Documentation Technique

GSB – AppliFrais

Table des matières

Présentation

Contexte

Le laboratoire Galaxy Swiss Bourdin (GSB) est issu de la fusion entre le géant américain Galaxy (spécialisé dans le secteur des maladies virales dont le SIDA et les hépatites) et le conglomérat européen Swiss Bourdin (travaillant sur des médicaments plus conventionnels), lui même déjà union de trois petits laboratoires. En 2009, les deux géants pharmaceutiques ont uni leurs forces pour créer un leader de ce secteur industriel. L'entité Galaxy Swiss Bourdin Europe a établi son siège administratif à Paris. Le siège social de la multinationale est situé à Philadelphie, Pennsylvanie, aux États-Unis.

Existant

Avant le début de notre projet, le laboratoire Galaxy Swiss Bourdin (GSB) disposait déjà d'une application PHP permettant aux visiteurs de se connecter et de renseigner leurs fiches de frais. Cependant, cette application ne permettait pas aux comptables d'accéder à ces informations ni de les traiter efficacement. Les comptables de l'entreprise devaient, en effet, utiliser des méthodes manuelles pour vérifier et traiter les données saisies par les visiteurs. Il nous a donc été demandé de compléter l'application afin de permettre aux comptables de se connecter, d’accéder aux fiches de frais, et de disposer d’une interface de gestion facilitant leur travail.

Déroulement du projet

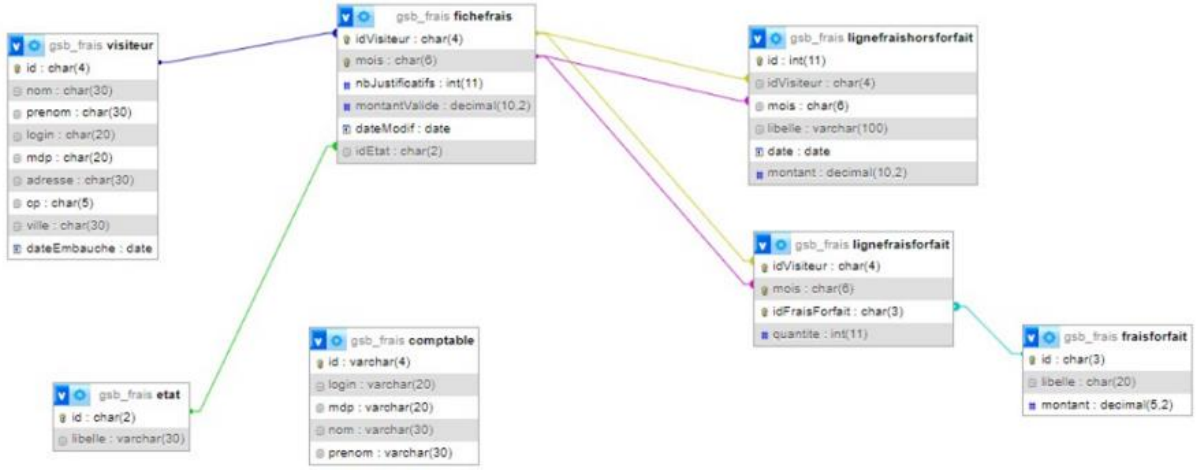
Le projet s'est déroulé en binôme, où chacun des membres a été responsable de certaines tâches spécifiques, en fonction de ses compétences et de son domaine de prédilection. La première phase a consisté à analyser l'existant et à définir les fonctionnalités nécessaires pour permettre aux comptables d'interagir avec l'application. Cela a impliqué la création de nouvelles interfaces de connexion, la gestion des autorisations d'accès, et l'intégration de nouvelles fonctionnalités pour le traitement des fiches de frais.

Le développement a été réalisé avec des outils modernes pour faciliter la gestion des versions et la collaboration. GitHub a été utilisé pour la gestion des versions du projet, permettant ainsi une mutualisation efficace des modifications et une bonne traçabilité des changements. Chaque fonctionnalité a été développée sur des branches séparées, et un système de pull requests a été mis en place pour valider les modifications avant leur fusion avec la branche principale. Cela a permis une gestion fluide du projet et une répartition claire des tâches entre les deux membres du binôme.

