APPLI FRAIS

WORK IN PROGRESS!! L'appli n'est pas finie, la doc ne l'est donc pas non plus !

Contexte :

Appli frais est une application de gestion des frais médicaux pour les employés de GSB, GSB (Galaxy Swiss Bourdin) est une entreprise fictive utilisé pour le BTS SIO.
Au départ appli frais était une application en PHP brut sur laquelle j'ai effectué un travail de débogage intensif au cours duquel j'ai pris la décision de migrer vers Laravel un Framework PHP pour une appli plus fluide, ergonomique, moderne et sécurisé.
Afin de travailler sur Laravel il m'a fallu faire une installation de Laragon qui fais tourner Laravel et l'installe facilement.

Travaux réalisé :

L'une des premières tâches effectué sur appli-frais fût de connecter ma base de donner et de modifier un paramètres sur phpMyAdmin :
J'ai dû modifier le type d'entré de ma colonne mdp pour la passer en char (255) afin de pouvoir hacher mes MDP en Bcrypt, standard Laravel

Pour rappel la faible sécurité de la BDD de appli frais en PHP brut (mdp en MD5) est l'une des raisons qui m'ont poussé à passer sur Laravel.
Ensuite j'ai modifier l'écran d'accueil de appli frais, lors ce qu'on arrive sur une app Laravel un gros logo de Laravel est présent en plein milieu j'ai choisi de mettre des infos sur Appli Frais pour une meilleure expérience utilisateur.

APPLI FRAIS

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

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>AppliFrais - GSB</title>
        @vite(['resources/css/app.css', 'resources/js/app.js'])
    </head>
    <body class="antialiased bg-gray-100">
        <div class="relative sm:flex sm:justify-center sm:items-center min-h-screen">

            @if (Route::has('login'))
                <div class="sm:fixed sm:top-0 sm:right-0 p-6 text-right z-10">
                    @auth
                        <a href="{{ url('/dashboard') }}" class="font-semibold text-gray-600
hover:text-gray-900 focus:outline focus:outline-2 focus:rounded-sm focus:outline-indigo-
500">Dashboard</a>
                    @else
                        <a href="{{ route('login') }}" class="font-semibold text-gray-600 hover:text-
gray-900 focus:outline focus:outline-2 focus:rounded-sm focus:outline-indigo-500 text-lg">Se
connecter</a>
                    @endauth
                </div>
            @endif

            <div class="max-w-4xl mx-auto p-6 lg:p-8 text-center bg-white shadow-2xl rounded-3xl">

                <div class="flex justify-center mb-8">
                    <img src="{{ asset('images/logo-gsb.png') }}" alt="Logo GSB" class="h-48 w-auto
transform hover:scale-105 transition duration-300">
                </div>

                <h1 class="text-4xl font-extrabold text-blue-900 mb-6 italic">
                    Bienvenue sur AppliFrais !
                </h1>

                <div class="text-gray-600 text-lg leading-relaxed px-4">
                    <p class="mb-4">
                        Le laboratoire <strong>Galaxy Swiss Bourdin (GSB)</strong> est un leader du
secteur pharmaceutique, issu de la fusion entre le géant américain Galaxy et le fleuron européen Swiss
Bourdin.
                    </p>
                    <p>
                        L'application <strong>AppliFrais</strong> a été conçue pour permettre à nos
visiteurs médicaux de saisir et de suivre leurs frais de déplacement (repas, nuitées, kilométrage) en
toute simplicité, garantissant ainsi un remboursement rapide et efficace de leurs missions sur le
terrain.
                    </p>
                </div>

                <div class="mt-10">
                    <a href="{{ route('login') }}" class="bg-blue-800 text-white px-8 py-4 rounded-full
font-bold text-xl hover:bg-blue-700 shadow-lg transition duration-200">
                        Accéder à mon espace
                    </a>
                </div>

            </div>
        </div>
    </body>
</html>
```

Le code HTML de mon fichier welcome.blade.php
Afin de pouvoir se connecter j'ai utilisé la commande « php artisan breeze :install » qui créer la fonction de connexion.

Ensuite j'ai hacher les MDP comme convenue en bcrypt

```php
/*Route::get('/hash-gsb-passwords', function () {
    // 1. On récupère tous les visiteurs
    $visiteurs = User::all();
    $count = 0;

    foreach ($visiteurs as $visiteur) {
        // On vérifie si le mot de passe n'est pas déjà haché avec Bcrypt
        // (Les hash Bcrypt commencent toujours par '$2y$')
        if (!str_starts_with($visiteur->mdp, '$2y$')) {

