

A lire : Travail à faire

Nous vous demandons **de travailler sans recopier les travaux des étudiants** qui avancent plus vite. Ce n'est pas une course, **vous avancez à votre allure** et surtout, vous **testez chaque fonctionnalité avant de passer à la suite**.

Vous vous limitez à ce document en oubliant le premier qui parle des frais hors forfait. Si vous avez fini, vous pouvez afficher :

- Les visiteurs avec modification
- Les visiteurs par laboratoire.
-

Modifier une fiche de Frais

Pour réaliser cette fonctionnalité, nous aurons besoin des composants suivants :

Composant	Valeur	Rôle
Routes	/modifierFrais	Prise en compte du clic sur le bouton Modifier de la liste des frais
	/validerFrais	Prise en compte du clic sur le bouton Valider du formulaire
Vue	formFrais.blade.php	Formulaire affichant la fiche de Frais sélectionnée et permettant sa modification
Classes métier	Frais	Stockage des données et requêtes
Contrôleur	FraisController	Pilotage de la demande de modification et de la validation
Méthodes métier	getById()	Lecture de la fiche de Frais
	updateFrais()	Enregistrement des modifications
Méthodes contrôleur	updateFrais()	Affichage du formulaire et appel de la méthode métier getById()
	validateFrais()	Récupération des données saisies et appel de la méthode métier updateFrais()

Les routes

Vous disposez de toutes les informations nécessaires pour construire les deux routes (tableau ci-dessus et vue listeFrais.blade.php). Un indice : une des deux routes dispose d'un paramètre !



Ajouter les deux routes nécessaires.

La page d'erreur

Elle affiche l'erreur

```
@extends('layouts.master')
@section('content')
<div class="alert-danger" role="alert">
    <span class="glyphicon glyphicon-exclamation-sign" aria-hidden="true"></span>
    <p> Voici l'erreur : {{ $monErreur }} </p>
</div>
```

Elle nous donne l'erreur de la SQL

Voici l'erreur :SQLSTATE[42S22]: Column not found: 1054 Unknown column 'frais.i_visiteur' in 'where clause' (SQL: select * from 'frais' where 'frais'.i_visiteur = 1)

La méthode updateFrais()

```
/**
 * Initialise le formulaire de Frais pour une modification
 * Lit la fiche de Frais
 * Initialise le titre de la vue
 * @param type $id Id de la fiche de Frais à modifier
 * @return type Vue formFrais
 */
public function updateFrais($id_frais) {
    try
    {
        $erreur = "";
        $unFrais = new Frais();
        $unFrais = $unFrais->getById($id_frais);
        $titreVue = "Modification d'une fiche de Frais";
        // Affiche le formulaire en lui fournissant les données à afficher
        return view('formFrais', compact('unFrais', 'titreVue', 'erreur'));
    } catch (MonException $e) {
        $monErreur = $e->getMessage();
        return view('vues/pageErreur', compact('monErreur'));
    } catch (Exception $e) {
        $monErreur = $e->getMessage();
        return view('vues/pageErreur', compact('monErreur'));
    }
}
```

La méthode ne présente pas de difficulté, on remarquera simplement que l'on passe au formulaire un objet \$unFrais et donc que les valeurs à afficher dans ce formulaire devront faire référence à cet objet. D'autre part si on passe la variable \$erreur initialisée à vide, cela signifie que l'on aura inclus la vue error dans le formulaire. Si une erreur plus conséquente survient (sql), on enverra le code de l'erreur à une page nommée pageErreur.