Caractéristiques

Base de données

L’application a pour source de données une base de données mysql, accessible sur des outils comme PHPMyAdmin. Voici une représentation de cette base de données ci-dessous :

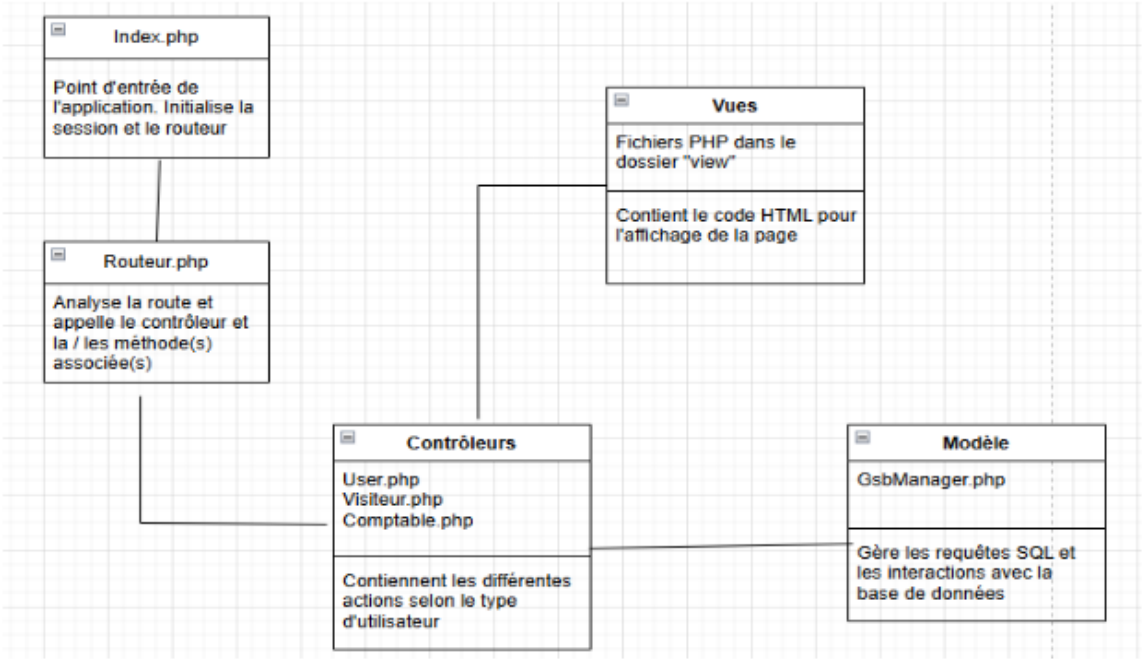


Architecture de l’application

L’application est organisée selon une architecture MVC. L'architecture **MVC** est une approche de conception logicielle qui divise une application en trois composants principaux :

1. **Modèle (Model)** :
    - Le modèle est responsable de la gestion des données de l’application. Il représente la logique métier et l'accès aux données. Par exemple, il peut s'agir de la récupération des informations dans la base de données, du traitement des données ou de la mise à jour de celles-ci. Il est indépendant de l'interface utilisateur (UI).
2. **Vue (View)** :
    - La vue est responsable de l'affichage des informations à l'utilisateur. Elle est l'interface qui permet à l'utilisateur d'interagir avec l'application, mais elle ne contient pas de logique métier. Elle reçoit les données du modèle et les affiche sous une forme lisible et compréhensible. Elle est en charge de la présentation de l’application.
3. **Contrôleur (Controller)** :
    - Le contrôleur fait le lien entre la vue et le modèle. Il écoute les actions de l'utilisateur (comme un clic ou une soumission de formulaire), interagit avec le modèle pour récupérer ou modifier les données, puis met à jour la vue pour refléter ces changements. Le contrôleur sert de médiateur et gère la logique de traitement des requêtes.