            // On hache le mot de passe actuel et on le remplace
            $visiteur->mdp = Hash::make($visiteur->mdp);

            // On désactive les timestamps au cas où ils seraient activés dans le modèle
            $visiteur->timestamps = false;

            $visiteur->save();
            $count++;
        }
    }

    return "Succès : $count mots de passe ont été hachés proprement !";
});*/ // Pour lancer le script on écrit à la fin de l'URL /hash-gsb-passwords
// Une fois lancer ON COMMENTE/SUPPRIME la fonction pour éviter un double hash
```

Je l'ai commenté une fois fais pour éviter le double hash.
Un exemple de MDP dans la BDD :

```
$2y$12$XHATDPc/yxA8SNdGll1JfuWhZyLqkO9Ls9XsQjd1mgi...
```

Enfin pour les fonctionnalités de bases j'ai créer le menu d'accueil

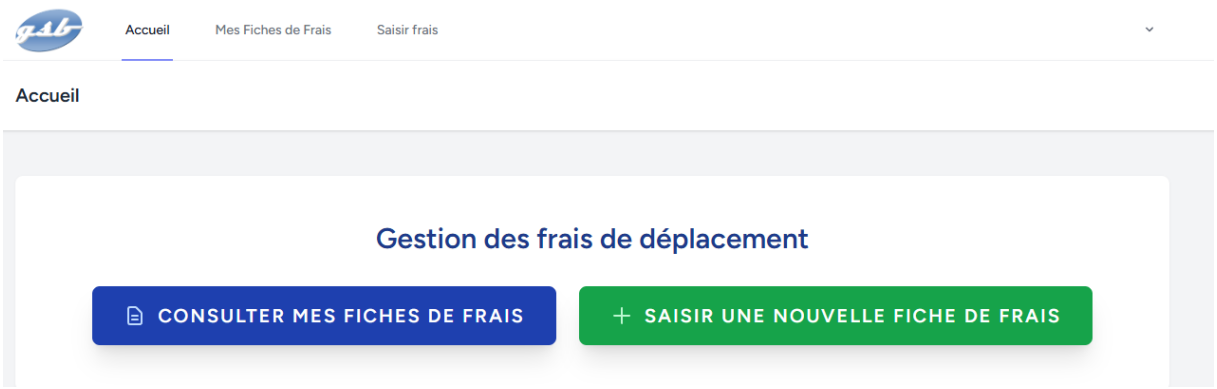
```blade
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Accueil') }}
        </h2>
    </x-slot>

    <div class="py-12">
        <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
            <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg p-12 text-center">

                <h1 class="text-3xl font-bold text-blue-900 mb-8">
                    Gestion des frais de déplacement
                </h1>

                <div class="flex justify-center">
                    <a href="/mes-fiches" class="group relative inline-flex items-center px-8 py-4 bg-blue-800 border border-transparent rounded-lg font-bold text-xl
                        <svg class="w-6 h-6 mr-3 text-blue-200" fill="none" stroke="currentColor" viewBox="0 0 24 24">
                            <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M9 12h6m-6 4h6m2 5H7a2 2 0 01-2-2V5a2 2 0 012-2h5.586a1 1 0 01.707.
                        </svg>
                        Consulter mes fiches de frais
                    </a>
                    <a href="/saisie-fiches" class="group relative inline-flex items-center px-8 py-4 bg-green-600 border border-transparent rounded-lg font-bold tex
                        <svg class="w-6 h-6 mr-3 text-green-200" fill="none" stroke="currentColor" viewBox="0 0 24 24">
                            <path stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M12 4v16m8-8H4"></path>
                        </svg>
                        Saisir une nouvelle fiche de frais
                    </a>
                </div>

            </div>
        </div>
    </div>
</x-app-layout>
```



Voici à quoi sa ressemble, pour lier les pages au différents boutons il a fallu créer beaucoup de routes dans web.php

```php
Route::get('/mes-fiches', [FicheFraisController::class, 'index'])->middleware('auth')->name('frais.index');
Route::get('/mes-fiches/{id}', [FicheFraisController::class, 'show'])->middleware('auth');
Route::get('/saisie-fiches', [FicheFraisController::class, 'create'])->middleware('auth')->name('frais.create');
Route::post('/saisie-fiches', [FicheFraisController::class, 'store'])->middleware('auth')->name('frais.store');
Route::post('/valider-lignes-frais', [FicheFraisController::class, 'storeLignes'])->middleware('auth')->name('lignes.store');
Route::delete('/mes-fiches/{mois}', [FicheFraisController::class, 'destroy'])->middleware('auth')->name('frais.destroy');
Route::post('/frais-hors-forfait', [FicheFraisController::class, 'storeHorsForfait'])->middleware('auth')->name('horsforfait.store');
Route::delete('/hors-forfait/{id}', [FicheFraisController::class, 'destroyHorsForfait'])->middleware('auth')->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');
```

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

Route::get('/dashboar
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

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');
});
```

Chargement en cours...