Saisir la méthode updateFrais

La méthode validateFrais()

```

/**
 * Enregistre un ajout ou une modification d'une fiche de Frais
 * Si id_frais > 0 c'est une modification, sinon
 * c'est un ajout. Id_frais est dans un INPUT HIDDEN
 * @return type route /getListeFrais
 */
public function validateFrais() {
    try
    {
        $id_frais = Request::input('id_frais');
        $anneemois = Request::input('anneemois');
        $nbjustificatifs = Request::input('nbjustificatifs');
        $unFrais = new Frais();
        if ($id_frais > 0) {
            $unFrais->updateFrais($id_frais, $anneemois, $nbjustificatifs);
        } else {
            $id_visiteur = Session::get('id');
            $unFrais->insertFrais($anneemois, $nbjustificatifs, $id_visiteur);
        }
        // Retourne à la liste des des Frais
        return redirect('/getListeFrais');
    } catch (MonException $e) {
        $monErreur = $e->getMessage();
        return view('vues/pageErreur', compact('monErreur'));
    } catch (Exception $e) {
        $monErreur = $e->getMessage();
        return view('vues/pageErreur', compact('monErreur'));
    }
}

```

On commence par récupérer l'id de la fiche de Frais qui avait été placé dans le champ INPUT HIDDEN, puis les valeurs saisies, on appelle la méthode métier updateFrais en lui passant les paramètres nécessaires et enfin on demande à réafficher la liste des frais à l'aide de l'instruction redirect().



Saisir la méthode validateFrais().

Le formulaire formFrais

Le formulaire est disponible dans les Ressources, il suffira de le compléter avec les informations prises dans les méthodes du contrôleur. On remarquera le champ INPUT HIDDEN dans lequel sera stocké l'id de la fiche de Frais (id_frais). Cet id sera récupéré lors de la validation et nous permettra selon qu'il sera > 0 ou non s'il s'agit de la validation d'une modification ou d'un ajout. Lorsqu'on clique sur le bouton Annuler on appelle un script JavaScript qui utilise la méthode window.location à laquelle on affecte l'url faisant référence à la route /getListeFrais.

Important : le montant validé n'est pas en saisi car ce n'est pas le visiteur qui saisit ce montant mais le comptable qui après avoir fait les vérifications nécessaires saisit ce montant, d'où son nom : montantvalide.

Note : on utilisera systématiquement la structure `{{ $variable or default }}`, par exemple : `{{ $objet->nom or '' }}` ou `{{ $objet->prix or 0 }}`



Récupérer et compléter le formulaire.

La méthode métier `updateFrais()`

```
/**
 * Mise à jour d'une fiche de Frais
 * @param type $id_frais : id de la fiche de Frais à modifier
 * @param type $anneemois : période de la fiche de Frais à modifier
 * @param type $nbjustificatifs : Nb justificatifs de la fiche de Frais à modifier
 */
public function updateFrais($id_frais, $anneemois, $nbjustificatifs) {
    try {
        $dateJour = date("Y-m-d");
        DB::table('frais')->where('id_frais', '=', $id_frais)
            ->update(['anneemois' => $anneemois, 'nbjustificatifs' => $nbjustificatifs,
                'datemodification' => $dateJour]);
    }
    catch (QueryException $e) {
        throw new MonException($e->getMessage(), 5);
    }
}
```

Pas de difficulté particulière. Comme ce n'est pas le visiteur qui saisit la date de modification de la fiche, celle-ci est générée par la fonction PHP `date()` avec un format compatible avec celui des date dans MySQL.



Saisir la méthode métier `updateFrais()` et tester une modification.

Modification d'une fiche de Frais

Période :

201607

Nb justificatifs :

4

Montant validé :

0.00

✓ Valider

✕ Annuler

Frais hors forfait

Ajouter une fiche de Frais

Pour réaliser cette fonctionnalité, nous aurons besoin des composants suivants :

Composant	Valeur	Rôle
Routes	/ajouterFrais	Prise en compte du clic sur le bouton Ajouter du menu général
	/validerFrais	Prise en compte du clic sur le bouton Valider du formulaire
Vue	formFrais.blade.php	Formulaire affichant la fiche de Frais sélectionnée et permettant un ajout
Classes métier	Frais	Stockage des données et requêtes
Contrôleur	FraisController	Pilotage de la demande d'ajout et de la validation
Méthode métier	insertFrais()	Insertion de la fiche de Frais dans la Bdd
Méthodes contrôleur	addFrais()	Affichage du formulaire vide pour un ajout
	validateFrais()	Récupération des données saisies et appel de la méthode métier insertFrais()

Les routes

La route validerFrais ne change pas, elle est commune à l'ajout et la modification. La route d'ajout est simple, elle appelle la méthode addFrais() du contrôleur FraisController. Il faudra aussi faire quelque chose dans la page principale master.blade.php !

