        'idEtat'
    ];

    // 4. Désactiver les timestamps automatiques (created_at, updated_at)
    public $timestamps = false;
}

```

Cette configuration a été reproduite pour les modèles FraisForfait et LigneFraisForfait afin d'assurer la cohérence de l'accès aux données.

3. Interface et Expérience Utilisateur (UX)

Le confort d'utilisation et la mise en conformité visuelle de l'application font partie des exigences du projet de refonte.

3.1 Personnalisation de la route racine de bienvenue

À l'initialisation d'un projet Laravel, la route racine affiche une page d'accueil par défaut propre au framework. Pour intégrer l'application dans le contexte d'entreprise GSB, j'ai intercepté cette route dans le fichier de configuration réseau pour rediriger automatiquement l'utilisateur vers notre formulaire de connexion personnalisé.

```php
Route::get('/', function () {
    return view('welcome');
});

// Authentification de base (Profile)
Route::middleware(['auth'])->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');
});

// --- ESPACE VISITEUR ---
Route::middleware(['auth', VisiteurMiddleware::class])->group(function () {
    Route::get('/dashboard', [FicheFraisController::class, 'dashboard'])->name('dashboard');
    Route::get('/rapport', [FicheFraisController::class, 'rapport'])->name('frais.rapport');

    Route::get('/mes-fiches', [FicheFraisController::class, 'index'])->name('frais.index');
    Route::get('/mes-fiches/{id}', [FicheFraisController::class, 'show']);
    Route::get('/saisie-fiches', [FicheFraisController::class, 'create'])->name('frais.create');
    Route::post('/saisie-fiches', [FicheFraisController::class, 'store'])->name('frais.store');
    Route::post('/valider-lignes-frais', [FicheFraisController::class, 'storeLignes'])->name('lignes.store');
    Route::delete('/mes-fiches/{mois}', [FicheFraisController::class, 'destroy'])->name('frais.destroy');

    Route::post('/frais-hors-forfait', [FicheFraisController::class, 'storeHorsForfait'])->name('horsforfait.store');
    Route::delete('/hors-forfait/{id}', [FicheFraisController::class, 'destroyHorsForfait'])->name('horsforfait.destroy');
    Route::get('/hors-forfait/{id}/edit', [FicheFraisController::class, 'editHorsForfait'])->name('horsforfait.edit');
    Route::put('/hors-forfait/{id}', [FicheFraisController::class, 'updateHorsForfait'])->name('horsforfait.update');

    Route::get('/mes-fiches/{mois}/pdf', [FicheFraisController::class, 'exportPdf'])->name('frais.pdf');
    Route::get('/hors-forfait/{id}/pdf', [FicheFraisController::class, 'exportHorsForfaitPdf'])->name('horsforfait.pdf');

    Route::get('/mon-compte', [AccountController::class, 'index'])->name('account.index');
    Route::post('/mon-compte/km', [AccountController::class, 'storeKmRequest'])->name('account.km.store');
});
```

3.2 Mise en place du layout parent et héritage de vues avec Blade

Pour éviter les répétitions de balises HTML sur les différentes pages, j'ai exploité le moteur de template Blade de Laravel en créant un layout principal commun (layouts/app.blade.php). Il intègre les éléments fixes : le header, le logo GSB et la barre de navigation latérale. Les autres pages étendent ce fichier de base.

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="csrf-token" content="{{ csrf_token() }}">

        <title>AppliFrais - GSB</title>

        <!-- Fonts -->
        <link rel="preconnect" href="https://fonts.bunny.net">
        <link href="https://fonts.bunny.net/css?family=figtree:400,500,600&display=swap" rel="stylesheet" />
        <!-- Favicon -->
        <link rel="icon" type="image/png" href="{{ asset('images/favicon.png') }}">
        <!-- Scripts -->
        @vite(['resources/css/app.css', 'resources/js/app.js'])
    </head>
    <body class="font-sans antialiased bg-gray-100">
        <div class="min-h-screen bg-gray-100">
            @include('layouts.navigation')

            <!-- Page Heading -->
            @isset($header)
                <header class="bg-white shadow">
                    <div class="max-w-7xl mx-auto py-4 px-4 sm:px-6 lg:px-8">
                        {{ $header }}
                    </div>
                </header>
            @endisset

            <!-- Page Content -->
            <main>
                {{ $slot }}
            </main>
        </div>
    </body>
</html>
```

4. Architecture logicielle : routage et contrôleurs

Le traitement des données suit un cheminement strict via le patron d'architecture Modèle-Vue-Contrôleur mis en œuvre par le framework.

4.1 Cartographie des routes applicatives métiers

Toutes les URLs de l'application sont déclarées de manière centralisée. Cela sécurise les accès en interdisant d'appeler directement un script physique à la racine, contrairement à l'ancienne méthode en PHP brut.