Enfin le gros du sujet, toute la logique de l'application la création de fiches de frais forfaitaire et hors forfait avec toute les fonctionnalités (que j'ai évidemment ajouter peu à peu) FicheFraisController.php

APPLI FRAIS

```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 Carbon\Carbon;

class FicheFraisController extends Controller
{
    /**
     * Retourne la liste des 12 mois glissants autorisés.
     * Format : [['valeur' => '202503', 'label' => 'Mars 2025'], ...]
     */
    private function getMoisDisponibles(): array
    {
        $mois = [];
        for ($i = 0; $i < 12; $i++) {
            $date = Carbon::now()->subMonths($i)->startOfMonth();
            $mois[] = [
                'valeur' => $date->format('Ym'),
                'label'  => ucfirst($date->translatedFormat('F Y')),
            ];
        }
        return $mois;
    }

    /**
     * Vérifie que le mois demandé est bien dans la fenêtre des 12 mois.
     * Retourne le mois courant si le mois est invalide.
     */
    private function validerMois(string $mois): string
    {
        $autorises = array_column($this->getMoisDisponibles(), 'valeur');
        return in_array($mois, $autorises) ? $mois : Carbon::now()->format('Ym');
```

```php
public function index()
{
    $user = Auth::user();

    $fiches = FicheFrais::where('idVisiteur', $user->id)->orderBy('mois', 'desc')->get();

    $horsForfaits = DB::table('lignefraishorsforfait')
                    ->where('idVisiteur', $user->id)
                    ->orderBy('date', 'desc')
                    ->get();

    return view('frais.index', compact('fiches', 'horsForfaits'));
}

public function show($mois)
{
    $user = Auth::user();

    $fiche = FicheFrais::where('idVisiteur', $user->id)
                       ->where('mois', $mois)
                       ->firstOrFail();

    $lignes = DB::table('lignefraisforfait')
                ->join('fraisforfait', 'lignefraisforfait.idFraisForfait', '=', 'fraisforfait.id')
                ->where('idVisiteur', $user->id)
                ->where('mois', $mois)
                ->select('lignefraisforfait.*', 'fraisforfait.libelle')
                ->get();

    return view('frais.show', compact('fiche', 'lignes'));
}
```