Le formulaire formFrais

Il est commun à la modification et l'ajout et ne change pas.

La méthode *addFrais()*

```
/**
 * Initialise le formulaire de Frais pour un ajout
 * @return type Vue formFrais
 */
public function addFrais() {
    try
    {
        $erreur = "";
        $unFrais = new Frais();
        $titreVue = "Ajout d'une fiche de Frais";
        // Affiche le formulaire en lui fournissant les données à afficher
        return view('formFrais', compact('unFrais', 'titreVue', 'erreur'));
    } catch (MonException $e) {
        $monErreur = $e->getMessage();
        return view('vues/pageErreur', compact('monErreur'));
    } catch (Exception $e) {
        $monErreur = $e->getMessage();
        return view('vues/pageErreur', compact('monErreur'));
    }
}
```

La seule chose qui est à noter, est cachée, en effet lorsqu'on fait un `new Frais()`, on invoque le constructeur, or dans ce constructeur, on initialise la propriété `id_frais` à 0 :

```
public function __construct() {
    $this->id_frais = 0;
}
```

On se rappelle que dans le formulaire il y a un champ INPUT HIDDEN qui contient cet id, donc quand on va demander un ajout ce champs contiendra la valeur 0 alors que lorsqu'on demandait une modification, ce champ contenait la valeur de l'id de la fiche à modifier (1, 22, 45 ...).

On verra plus loin que cette valeur 0 va nous être très utile.

La méthode *validateFrais()*

Comme elle est commune à la modification et à l'ajout, elle doit changer pour pouvoir faire la différence entre un update et un insert :

```

/**
 * Enregistre un ajout ou une modification d'une fiche de Frais
 * Si id_frais > 0 c'est une modification, sinon
 * c'est un ajout. Id_frais est dans un INPUT HIDDEN
 * @return type route /getListeFrais
 */
public function validateFrais() {
    try
    {
        $id_frais = Request::input('id_frais');
        $anneemois = Request::input('anneemois');
        $nbjustificatifs = Request::input('nbjustificatifs');
        $unFrais = new Frais();
        if ($id_frais > 0) {
            $unFrais->updateFrais($id_frais, $anneemois, $nbjustificatifs);
        } else {
            $id_visiteur = Session::get('id');
            $unFrais->insertFrais($anneemois, $nbjustificatifs, $id_visiteur);
        }
        // Retourne à la liste des des Frais
        return redirect('/getListeFrais');
    } catch (MonException $e) {
        $monErreur = $e->getMessage();
        return view('vues/pageErreur', compact('monErreur'));
    } catch (Exception $e) {
        $monErreur = $e->getMessage();
        return view('vues/pageErreur', compact('monErreur'));
    }
}

```

Si l'id_frais est égal à 0, c'est qu'il s'agit d'un ajout, sinon c'est qu'il s'agit d'une modification. Dans le cas d'un ajout il faut récupérer dans la Session l'id du visiteur à qui appartient la fiche de Frais que l'on vient de saisir.



Effectuer l'ensemble des modifications nécessaires et tester l'ajout d'une fiche de frais.

Supprimer une fiche de Frais

La suppression d'une fiche en soi n'a rien de difficile, si ce n'est que l'on a ici affaire à une contrainte d'intégrité référentielle entre la table frais et la table fraishorsforfait :

```

alter table fraishorsforfait
    add constraint fk_fraishorsforfait_frais foreign key (id_frais)
    references frais (id_frais);

```

Cela implique que si l'on tente de supprimer une fiche de Frais qui a des Frais Hors Forfait qui lui sont rattachés, la base de données va générer une erreur fatale et "planter" notre application. Nous allons d'abord mettre en évidence ce "plantage", puis ensuite voir comment y remédier.