4.2 Développement des méthodes de traitement des fiches

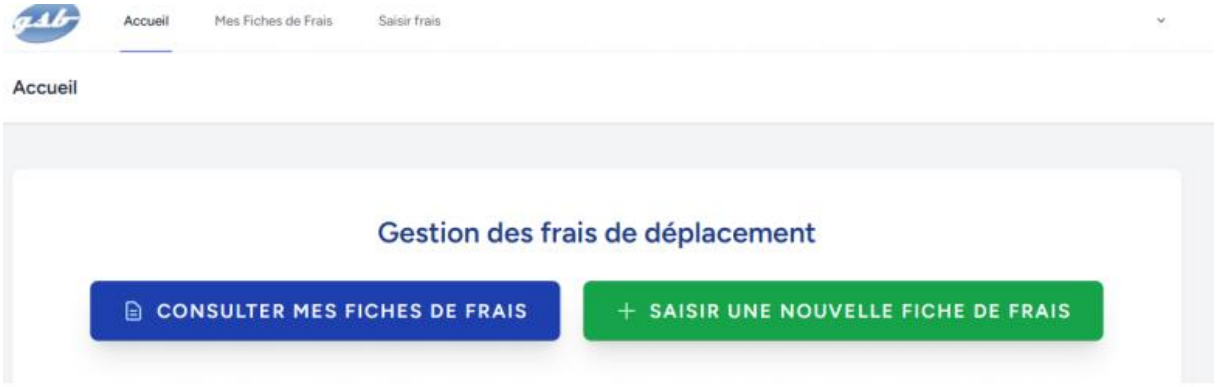
Le contrôleur FicheFraisController fait le lien entre les requêtes de l'utilisateur et les données. La méthode index permet d'extraire les fiches correspondant à l'utilisateur actuellement identifié en session, puis trie les résultats par date décroissante avant de les transmettre à l'interface Blade.

app > Http > Controllers > FicheFraisController.php > ...

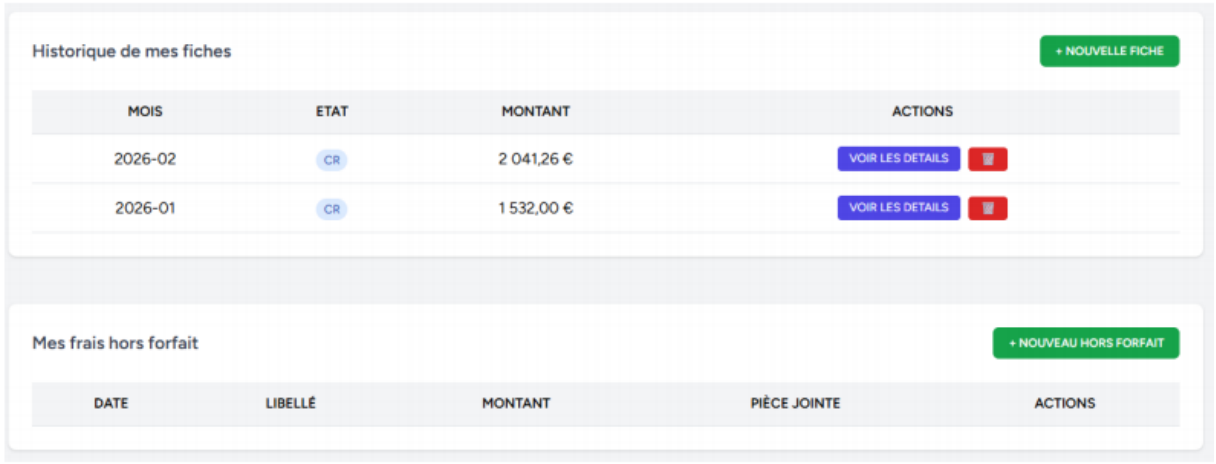
```php
<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Storage;
use App\Models\FicheFrais;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use DateTime;

class FicheFraisController extends Controller
{
    private function moisCourant(): string
    {
        return date('Ym');
    }

    private function estMoisCourant(string $mois): bool
    {
        return $mois === $this->moisCourant();
    }

    private function getKmRateSql($user): string    Parameter $user has no type information available.
    {
        return ($user->fraisKM !== null) ? (float)$user->fraisKM : 'fraisforfait.montant';
    }

    private function getKmRateValue($user): float    Parameter $user has no type information available.
    {
        return $user->fraisKM ?? (float)DB::table('fraisforfait')->where('id', 'KM')->value('montant');
    }

    public function dashboard()
    {
        $user   = Auth::user();
        $kmRate = $this->getKmRateSql($user);

        $premiereFiche = DB::table('fichefrais')
            ->where('idVisiteur', $user->id)
            ->orderBy('mois', 'asc')
            ->first();

        if ($premiereFiche) {
            $dateCurseur = DateTime::createFromFormat('Ym', $premiereFiche->mois);
            $moisActuel  = date('Ym');

            while ($dateCurseur->format('Ym') < $moisActuel) {
                $moisAControler = $dateCurseur->format('Ym');
                $existe = DB::table('fichefrais')
                    ->where('idVisiteur', $user->id)
                    ->where('mois', $moisAControler)
                    ->exists();
```