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

```php
public function store(Request $request)
{
    $user = Auth::user();

    $dateInput = $request->input('date_saisie');
    $mois = str_replace('-', '', $dateInput);

    // Sécurité : rejeter tout mois hors de la fenêtre des 12 mois
    $moisValide = $this->validerMois($mois);
    if ($moisValide !== $mois) {
        return redirect()->back()->withErrors([
            'date_saisie' => 'Ce mois n\'est pas dans la période autorisée (12 derniers mois).'
        ]);
    }

    FicheFrais::updateOrCreate(
        ['idVisiteur' => $user->id, 'mois' => $mois],
        [
            'idEtat'          => 'CR',
            'dateModif'       => now(),
            'nbJustificatifs' => 0,
            'montantValide'   => 0,
        ]
    );

    return redirect()->route('frais.create', ['mois' => $mois])
                     ->with('status', 'Fiche créée avec succès.');
}
```

```php
public function create(Request $request)
{
    $user = Auth::user();

    // Récupérer et valider le mois demandé
    $moisSelectionne = $this->validerMois(
        $request->query('mois', Carbon::now()->format('Ym'))
    );

    $moisDisponibles = $this->getMoisDisponibles();

    $fiche = FicheFrais::where('idVisiteur', $user->id)
                       ->where('mois', $moisSelectionne)
                       ->first();

    $lignes = DB::table('lignefraisforfait')
                ->where('idVisiteur', $user->id)
                ->where('mois', $moisSelectionne)
                ->pluck('quantite', 'idFraisForfait');

    $horsForfaits = DB::table('lignefraishorsforfait')
                      ->where('idVisiteur', $user->id)
                      ->where('mois', $moisSelectionne)
                      ->get();

    return view('frais.create', [
        'moisActuel'      => $moisSelectionne,
        'moisDisponibles' => $moisDisponibles,
        'fiche'           => $fiche,
        'lignes'          => $lignes,
        'horsForfaits'    => $horsForfaits,
    ]);
}
```

```php
public function storeLignes(Request $request)
{
    $user = Auth::user();
    $mois = $request->input('mois');

    $typesFrais = ['REP', 'NUI', 'ETP', 'KM'];

    foreach ($typesFrais as $idFrais) {
        $quantite = $request->input($idFrais, 0);

        DB::table('lignefraisforfait')->updateOrInsert(
            [
                'idVisiteur' => $user->id,
                'mois' => $mois,
                'idFraisForfait' => $idFrais
            ],
            [
                'quantite' => $quantite
            ]
        );
    }

    // ✅ Calcul du montant total : quantité x tarif unitaire
    $montantTotal = DB::table('lignefraisforfait')
        ->join('fraisforfait', 'lignefraisforfait.idFraisForfait', '=', 'fraisforfait.id')
        ->where('lignefraisforfait.idVisiteur', $user->id)
        ->where('lignefraisforfait.mois', $mois)
        ->selectRaw('SUM(lignefraisforfait.quantite * fraisforfait.montant) as total')
        ->value('total') ?? 0;
```

```php
        DB::table('fichefrais')
            ->where('idVisiteur', $user->id)
            ->where('mois', $mois)
            ->update([
                'dateModif' => now(),
                'montantValide' => $montantTotal,  // Mise à jour du montant
            ]);

        return redirect()->route('frais.create', ['mois' => $mois])
                         ->with('status', 'Frais forfaitisés enregistrés !');
    }

    public function destroyLigne($mois, $idFrais)
    {
        $user = Auth::user();

        DB::table('lignefraisforfait')
            ->where('idVisiteur', $user->id)
            ->where('mois', $mois)
            ->where('idFraisForfait', $idFrais)
            ->delete();

        return redirect()->back()->with('status', 'Le frais a été supprimé avec succès.');
    }

    public function destroy($mois)
    {
        $user = Auth::user();

        DB::table('lignefraisforfait')
            ->where('idVisiteur', $user->id)
            ->where('mois', $mois)
            ->delete();

        // Supprimer les fichiers uploadés avant de supprimer les lignes
        $horsForfaits = DB::table('lignefraishorsforfait')
```

```
            ->where('idVisiteur', $user->id)
            ->where('mois', $mois)
            ->get();

        foreach ($horsForfaits as $hf) {
            if ($hf->piece_jointe) {
                Storage::disk('public')->delete($hf->piece_jointe);
            }
        }

        DB::table('lignefraishorsforfait')
            ->where('idVisiteur', $user->id)
            ->where('mois', $mois)
            ->delete();

        DB::table('fichefrais')
            ->where('idVisiteur', $user->id)
            ->where('mois', $mois)
            ->delete();

        return redirect()->back()->with('status', 'La fiche de frais a bien été supprimée.');
    }

    public function storeHorsForfait(Request $request)
    {
        $request->validate([
            'libelle'      => 'required|string|max:100',
            'date'         => 'required|date',
            'montant'      => 'required|numeric',
            'piece_jointe' => 'nullable|file|mimes:jpg,jpeg,png,pdf|max:2048',
        ]);

        $user = Auth::user();
        $path = null;
```

```
        if ($request->hasFile('piece_jointe')) {
            $path = $request->file('piece_jointe')->store('pieces_jointes', 'public');
        }

        DB::table('lignefraishorsforfait')->insert([
            'idVisiteur'   => $user->id,
            'mois'         => $request->mois,
            'libelle'      => $request->libelle,
            'date'         => $request->date,
            'montant'      => $request->montant,
            'piece_jointe' => $path,
        ]);

        return redirect()->back()->with('status', 'Frais hors forfait ajouté !');
    }

    public function editHorsForfait($id)
    {
        $hf = DB::table('lignefraishorsforfait')->where('id', $id)->first();
        return view('frais.edit_horsforfait', compact('hf'));
    }

    public function updateHorsForfait(Request $request, $id)
    {
        $request->validate([
            'libelle'      => 'required|string|max:100',
            'date'         => 'required|date',
            'montant'      => 'required|numeric',
            'piece_jointe' => 'nullable|file|mimes:jpg,jpeg,png,pdf|max:2048',
        ]);

        $data = [
            'libelle' => $request->libelle,
            'date'    => $request->date,
            'montant' => $request->montant,
```

```php
    if ($request->hasFile('piece_jointe')) {
        $old = DB::table('lignefraishorsforfait')->where('id', $id)->value('piece_jointe');
        if ($old) {
            Storage::disk('public')->delete($old);
        }
        $data['piece_jointe'] = $request->file('piece_jointe')->store('pieces_jointes', 'public');
    }

    DB::table('lignefraishorsforfait')->where('id', $id)->update($data);

    return redirect()->route('frais.index')->with('status', 'Modifié avec succès.');
}

public function destroyHorsForfait($id)
{
    $hf = DB::table('lignefraishorsforfait')
            ->where('id', $id)
            ->where('idVisiteur', Auth::user()->id)
            ->first();

    if ($hf && $hf->piece_jointe) {
        Storage::disk('public')->delete($hf->piece_jointe);
    }

    DB::table('lignefraishorsforfait')
        ->where('id', $id)
        ->where('idVisiteur', Auth::user()->id)
        ->delete();

    return redirect()->route('frais.index')->with('status', 'Frais hors forfait supprimé.');
}
```

