Développement de la partie Comptable de l'Application GSB frais

SOMMAIRE

I PRESENTATION DU PROJET

1) Présentation

Développement :
Il s'agit d'une Application pour la validation et le paiement de notes de frais, destinée au comptable, en suivant le modèle MVC.

Cadre :
Il s'agit de la Mission n°1 : Développement de la partie Comptable de la gestion des notes de frais, proposée dans le cadre de PPE par le réseau Certa, dans le contexte professionnel des laboratoires pharmaceutiques Galaxy Swiss Bourdin

Support :
Développé sur un serveur local Wamp, avec l'IDE NetBeans.
Langages = html/css pour la partie Front, et php pour le Back-end.

Détails du contexte :
- Description du contexte : https://portfolio.lencodage.fr/docs/gsb_description_GSB.pdf
- Normes de développement : https://portfolio.lencodage.fr/docs/gsb_Normes_de_developpement.pdf
- Projet appli-frais : https://portfolio.lencodage.fr/docs/gsb_projet_appli_frais.pdf
- Architecture du projet : https://portfolio.lencodage.fr/docs/gsb_architectureMVC.pdf

Accès aux productions :
- Code source : https://github.com/Steph-bts/GSB-Comptable
- Application : https://gsb.lencodage.fr
  Accès Visiteur Médical : identifiant = mdebelle
  mot de passe = od5rt
  Accès Comptable : identifiant = steph
  mot de passe = gsb
- Documentation du code : https://gsbdocumentation.lencodage.fr/

Difficultés rencontrées :
Il m'a été difficile de 'réfléchir' MVC (cela me paraissait au départ très compliqué) tout en développant les fonctionalités demandées.

2) Choix organisationnels

2.1) Nom des tables

Lorsque j'ai démarré le développement de cette application, j'avais déjà un hébergement avec un nom de domaine. Pour des raisons à la fois pratiques et budgétaires, j'avais choisi une formule d'hébergement qui comprend 1 seule base de données. Base que j'avais déjà utilisée pour y mettre des tables indispensables au fonctionnement de quelques TP que j'avais pu déjà héberger.

Ainsi, je savais avant même de faire quoi que ce soit, que pour des raisons purement pratique je ne pouvais pas conserver le nom des tables : elles sont classées par ordre alphabétique par phpMyAdmin, vous pouvez aisément imaginer la difficulté de trouver mes tables relatives à ce projet dans une base contenant déjà à ce moment-là une dizaine d'autres tables sans aucun rapport.

Donc dès le départ, dans la base et dans le code, toutes les tables ont été renommées : j'ai simplement ajouté 'gsb_' devant chaque table. Il est ainsi très facile de les repérer au premier coup d'oeil.

2.2) Organisation du code

Il nous était donc fourni l'appli visiteurs ainsi que le code source. Le code d'origine contenait déjà la classe PdoGsb qui servait pour les accès à la base de données, ainsi que des fonctions. Bien sûr, j'ai réutilisé la majorité de ce qui existait déjà, en y ajoutant mes propres méthodes/fonctions au fur et à mesure du développement.

En sachant ça, il y avait plusieurs possibilités : intégrer l'appli comptable à l'intérieur de l'appli visiteurs existante était l'une d'entre elles, que je n'ai pas choisie. J'ai préféré créer une appli tout à fait distincte de l'application originale, en y intégrant la classe PdoGsb ainsi que les fonctions. Cela permet, dans une situation professionnelle, de travailler en toute indépendance, sans 'toucher' au travail de celui qui a développé la partie visiteurs.

C’est pourquoi vous vous apercevrez que sur mon serveur distant, il y a plusieurs sous-domaines distincts : gsb.lencodage.fr qui vous dirige sur une page d’accueil général, en vous proposant soit l’accès visiteur soit l’accès comptable. Le code de cette page n’est pas très intéressant je n’en détaillerait donc pas le développement dans ce compte-rendu. Les 2 autres sous-domaines sont gsbvisiteurs.lencodage.fr qui héberge l’appli originale, et gsbcomptable.lencodage.fr qui héberge l’application dont je détaille ici le développement.

J’ai aussi fait le choix d’héberger la documentation du code générée sur un sous-domaine dédié, gsbdocumentation.lencodage.fr.

II LA PAGE INDEX.PHP

Cette page, sur laquelle on est automatiquement dirigé lorsqu'on souhaite accéder à l'application, est le chef d'orchestre de l'appli. C'est ici que l'on détermine ce qui se passe, quelle page gère quoi.