Les données

Dans un premier temps comme nous n'avons pas développé la saisie des frais hors forfait, nous allons rajouter manuellement des données dans la table fraishorsforfait.



Saisir deux ou trois lignes dans la table fraishorsforfait en les rattachant à des fiche de Frais (id_frais) existentes.

GSB Frais

Pour réaliser la suppression d'une fiche de Frais, nous aurons besoin des composants suivants :

Composant	Valeur	Rôle
Route	/supprimerFrais	Prise en compte du clic sur le bouton Supprimer de la liste de Frais
Vue	listeFrais.blade.php	Liste des Frais du visiteur connecté
Classes métier	Frais	Stockage des données et requêtes
Contrôleur	FraisController	Pilotage de la demande de suppression
Méthode métier	deleteFrais()	Suppression de la fiche de Frais dans la Bdd
Méthode contrôleur	supprimeFrais()	Appel à la méthode métier deleteFrais()

La route

La route supprimerFrais admet le paramètre id_frais et appelle la méthode supprimeFrais du contrôleur.

La vue listeFrais

Elle ne change pas, c'est le script JavaScript qui appellera la route /supprimerFrais.

La méthode métier deleteFrais

```
/**
 * Suppression d'une fiche de Frais
 * @param type $id_frais : Id de la fiche de frais à supprimer
 */
public function deleteFrais($id_frais) {
    try
    {
        DB::table('frais')->where('id_frais', '=', $id_frais)->delete();
    }
    catch (QueryException $e) {
        throw new MonException($e->getMessage(), 5);
    }
}
```

La méthode supprimeFrais

C'est elle la plus importante, mais pour le moment elle reste banale :

```
public function deleteFrais($id_frais) {
    $unFrais = new Frais();
    return redirect('/getListeFrais');
}
```



Effectuer les modifications nécessaires et tenter la suppression d'une fiche de Frais ayant des Frais Hors Forfait. Vous devriez obtenir quelque chose de proche de la copie d'écran ci-dessous :

Whoops, looks like something went wrong.

2/2

QueryException in Connection.php line 729:

SQLSTATE[23000]: Integrity constraint violation: 1451 Cannot delete or update a parent row: a foreign key constraint fails (`gsb`.`fraishorsforfait`, CONSTRAINT `fk_fraishorsforfait_frais` FOREIGN KEY (`id_frais`) REFERENCES `frais` (`id_frais`)) (SQL: delete from `frais` where `id_frais` = 1)

1. in Connection.php line 729
2. at Connection->runQueryCallback('delete from `frais` where `id_frais` = ?', array('1'), object(Closure)) in Connection.php line 685
3. at Connection->run('delete from `frais` where `id_frais` = ?', array('1'), object(Closure)) in Connection.php line 483
4. at Connection->affectingStatement('delete from `frais` where `id_frais` = ?', array('1')) in Connection.php line 438

Notre application est bel et bien plantée, en ce sens qu'elle n'est plus active, l'utilisateur ne peut plus l'utiliser !

Et ça c'est inacceptable, n'oubliez jamais en tant que futur développeur que l'on pourra à la grande rigueur vous "pardonner" qu'une application ne fasse pas exactement ce qu'elle était censée faire, mais on ne vous "pardonnera" JAMAIS qu'elle "plante" !

Une solution

Globalement la solution consiste à "trapper" l'erreur fatale à l'aide d'un gestionnaire d'erreurs. Le framework Laravel met à disposition toute une panoplie d'outils très performants mais aussi assez complexe à mettre en œuvre. Aussi nous allons utiliser une solution à la fois simple et commune à tous les langages, les Exceptions. Le principe est relativement simple on utilise la structure try .. catch ... finally qui permet de lancer des opérations à risque et de capturer l'éventuelle erreur fatale empêchant ainsi l'application de "planter". Voici le nouveau code de la méthode `supprimeFrais()` :

```
/**
 * Supression d'une fiche de frais sur son Id
 * Si la fiche de Frais a des lignes, cela produira une erreur
 * à cause de la contrainte d'intégrité référentielle, on
 * stockera dans la Session le message d'erreur qui sera
 * repris et affiché lors du réaffichage de la liste
 * @param type $id_frais Id de la fiche de Frais à supprimer
 * @return type Vue getListeFrais ou getErrors
 */
public function supprimeFrais($id_frais) {
    $unFrais = new Frais();
    try {
        $unFrais->deleteFrais($id_frais);
    } catch (Exception $ex) {
        Session::put('erreur', $ex->getMessage());
    } finally {
        return redirect('/getListeFrais');
    }
}
```