Tout ceci fait énormément de lignes de codes, il gère ce qui concerne les fiches de frais. Voici ce que fait chaque méthode :

Affichage :

- index() → affiche la liste de toutes tes fiches de frais et tes frais hors forfait
- show($mois) → affiche le détail d'une fiche pour un mois précis avec ses lignes forfaitisées
- create() → affiche le formulaire de saisie pour un mois donné

Fiches de frais :

- store() → crée ou initialise une nouvelle fiche pour un mois (avec updateOrCreate)
- destroy($mois) → supprime une fiche entière avec toutes ses lignes forfaitisées, hors forfait et les fichiers uploadés associés

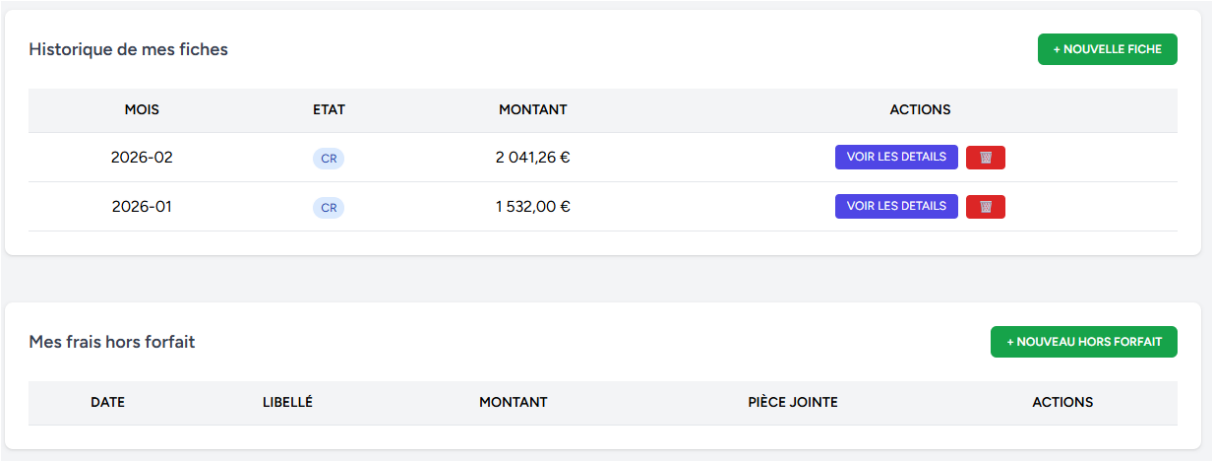
Frais forfaitisés (Repas, Nuitée, Étape, KM) :

- storeLignes() → enregistre ou met à jour les quantités des 4 types de frais forfaitisés
- destroyLigne($mois, $idFrais) → supprime une ligne forfaitisée spécifique

Frais hors forfait :