On commence par préciser les fichiers requis, puis on lance une session php pour que le navigateur reçoive le cookie de session approprié lors de la connexion de l'utilisateur, et enfin, à partir de l'url c'est cette page qui renvoie vers le contrôleur chargé de gérer la fonctionnalité demandée.

```
require_once 'includes/fct.inc.php';
require_once 'includes/class.pdogsb.inc.php';

session_start();

$pdo = PdoGsb::getPdoGsb();

$estConnecte = estConnecte();
require 'vues/v_entete.php';

$uc = filter_input(INPUT_GET, 'uc', FILTER_SANITIZE_STRING);

if ($uc && !$estConnecte) {
    $uc = 'connexion';
} elseif (empty($uc)) {
    $uc = 'accueil';
}

switch ($uc) {
case 'connexion':
    include 'controleurs/c_connexion.php';
    break;
case 'accueil':
    include 'controleurs/c_accueil.php';
    break;
case 'cloture':
    include 'controleurs/c_cloture.php';
    break;
case 'validation':
    include 'controleurs/c_validation.php';
    break;
case 'paiement':
    include 'controleurs/c_paiement.php';
    break;
case 'deconnexion':
    include 'controleurs/c_deconnexion.php';
    break;
}
require 'vues/v_pied.php';
```

III LES EN-TÊTES

Puisque sur toutes les pages l'en-tête sera incluse, il est temps de s'occuper de son développement. Je me suis bien sûr largement inspirée de ce qui a été fait pour l'appli visiteurs, en modifiant la couleur afin de respecter la charte graphique :

Voici un extrait du code, dans les Vues pour une option :

```
<li <?php
   if ($uc == 'validation') {
 ?>
      class="active"
<?php
    }
 ?>>
      <a href="index.php?uc=validation&action=selectionVisiteurMois">
            <span class="glyphicon glyphicon-hand-right">
             </span>
                    Validation
       </a>
</li>
```

Grâce au framework Bootstrap, il est aisé d'insérer les icônes correspondantes.

IV CLÔTURE AUTOMATIQUE DES FICHES DE FRAIS

Cette tâche n'était pas demandée dans la mission 1, mais seulement dans la mission 2 et dans un autre langage.

Cependant, le jeu test qui remplit automatiquement la base de données créée des fiches de frais en cours de création (idetat = 'CR'), ainsi que des fiches validées (idetat = 'VA'), mais pas de fiches clôturées. Or, puisque l'objectif est de pouvoir valider des fiches … il faut bien qu'au préalable elles aient été clôturées.

J'ai donc décidé, pour des raisons de logique d'utilisation, de commencer par créer un bouton 'Clôture Automatique' afin de pouvoir clôturer les fiches de frais du mois précédent (par rapport à la date de connexion).

Pour ce faire, il faut : créér une méthode dans la classe PdoGsb qui va aller chercher dans la base de données toutes les fiches de frais qui sont en cours de création pour un mois donné en paramètre :

```
public function getVisiteursNonClos($mois)
    {
        $requetePrepare = PdoGSB::$monPdo->prepare(
            "SELECT idvisiteur FROM gsb_fichefrais WHERE idetat = 'CR' "
              . "AND mois = :unMois"
        );
        $requetePrepare->bindParam('unMois', $mois, PDO::PARAM_STR);
        $requetePrepare->execute();
        return $requetePrepare->fetchAll();
    }
```

Ensuite, j'ai besoin d'une fonction qui, quand je lui donne un mois en paramètre, va me retourner le mois précédent :

```
function getMoisPrecedent($mois)
{
    $numAnnee = substr($mois, 0, 4);
    $numMois = substr($mois, 4, 2);
    $numMoisPrecedent = intval($numMois) - 1;
    $numMoisPrecedent = strval($numMoisPrecedent);
    if (strlen($numMoisPrecedent) == 1) {
        $numMoisPrecedent = '0' . $numMoisPrecedent;
    }
    return $numAnnee . $numMoisPrecedent;
}
```

Et enfin, il ne reste qu'à utiliser tout ceci dans le contrôleur, afin qu'il détermine le mois précédent et qu'il aille récupérer dans la bdd les visiteurs concernés :

```
$mois = getMois(date('d/m/Y'));
$moisPrecedent = getMoisPrecedent($mois);
$visiteurs = $pdo->getVisiteursNonClos($moisPrecedent);
```