Voici un schéma de l’architecture de l’application ci-dessous :



Analyse

Modèle (accès aux données)

Le modèle dans votre projet suit une architecture orientée objet pour gérer l'accès aux données. Il est principalement implémenté dans la classe GsbManager. Voici une explication détaillée de son fonctionnement :

1. Connexion à la base de données

La classe GsbManager utilise un singleton pour gérer la connexion à la base de données. Cela garantit qu'une seule instance de connexion est utilisée tout au long de l'application.

```
<?php
private static $monPdoGsb = null;

public static function getPdoGsb()
{
    if (GsbManager::$monPdoGsb == null) {
        GsbManager::$monPdoGsb = new GsbManager();
    }
    return GsbManager::$monPdoGsb;
}
```

Le constructeur privé initialise une instance de PDO pour interagir avec la base de données.

```
<?php

private function __construct()
{
    GsbManager::$monPdo = new PDO(
        GsbManager::$serveur . ';' . GsbManager::$bdd,
        GsbManager::$user,
        GsbManager::$mdp
    );
    GsbManager::$monPdo->query('SET CHARACTER SET utf8');
}
```

2. Requêtes SQL

La classe contient des méthodes pour exécuter des requêtes SQL spécifiques. Ces méthodes encapsulent les interactions avec la base de données, ce qui permet de centraliser la logique d'accès aux données.

a. Lecture des données

Les méthodes de lecture récupèrent des informations spécifiques de la base de données. Par exemple :

- `getLesFraisForfait($idVisiteur, $mois)` : Récupère les frais forfaitisés d'un visiteur pour un mois donné.
- `getLesFraisHorsForfait($idVisiteur, $mois)` : Récupère les frais hors forfait d'un visiteur pour un mois donné.
- `getLesInfosFicheFrais($idVisiteur, $mois)` : Récupère les informations générales d'une fiche de frais.

Exemple :

```
<?php
public function getLesFraisForfait($idVisiteur, $mois)
{
    $requetePrepare = GsbManager::$monPdo->prepare(
        'SELECT fraisforfait.id as idfrais, fraisforfait.libelle as libelle,
        lignefraisforfait.quantite as quantite
        FROM lignefraisforfait
        INNER JOIN fraisforfait
        ON fraisforfait.id = lignefraisforfait.idfraisforfait
        WHERE lignefraisforfait.idvisiteur = :unIdVisiteur
        AND lignefraisforfait.mois = :unMois'
    );
    $requetePrepare->bindParam(':unIdVisiteur', $idVisiteur, PDO::PARAM_STR);
    $requetePrepare->bindParam(':unMois', $mois, PDO::PARAM_STR);
    $requetePrepare->execute();
    return $requetePrepare->fetchAll();
}
```

b. Mise à jour des données

Les méthodes de mise à jour modifient les données existantes dans la base. Par exemple :

- majFraisForfait($idVisiteur, $mois, $lesFrais) : Met à jour les quantités des frais forfaitisés.
- majEtatFicheFrais($idVisiteur, $mois, $etat) : Met à jour l'état d'une fiche de frais.

Exemple :

```
<?php

public function majEtatFicheFrais($idVisiteur, $mois, $etat)
{
    $requetePrepare = GsbManager::$monPdo->prepare(
        'UPDATE ficheFrais
        SET idetat = :unEtat, datemodif = now()
        WHERE fichefrais.idvisiteur = :unIdVisiteur
        AND fichefrais.mois = :unMois'
    );
    $requetePrepare->bindParam(':unEtat', $etat, PDO::PARAM_STR);
    $requetePrepare->bindParam(':unIdVisiteur', $idVisiteur, PDO::PARAM_STR);
    $requetePrepare->bindParam(':unMois', $mois, PDO::PARAM_STR);
```

```
    $requetePrepare->execute();
}
```

c. Création et suppression

Certaines méthodes permettent d'ajouter ou de supprimer des données :

- `creeNouveauFraisHorsForfait($idVisiteur, $mois, $libelle, $date, $montant)` : Ajoute un nouveau frais hors forfait.
- `supprimerFraisHorsForfait($idFrais)` : Supprime un frais hors forfait.