On tente d'appeler la méthode métier deleteFrais() si tout se passe bien on passe par le bloc finally qui demande le réaffichage de la liste des Frais, par contre si une erreur fatale se produit, on rentre dans le bloc catch dans lequel on place le message d'erreur produit par l'Exception \$ex dans une variable de Session, puis comme précédemment on passe par le bloc finally qui fera réafficher la liste des Frais.

Il nous faut maintenant modifier la méthode getFraisVisiteur() pour tenir compte de la possibilité d'une erreur fatale :

```
/**
 * Affiche la liste de toutes les fiches
 * de Frais du visiteur connecté
 * Si une erreur a été stockée dans la Session
 * on la récupère, on l'efface de la Session
 * et la passe au formulaire
 * @return type Vue listeFrais
 */
public function getFraisVisiteur() {
    try
    {
        $erreur = Session::get('erreur');
        Session::forget('erreur');
        $unFrais = new Frais();
        $id_visiteur = Session::get('id');
        // On récupère la liste de tous les frais
        $mesFrais = $unFrais->getFrais($id_visiteur);
        // On affiche la liste de ces frais
        return view('listeFrais', compact('mesFrais', 'erreur'));

        } catch (MonException $e) {
            $monErreur = $e->getMessage();
            return view('vues/pageErreur', compact('monErreur'));
        } catch (Exception $e) {
            $monErreur = $e->getMessage();
            return view('vues/pageErreur', compact('monErreur'));
        }
    }
}
```

La méthode de classe `Session::get('erreur')` récupère le contenu de la variable de session 'erreur' si celle-ci est vide ou n'existe pas la variable `$erreur` contiendra une chaîne vide ou sera nulle sinon elle contiendra le message d'erreur. Une fois que l'on a récupéré l'erreur, il faut effacer la variable de session erreur, pour ne pas réafficher cette erreur éternellement, ce qui est fait avec la méthode `forget()`.

La vue `error.blade.php` qui est incluse dans la vue `listeFrais.blade.php` teste le contenu de la variable `$erreur` et l'affiche s'il n'est pas vide.









Important : pour que cela fonctionne, il faut faire référence au package PHP Exception, donc en haut de la classe `FraisController`, on ajoutera la ligne :

```
use Request;
use Illuminate\Support\Facades\Session;
use App\metier\Frais;
use Exception;
```



Effectuer les modifications nécessaires et tester une suppression.

Liste des frais

Période	Modifier	Supprimer
201607		
201608		
201609		
201610		

! SQLSTATE[23000]: Integrity constraint violation: 1451 Cannot delete or update a parent row: a foreign key constraint fails (`gsb`.`fraishorsforfait`, CONSTRAINT `fk_fraishorsforfait_frais` FOREIGN KEY (`id_frais`) REFERENCES `frais` (`id_frais`)) (SQL: delete from `frais` where `id_frais` = 1)









Evidemment dans une application aboutie, on n'afficherait pas le message d'erreur de la base de données, mais un message plus compréhensible par l'utilisateur. Il faudrait analyser le contenu du message et si l'on trouve tel ou tel expression, on produirait tel ou tel message. C'est faisable, mais nous emmènerait trop loin ici, nous on se contentera d'afficher le message suivant :

Impossible de supprimer une fiche ayant des Frais Hors Forfait !



Effectuer la modification et tenter une suppression.

Liste des frais

Période	Modifier	Supprimer
201607		
201608		
201609		
201610		

! Impossible de supprimer une fiche ayant des Frais Hors Forfait !