Le contrôleur, ensuite, gère les différents cas de figures en fonction des actions de l'utilisateur (en envoyant le message d'erreur approprié par exemple). Ci-dessous le contrôleur de la clôture :

```php
$action = filter_input(INPUT_GET, 'action', FILTER_SANITIZE_STRING);

switch ($action) {
case 'cloturer':
    // s'il n'y a pas de fiche en attente de clôture => msq erreur approprié
    if ($nbreFichesACloturer === 0) {
        ajouterErreur("Il n'y a aucune fiche Visiteurs en attente de clôture ! ");
        include 'vues/v_erreurs.php';
    } else {
        include 'vues/v_clotureAutomatique.php';
    }
    break;
case 'succesCloture':
    // clôture des fiches du mois précédent + création de la nouvelle fiche
    $nbreVisiteursClos = count($visiteurs);
    foreach ($visiteurs as $visiteur) {
        $pdo->majEtatFicheFrais($visiteur['idvisiteur'], $moisPrecedent, 'CL');
        $pdo->creeNouvellesLignesFrais($visiteur['idvisiteur'], $mois);
    }
    include 'vues/v_succesCloture.php';
    break;
default:
    include 'vues/v_accueil.php';
}
```

Ce contrôleur utilise, majoritairement, des méthodes qui étaient déjà fournies pour l'appli visiteur.
Et qui fonctionnaient parfaitement … en tout cas sur mon serveur local. Mauvaise surprise lors de l'upload du projet, il y avait une erreur de casse dans la méthode majEtatFicheFrais, qui fait une requête d'update de la bdd de la table ficheFrais. Sauf que … dans la base, cette table s'appelle en réalité fichefrais. Une innocente majuscule responsable de beaucoup de perplexité, je l'avoue, mais qui s'est révélée très formatrice.

V LA VALIDATION DES FICHES DE FRAIS

Maintenant que nous avons des fiches de frais clôturées, on peut enfin s'intéresser au coeur du sujet, c'est-à-dire la validation desdites fiches. Cela nécessite : la sélection de la fiche à valider, son remplissage, les possibilités de corriger indépendamment les frais forfait et les frais hors forfait, pour enfin, quand tout paraît correct à la comptable, procéder à la validation proprement dite.

1) Sélection de la fiche à valider

Le visuel est simple, et cohérent avec celui utilisé pour l'appli visiteurs :

Dans le contrôleur, je récupère toutes les fiches de frais à valider (idetat = 'CL'), et tous les mois concernés pour que l'utilisateur puisse choisir. Par contre, j'ai dû " nettoyer " les informations récupérées à partir de la bdd afin que chaque visiteur et chaque mois apparaissent une seule fois.

```
case 'selectionVisiteurMois':
    // Pour remplir les listes déroulantes visiteurs et mois :
    $lesVisiteurs = $pdo->getLesVisiteursAValider();
    if (!is_array($lesVisiteurs) || count($lesVisiteurs) === 0) {
        ajouterErreur("Il n'y a aucune fiche Visiteur à valider !");
        include 'vues/v_erreurs.php';
    } else {
    /**
    * On récupère les mois à valider pour tous les visiteurs
    * sélectionnés précédemment
    */
        foreach ($lesVisiteurs as $unVisiteur) {
            $lesMois[] = $unVisiteur['mois'];
        }
   /**
    * Et, bien sûr, on supprime les doublons, c'est à dire les mois qui apparaissent
    * plusieurs fois, pour qu'il ne reste qu'un 'exemplaire' de chaque mois
    * et de chaque visiteur
    */
        $lesMois = array_unique($lesMois);
        $visiteurASelectionner = $lesVisiteurs[0];
        $moisASelectionner = $lesMois[0];
        $lesVisiteurs = unique_multidim_array($lesVisiteurs, 'id');
    }
    include 'vues/v_selectionVisiteurMois.php';
    break;
```

Quand il s'agit d'un tableau 'simple', pour supprimer les doublons dans le langage php il existe la fonction array_unique, qui fait parfaitement son travail. Par contre, pour les visiteurs, c'est un tableau multidimentionnel il a donc fallu développer une fonction pour faire ce travail :

```php
function unique_multidim_array($array, $key)
{
    $temp_array = array();
    $i = 0;
    $key_array = array();

    foreach ($array as $val) {
        if (!in_array($val[$key], $key_array)) {
            $key_array[$i] = $val[$key];
            $temp_array[$i] = $val;
        }
        $i++;
    }
    return $temp_array;
}
```

J'avoue ne pas être l'auteure de cette fonction (même si j'aurai été capable de le faire) : développer, c'est aussi savoir utiliser le travail des autres. Cette fonction était donc proposée par un contributeur dans le manuel PHP. Je l'ai prise, l'ai testée, ait constaté qu'elle faisait parfaitement le travail que je voulais qu'elle fasse, je l'ai donc gardée.