Exemple

```
<?php
public function supprimerFraisHorsForfait($idFrais)
{
    $requetePrepare = GsbManager::$monPdo->prepare(
        'DELETE FROM lignefraishorsforfait
        WHERE lignefraishorsforfait.id = :unIdFrais'
    );
    $requetePrepare->bindParam(':unIdFrais', $idFrais, PDO::PARAM_STR);
    $requetePrepare->execute();
}
```

3. Gestion des erreurs

Les requêtes SQL sont préparées pour éviter les injections SQL. Les paramètres sont liés à l'aide de `bindParam`, ce qui renforce la sécurité.

4. Centralisation des données

Toutes les interactions avec la base de données passent par cette classe, ce qui facilite la maintenance et les modifications futures. Les contrôleurs utilisent cette classe pour récupérer ou modifier les données nécessaires.

5. Exemple d'utilisation dans un contrôleur

Dans Comptable.php, les méthodes du modèle sont appelées pour récupérer ou mettre à jour les données :

```
<?php
$pdo = GsbManager::getPdoGsb();
$lesFraisForfait = $pdo->getLesFraisForfait($idVisiteur, $leMois);
$pdo->majEtatFicheFrais($idVisiteur, $leMois, "VA");
```

En résumé, le modèle est une couche d'abstraction qui encapsule toutes les interactions avec la base de données, offrant une interface simple et sécurisée pour les contrôleurs.

Vue

La gestion des vues dans votre projet repose sur la classe View, qui est responsable de l'affichage des pages HTML en fonction des données fournies par les contrôleurs. Voici une explication détaillée de son fonctionnement :

1. Classe View

La classe View encapsule la logique de rendu des vues. Elle utilise un système de templates pour inclure les fichiers de vue correspondants.

a. Attribut principal

- `$template` : Définit le nom du fichier de vue à utiliser.

b. Constructeur

Le constructeur initialise le template à utiliser pour la vue.

```php
<?php
public function __construct($template = null)
{
    $this->template = $template;
}
```

2. Méthodes principales

a. Méthode `render`

Cette méthode est utilisée pour afficher une vue standard. Elle extrait les paramètres passés par le contrôleur et inclut le fichier de vue correspondant.

```php
<?php
public function render($params = array())
{
    extract($params); // Rend les variables accessibles dans la vue
    $template = $this->template;

    ob_start(); // Démarre la mise en tampon de sortie
    if ($template == 'v_etatFrais') {
        include(VIEW . 'v_listeMois.php'); // Inclut une vue supplémentaire si nécessaire
    }
    include(VIEW . $template . '.php'); // Inclut le fichier de vue principal
    $contentPage = ob_get_clean(); // Récupère le contenu généré

    include_once(VIEW . '_gabarit.php'); // Affiche le gabarit principal
}
```

b. Méthode `renderc`

Cette méthode est utilisée pour des vues spécifiques aux comptables. Elle inclut des fichiers de vue supplémentaires en fonction du template.

```php
<?php
public function renderc($params = array())
{
    extract($params);
    $template = $this->template;

    ob_start();
    if ($template == 'vc_ficheFrais') {
        include(VIEW . 'vc_listeMois.php');
    } elseif ($template == 'vc_ficheFraisVA') {
        include(VIEW . 'vc_listeMoisVA.php');
    }
    include(VIEW . $template . '.php');
    $contentPage = ob_get_clean();

    include_once(VIEW . '_gabarit.php');
}
```

3. Structure des vues

Les fichiers de vue sont situés dans le dossier `view`. Voici quelques exemples de fichiers de vue et leur rôle :

a. Fichier _gabarit.php

Ce fichier sert de gabarit principal pour toutes les vues. Il inclut les en-têtes HTML, les styles CSS, et le contenu de la page

```php
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Intranet du Laboratoire Galaxy-Swiss Bourdin</title>
    <link href="<?= ASSETS; ?>styles/bootstrap/bootstrap.css" rel="stylesheet">
    <link href="<?= ASSETS; ?>styles/style.css" rel="stylesheet">
</head>
<body>
    <div class="container">
        <?= $contentPage; ?> <!-- Contenu de la vue -->
    </div>
</body>
</html>
```