APPLI FRAIS

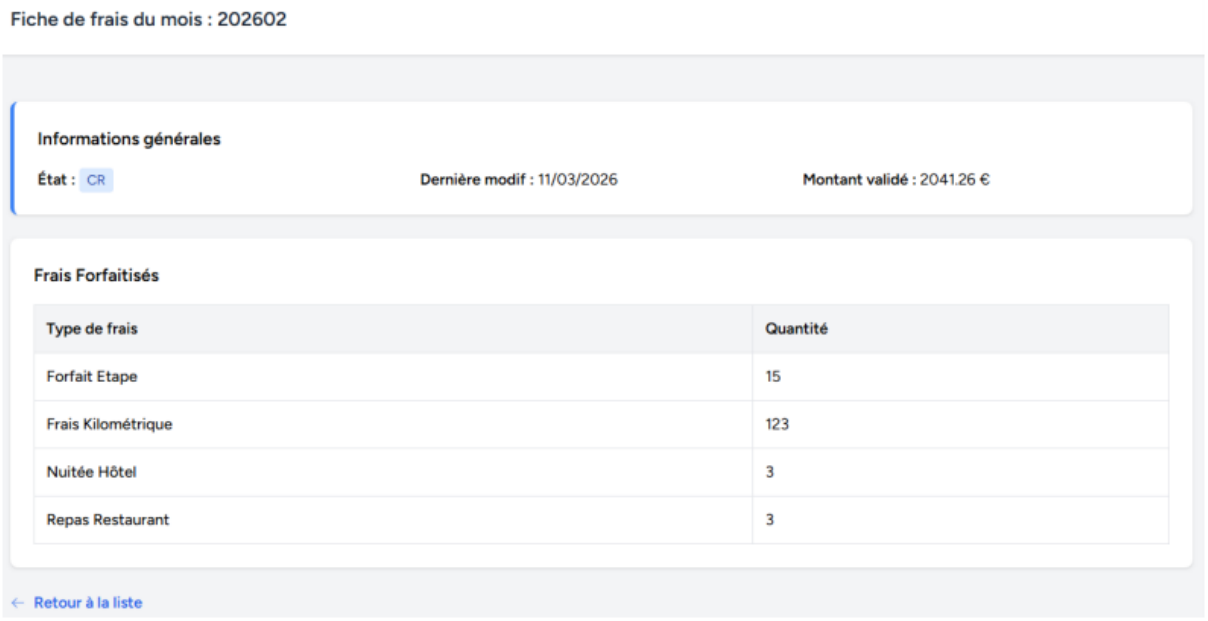
Pour la saisie des fiches de frais, le contrôleur intègre également des routines de réinitialisation permettant de vider proprement une période donnée avant de réécrire les nouvelles lignes, évitant ainsi la création de doublons en base de données.