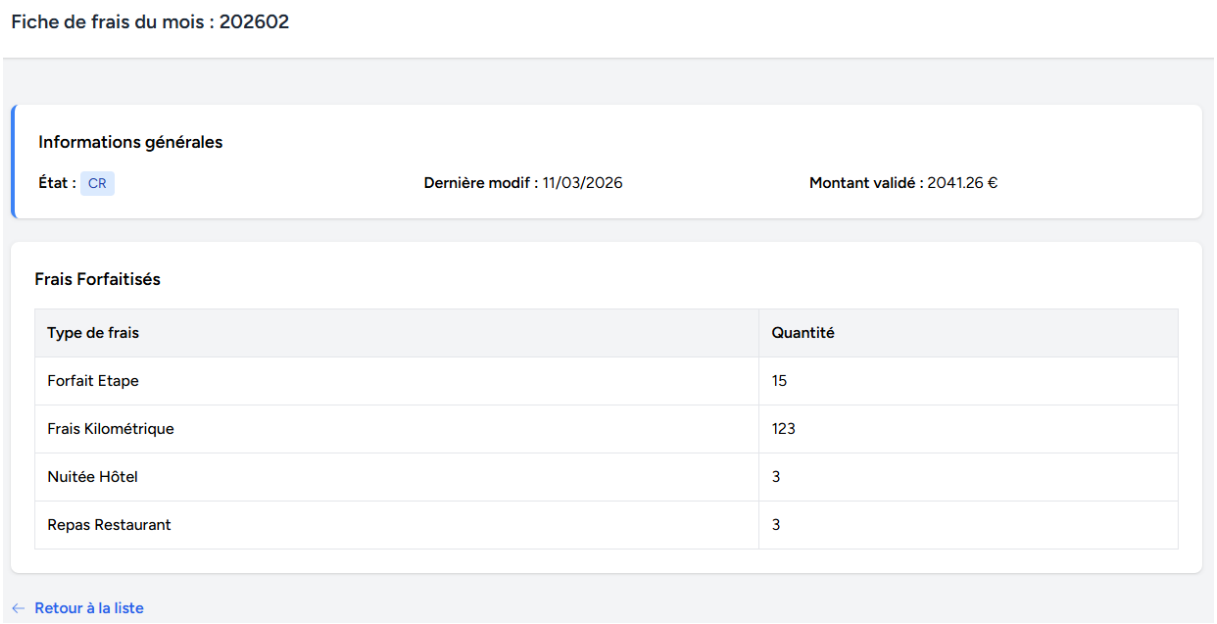
- storeHorsForfait() → ajoute un frais hors forfait avec éventuellement une pièce jointe (jpg, png, pdf)
- editHorsForfait($id) → affiche le formulaire de modification d'un hors forfait
- updateHorsForfait($id) → enregistre les modifications, et remplace la pièce jointe si une nouvelle est uploadée
- destroyHorsForfait($id) → supprime un hors forfait et son fichier associé sur le disque

Voici mon écran mes fiches de frais :

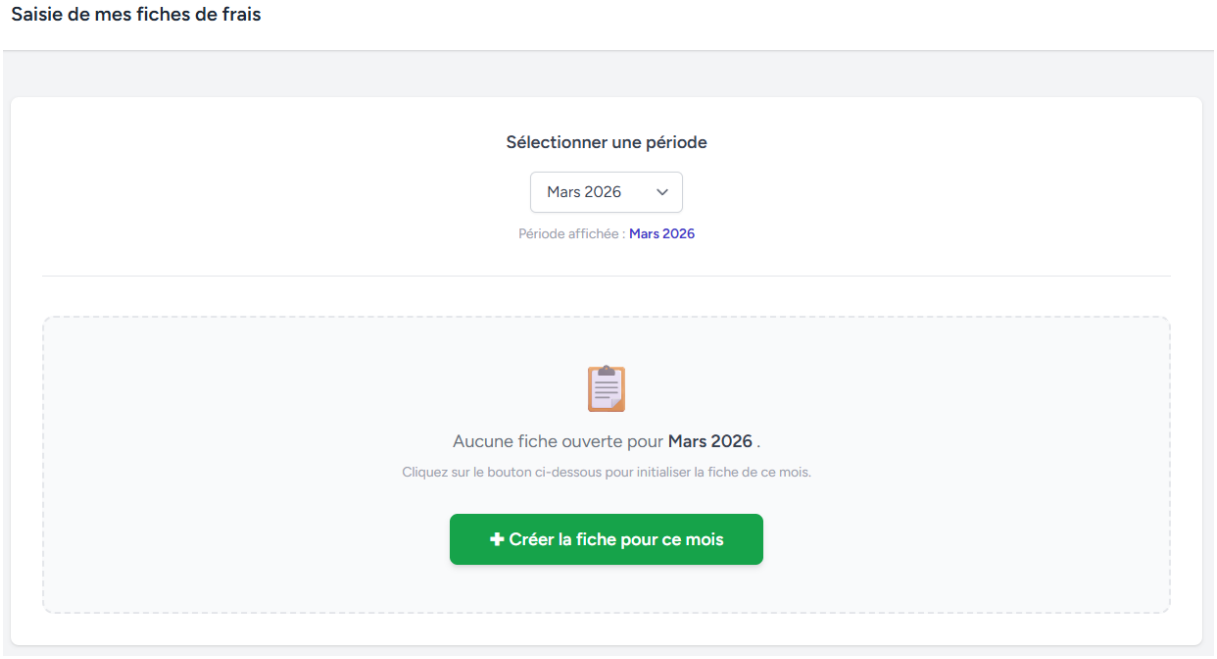


APPLI FRAIS

L’écran de détail d’une fiche :