b. Fichier v_404.php

Affiche une page d'erreur 404.

```php
<div id="accueil">
    <h2>Gestion des frais</h2>
</div>
```

```
<div class="row text-center">
    <h2>La page demandée n'existe pas !</h2>
</div>
```

c. Fichier vc_ficheFrais.php

Affiche les détails d'une fiche de frais.

```
<div class="panel panel-primary">
    <div class="panel-heading">Fiche de frais du mois <?= $numMois . '-' . $numAnnee ?></div>
    <div class="panel-body">
        <strong>État :</strong> <?= $libEtat ?> depuis le <?= $dateModif ?><br>
        <strong>Montant validé :</strong> <?= $montantValide ?>
    </div>
</div>
```

4. Interaction avec les contrôleurs

Les contrôleurs, comme Comptable, appellent la classe View pour afficher les pages. Par exemple :

```
<?php
$myView = new View('vc_ficheFrais');
$myView->renderc(array(
    'estConnecte' => true,
    'visiteurSelectionne' => $idVisiteur,
    'lesVisiteurs' => $lesVisiteurs,
    'lstMois' => $leMois,
    'lesMois' => $lesMois,
    'numMois' => $numMois,
    'numAnnee' => $numAnnee,
    'libEtat' => $libEtat,
    'dateModif' => $dateModif,
    'montantValide' => $montantValide,
    'nbJustificatifs' => $nbJustificatifs,
    'lesFraisForfait' => $lesFraisForfait,
    'lesFraisHorsForfait' => $lesFraisHorsForfait
));
```

Contrôleur

Le fonctionnement des contrôleurs dans votre projet repose sur une architecture MVC (Modèle-Vue-Contrôleur). Les contrôleurs sont responsables de la gestion des actions demandées par l'utilisateur, de l'interaction avec le modèle pour récupérer ou modifier les données, et de l'appel des vues pour afficher les résultats.

1. Structure des contrôleurs

Les contrôleurs sont situés dans le dossier `controller`. Voici les principaux contrôleurs et leurs responsabilités :

- **User.php** : Gère les actions liées à la connexion et déconnexion des utilisateurs (visiteurs ou comptables).
- **Visiteur.php** : Gère les actions spécifiques aux visiteurs, comme la gestion des frais.
- **Comptable.php** : Gère les actions spécifiques aux comptables, comme la validation ou le suivi des fiches de frais.

2. Fonctionnement général

Chaque contrôleur est une classe qui contient des méthodes correspondant aux actions possibles. Ces méthodes sont appelées par le routeur en fonction de la requête utilisateur.

Exemple de méthode dans un contrôleur :

```php
<?php
public function accueil($params)
{
    // Extraction des paramètres de la requête
    extract($params);
    // Appel de la vue pour afficher la page d'accueil
    $myView = new View('v_accueil');
    $myView->render(array('estConnecte' => true));
}
```

3. Contrôleur `User`

Le contrôleur User gère les actions de connexion et déconnexion.

a. Connexion

La méthode `validerConnexion` vérifie les identifiants de l'utilisateur (visiteur ou comptable) et initialise la session.

```php
<?php
public function validerConnexion($params)
{
    extract($params);
```

```
    $pdo = GsbManager::getPdoGsb();
    $visiteur = $pdo->getInfosVisiteur($login, $mdp);
    if (!is_array($visiteur)) {
        $comptable = $pdo->getInfosComptable($login, $mdp);
        if (!is_array($comptable)) {
            // Affiche une vue d'erreur si les identifiants sont incorrects
            $myView = new View('v_erreurs');
            $myView->render(array('estConnecte' => null, 'retour' => "/"));
        } else {
            // Connexion en tant que comptable
            connecter("C", $comptable['id'], $comptable['nom'], $comptable['prenom']);
            $myView = new View('vc_accueil');
            $myView->render(array('estConnecte' => true));
        }
    } else {
        // Connexion en tant que visiteur
        connecter("V", $visiteur['id'], $visiteur['nom'], $visiteur['prenom']);
        $myView = new View('v_accueil');
        $myView->render(array('estConnecte' => true));
    }
}
```

b. Déconnexion

La méthode `deconnexion` détruit la session et affiche une vue de déconnexion.

```
<?php
public function deconnexion($params)
{
    $myView = new View('v_deconnexion');
    deconnecter();
    $myView->render(array('estConnecte' => true));
}
```