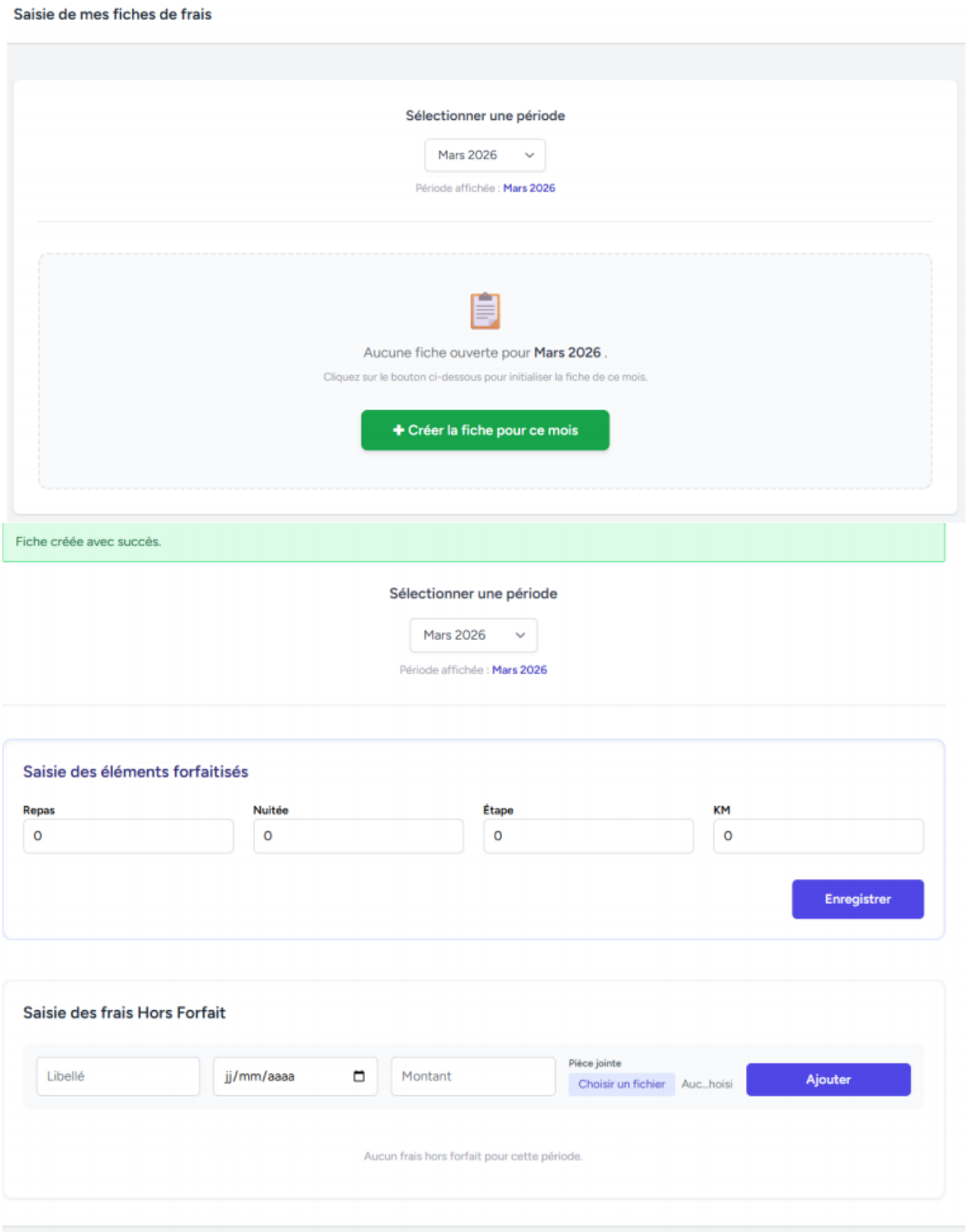
Voici mon écran mes fiches de frais :



L'écran de détail d'une fiche :



L'écran de création d'une fiche :



5. Extension métier N°1 : gestion de la puissance fiscale

Une évolution fonctionnelle majeure a été intégrée : le calcul automatique du remboursement des frais kilométriques basé sur la puissance fiscale du véhicule du visiteur.

5.1 Persistance des informations de la Carte Grise

J'ai ajouté un formulaire permettant au visiteur de renseigner les informations de sa carte grise : marque, immatriculation et chevaux fiscaux (CV). Un modèle Vehicule gère la persistance de ces données.

```php
    /**
     * Afficher le formulaire de confirmation pour une demande KM
     */
    public function showDemandeKM($id)    Parameter $id has no type information available.
    {
        $demande = DB::table('demande_km')
            ->join('visiteur', 'demande_km.idVisiteur', '=', 'visiteur.id')
            ->where('demande_km.id', $id)
            ->select('demande_km.*', 'visiteur.nom', 'visiteur.prenom', 'visiteur.fraisKM')
            ->firstOrFail();

        return view('comptable.show_demande_km', compact('demande'));    View [comptable.show_demande_km] not found.
    }

    /**
     * Valider une demande KM avec confirmation du tarif
     */
    public function validerDemandeKM(Request $request, $id)    Parameter $id has no type information available.
    {
        $request->validate([
            'tarif_confirme' => 'required|numeric|min:0.01',
        ]);

        $demande = DB::table('demande_km')->where('id', $id)->firstOrFail();

        // Met à jour le fraisKM du visiteur
        DB::table('visiteur')
            ->where('id', $demande->idVisiteur)
            ->update(['fraisKM' => $request->tarif_confirme]);

        // Passe la demande en validée
        DB::table('demande_km')
            ->where('id', $id)
            ->update(['statut' => 'VA']);

        return redirect()->route('comptable.dashboard')
            ->with('status', 'Demande KM validée et tarif mis à jour.');
    }

    /**
     * Refuser une demande KM
     */
    public function refuserDemandeKM($id)    Parameter $id has no type information available.
    {
        DB::table('demande_km')
            ->where('id', $id)
            ->update(['statut' => 'RE']);

        return redirect()->route('comptable.dashboard')
            ->with('error', 'Demande KM refusée.');
    }
}
```