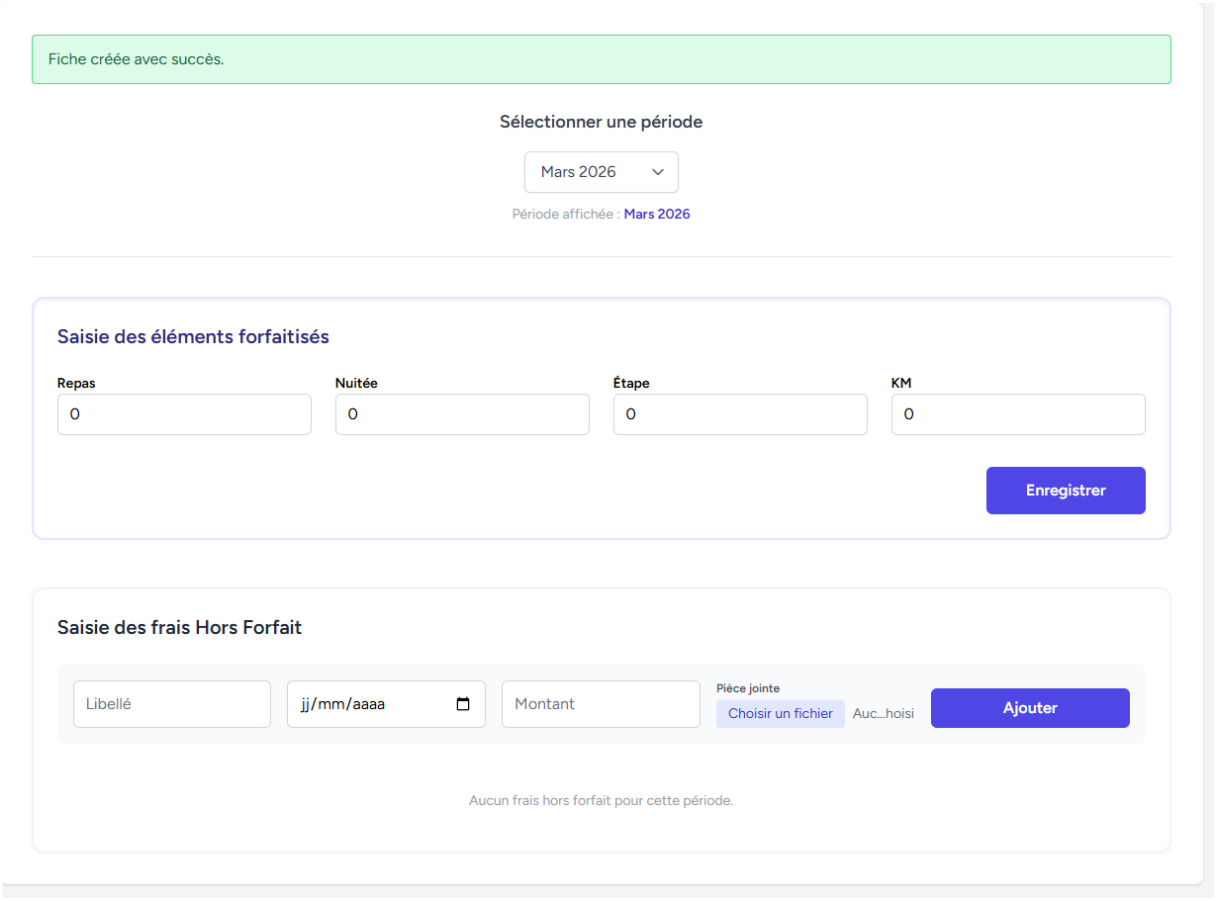


L’écran de création d’une fiche :



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

APPLI FRAIS



Pour adapter la logique Laravel à ma BDD j'ai dû faire quelques modifications. Pour commencer la table User de Laravel n'est pas utilisé, on utilise la table visiteur. Pour cela j'ai créé un model user.php

```php
<?php

namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class User extends Authenticatable
{
    use HasFactory;
    use Notifiable;

    // 1. On utilise la table de GSB pas laravel
    protected $table = 'visiteur';

    protected $primaryKey = 'id';
    public $incrementing = false;
    protected $keyType = 'string';

    // 2. Désactive les colonnes created_at et updated_at
    public $timestamps = false;

    // 3. Les colonnes autorisées
    protected $fillable = [
        'id',
        'nom',
        'prenom',
        'login',
        'mdp',
    ];

    protected $hidden = [
        'mdp',
        'remember_token',
    ];

    // 4. On indique à Laravel que la colonne du mot de passe s'appelle 'mdp'
    public function getAuthPassword()
    {
        return $this->mdp;
    }

    // 5. On indique que l'identifiant pour la session est la colonne 'login'
    public function getAuthIdentifierName()
    {
        return 'login';
    }

    // 6. On hash les MDP
    protected function casts(): array
    {
        return [
            'mdp' => 'hashed',
        ];
    }
}
```