4. Contrôleur `Visiteur`

Le contrôleur Visiteur gère les actions spécifiques aux visiteurs.

a. Gestion des frais

La méthode `gererFrais` permet au visiteur de gérer ses frais pour le mois courant.

```
<?php
public function gererFrais($params)
{
    $idVisiteur = $_SESSION['id'];
    $mois = getMois(date('d/m/Y'));
    $pdo = GsbManager::getPdoGsb();
```

```
    extract($params);
    switch ($action) {
        case 'saisirFrais':
            if ($pdo->estPremierFraisMois($idVisiteur, $mois)) {
                $pdo->creeNouvellesLignesFrais($idVisiteur, $mois);
            }
            break;
        case 'validerMajFraisForfait':
            if (lesQteFraisValides($lesFrais)) {
                $pdo->majFraisForfait($idVisiteur, $mois, $lesFrais);
                $pdo->majMontantFicheFrais($idVisiteur, $mois);
            } else {
                $myView = new View('v_erreurs');
                $myView->render(array('estConnecte' => true, 'retour' => 'accueil'));
            }
            break;
    }
}
```

b. État des frais

La méthode `etatFrais` affiche l'état des frais pour un mois donné.

```
<?php
public function etatFrais($params)
{
    $idVisiteur = $_SESSION['id'];
    $pdo = GsbManager::getPdoGsb();
    extract($params);
    switch ($action) {
        case 'selectionnerMois':
            $lesMois = $pdo->getLesMoisDisponibles($idVisiteur);
            $moisASelectionner = array_keys($lesMois)[0];
            $myView = new View('v_listeMois');
            $myView->render(array('estConnecte' => true, 'lesMois' => $lesMois));
            break;
        case 'voirEtatFrais':
            $lesInfosFicheFrais = $pdo->getLesInfosFicheFrais($idVisiteur, $lstMois);
            $myView = new View('v_etatFrais');
            $myView->render(array_merge($lesInfosFicheFrais, ['estConnecte' => true]));
            break;
    }
}
```

5. Contrôleur `Comptable`

Le contrôleur Comptable gère les actions spécifiques aux comptables.

a. Validation des fiches de frais

La méthode `validerFrais` permet au comptable de valider les fiches de frais.

```php
<?php
public function validerFrais($params)
{
    $pdo = GsbManager::getPdoGsb();
    extract($params);
    switch ($action) {
        case 'selectionnerVisiteur':
            $lesVisiteurs = $pdo->getLesVisiteursAValider("CL");
            $myView = new View('vc_listeVisiteurs');
            $myView->renderc(array('estConnecte' => true, 'lesVisiteurs' => $lesVisiteurs));
            break;
        case 'voirFiche':
            $lesInfosFicheFrais = $pdo->getLesInfosFicheFrais($idVisiteur, $lstMois);
            $myView = new View('vc_ficheFrais');
            $myView->renderc(array_merge($lesInfosFicheFrais, ['estConnecte' => true]));
            break;
    }
}
```

b. Suivi des frais

La méthode `suivreFrais` permet au comptable de suivre l'état des fiches de frais.

```php
<?php
public function suivreFrais($params)
{
    $pdo = GsbManager::getPdoGsb();
    extract($params);
    switch ($action) {
        case 'selectionnerVisiteur':
            $lesVisiteurs = $pdo->getLesVisiteursAValider("VA");
            $myView = new View('vc_listeVisiteursVA');
            $myView->renderc(array('estConnecte' => true, 'lesVisiteurs' => $lesVisiteurs));
            break;
    }
}
```

6. Gestion des erreurs

Tous les contrôleurs incluent une méthode `erreur` pour afficher une page 404 en cas d'action inexistante.

```php
<?php
public function erreur()
{
    $myView = new View('v_404');
```

```
    $myView->render(array('estConnecte' => true));
}
```

Tableau des fonctions de l’application

Nom de fonction	Description
__destruct	Méthode destructeur appelée dès qu'il n'y a plus de référence sur un objet donné, ou dans n'importe quel ordre pendant la séquence d'arrêt.
__construct	Constructeur privé, crée l'instance de PDO qui sera sollicitée pour toutes les méthodes de la classe.
creeNouveauFraisHorsForfait	Crée un nouveau frais hors forfait pour un visiteur un mois donné à partir des informations fournies en paramètre.
creeNouvellesLignesFrais	Crée une nouvelle fiche de frais et les lignes de frais au forfait pour un visiteur et un mois donnés. Récupère le dernier mois en cours de traitement, met à 'CL' son champs idEtat, crée une nouvelle fiche de frais avec un idEtat à 'CR' et crée les lignes de frais forfait de quantités nulles.
dernierMoisSaisi	Retourne le dernier mois en cours d'un visiteur.
estPremierFraisMois	Teste si un visiteur possède une fiche de frais pour le mois passé en argument.
getInfosUser	Retourne les informations d'un utilisateur.
getInfosVisiteur	Retourne le nom et prénom d'un visiteur.
getLesFichesFrais	Modifie l'état et la date de modification d'une fiche de frais. Modifie le champ idEtat et met la date de modif à aujourd'hui.
getLesFraisForfait	Retourne sous forme d'un tableau associatif toutes les lignes de frais au forfait concernées par les deux arguments.
getLesFraisHorsForfait	Retourne sous forme d'un tableau associatif toutes les lignes de frais hors forfait concernées par les deux arguments. La boucle foreach ne peut être utilisée ici car on procède à une modification de la structure itérée - transformation du champ date.
getLesIdFrais	Retourne tous les id de la table FraisForfait.
getLesMoisDisponibles	Crée un nouveau frais hors forfait pour un visiteur un mois donné à partir des informations fournies en paramètre.
getLesVisiteurs	Retourne tous les visiteurs de la base de données.
majEtatFicheFrais	Modifie l'état et la date de modification d'une fiche de frais. Modifie le champ idEtat et met la date de modif à aujourd'hui.
majFraisForfait	Met à jour la table ligneFraisForfait pour un visiteur et un mois donné en enregistrant les nouveaux montants.
modifierElementForfait	Modifie les éléments forfaitisés mis à jour par le comptable.
modifierEtatFrais	Modifie l'état d'une fiche de frais de CL à VA. Sert à valider des fiches de frais.
supprimerFraisHorsForfait	Crée un nouveau frais hors forfait pour un visiteur un mois donné à partir des informations fournies en paramètre.
ajouterErreur	Ajoute le libellé d'une erreur au tableau des erreurs.
ajouterInformation	Ajoute le libellé d'une information au tableau des informations.
connecter	Enregistre dans une variable session les infos d'un visiteur.
dateAnglaisVersFrancais	Transforme une date au format anglais aaaa-mm-jj vers le format français jj/mm/aaaa.

Nom de fonction	Description
dateFrancaisVersAnglais	Transforme une date au format français jj/mm/aaaa vers le format anglais aaaa-mm-jj.
deconnecter	Détruit la session active.
estConnecte	Teste si un quelconque visiteur est connecté.
estDateDepassee	Vérifie si une date est inférieure d'un an à la date actuelle.
estDateValide	Vérifie la validité du format d'une date française jj/mm/aaaa.
estEntierPositif	Indique si une valeur est un entier positif ou nul.
estTableauEntiers	Indique si un tableau de valeurs est constitué d'entiers positifs ou nuls.
getMois	Retourne le mois au format aaaamm selon le jour dans le mois.
lesQteFraisValides	Vérifie que le tableau de frais ne contient que des valeurs numériques.
nbErreurs	Retourne le nombre de lignes du tableau des erreurs.
valideInfosFrais	Vérifie la validité des trois arguments : la date, le libellé du frais et le montant. Des messages d'erreurs sont ajoutés au tableau des erreurs.
verifierFichier	Teste la validité et le format d'un fichier à déposer.