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

J'ai dû faire de même pour les fiches de frais avec FicheFrais.php

```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;
}
```

Mon fichier auth.php dans routes contient toutes les routes liées à l'authentification, générées par Laravel Breeze. Voici ce qu'il contient :

```php
<?php

use App\Http\Controllers\Auth\AuthenticatedSessionController;
use App\Http\Controllers\Auth\ConfirmablePasswordController;
use App\Http\Controllers\Auth\EmailVerificationNotificationController;
use App\Http\Controllers\Auth\EmailVerificationPromptController;
use App\Http\Controllers\Auth\NewPasswordController;
use App\Http\Controllers\Auth\PasswordController;
use App\Http\Controllers\Auth\PasswordResetLinkController;
use App\Http\Controllers\Auth\RegisteredUserController;
use App\Http\Controllers\Auth\VerifyEmailController;
use Illuminate\Support\Facades\Route;
```

```php
Route::middleware('guest')->group(function () {
    /*Route::get('register', [RegisteredUserController::class, 'create'])
        ->name('register');

    Route::post('register', [RegisteredUserController::class, 'store']);*/

    Route::get('login', [AuthenticatedSessionController::class, 'create'])
        ->name('login');

    Route::post('login', [AuthenticatedSessionController::class, 'store']);

    Route::get('forgot-password', [PasswordResetLinkController::class, 'create'])
        ->name('password.request');

    Route::post('forgot-password', [PasswordResetLinkController::class, 'store'])
        ->name('password.email');

    Route::get('reset-password/{token}', [NewPasswordController::class, 'create'])
        ->name('password.reset');

    Route::post('reset-password', [NewPasswordController::class, 'store'])
        ->name('password.store');
});
```

(La route register est commenté car on ne veut pas que les gens puissent créer un nouveau comptes)

```php
Route::middleware('auth')->group(function () {
    Route::get('verify-email', EmailVerificationPromptController::class)
        ->name('verification.notice');

    Route::get('verify-email/{id}/{hash}', VerifyEmailController::class)
        ->middleware(['signed', 'throttle:6,1'])
        ->name('verification.verify');

    Route::post('email/verification-notification', [EmailVerificationNotificationController::class, 'store'])
        ->middleware('throttle:6,1')
        ->name('verification.send');

    Route::get('confirm-password', [ConfirmablePasswordController::class, 'show'])
        ->name('password.confirm');

    Route::post('confirm-password', [ConfirmablePasswordController::class, 'store']);

    Route::put('password', [PasswordController::class, 'update'])->name('password.update');

    Route::post('logout', [AuthenticatedSessionController::class, 'destroy'])
        ->name('logout');
});
```

Cet application bien que pas encore fini respecte la logique CRUD (create read update delete) que j'avais déjà travailler sur l'appli en php brut. Cependant j'ai dû prendre en main Laravel et ses spécialités.

APPLI FRAIS

Les routes : Quand l'utilisateur tape une URL ou clique un bouton, Laravel regarde dans les routes pour savoir quoi faire.
Exemple : Route::get('/mes-fiches', [FicheFraisController::class, 'index'])
Si quelqu'un va sur /mes-fiches, appelle la méthode index du contrôleur.

Les Contrôleurs : Il reçoit la requête, fait la logique métier, interroge la base de données via les modèles, puis envoie les données à la vue.
Exemple : public function index() {

```
$fiches = FicheFrais::where(...)->get();

return view('frais.index', compact('fiches'));
```

}
Cette fonction interroge la base de données pour trouver des fiches de frais filtrées, puis transmet ces données à une page web

Les Modèles : Il représente une table et permet de lire/écrire des données facilement sans écrire de SQL brut
Exemple : FicheFrais.php (qu'on a vue plus haut) une fois créer il nous reste à l'appeler FicheFrais::where('idVisiteur', $user->id)

```
->where('mois', $mois)

->delete();
```

le modèle fait le lien entre le code php et la table de ma BDD.

Les Vues : C'est du HTML avec du Blade (le moteur de templates de Laravel) qui permet d'afficher les données envoyées par le contrôleur, conditions et boucles.
Nous avons vues des exemples bien plus haut.