Ainsi, dans les listes déroulantes, vous ne trouvez que des visiteurs qui ont en effet une/des fiche(s) à valider, et uniquement des mois pour lesquels il y a des fiches à valider. Vos chances de tomber sur le message d'erreur sont donc réduites au maximum.

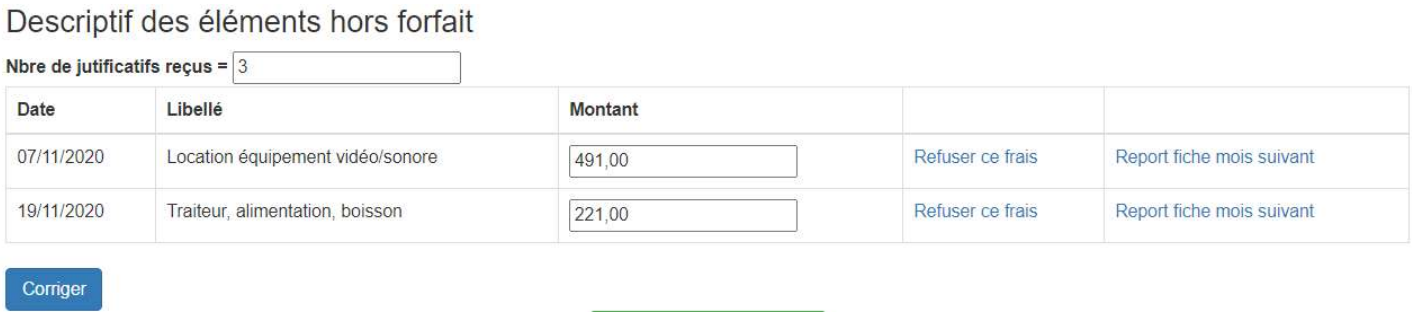
2) Remplissage de la fiche sélectionnée

La page de validation est découpée en plusieurs Vues, afin de permettre la souplesse nécessaire dans la manipulation par l'utilisateur.

- l'en-tête

- les frais forfait

- les frais hors forfait



- et enfin le bouton pour valider la fiche :

Un tel découpage était nécessaire, afin de pouvoir facilement effectuer les corrections nécessaires.

Pour le remplissage de tous ces éléments, il était important de pouvoir, de n'importe où, récupérer l'idVisiteur et le mois de la fiche concernée, puisque toutes les méthodes de sélection/modification de la base de données nécessitent ces 2 informations.

Pour éviter la répétition de code, j'indique dès le début au contrôleur que si l'utilisateur vient de sélectionner le visiteur et le mois dans les listes déroulantes, ces variables sont dans $_POST, sinon, elles se trouvent dans l'URL :

```
if (!empty($_POST['lstVisiteurs']) && !empty($_POST['lstMois'])) {
    $idVisiteur = $_POST['lstVisiteurs'];
    $leMois = $_POST['lstMois'];
} else {
    $idVisiteur = filter_input(INPUT_GET, 'idVisiteur', FILTER_SANITIZE_STRING);
    $leMois = filter_input(INPUT_GET, 'leMois', FILTER_SANITIZE_STRING);
}
```

Ensuite, il n'y a plus qu'à utiliser les méthodes appropriées pour récupérer toutes les informations dont on a besoin :

```
// Récupération des informations pour remplissage de la fiche de frais
    $lesFraisHorsForfait = $pdo->getLesFraisHorsForfait($idVisiteur, $leMois);
    $lesFraisForfait = $pdo->getLesFraisForfait($idVisiteur, $leMois);
    $etatFicheVisiteur = $pdo->getEtatFicheFrais($idVisiteur, $leMois);

    if (empty($etatFicheVisiteur)) {
        ajouterErreur("Il n'y a pas de fiche de frais pour ce visiteur pour le mois sélectionné");
        include 'vues/v_erreurs.php';
    } elseif ($etatFicheVisiteur != 'CL') {
        ajouterErreur("La fiche de ce mois n'est pas à valider pour ce visiteur");
        include 'vues/v_erreurs.php';
    } else {
        $lesInfosFicheFrais = $pdo->getLesInfosFicheFrais($idVisiteur, $leMois);
        $montantForfait = $pdo->calculFraisForfait($idVisiteur, $leMois);
        $montantHorsForfait = $pdo->calculFraisHorsForfait($idVisiteur, $leMois);
        $libEtat = $lesInfosFicheFrais['libEtat'];
        $enCours = $montantForfait + $montantHorsForfait;
        $nbJustificatifs = $lesInfosFicheFrais['nbJustificatifs'];
        $dateModif = dateAnglaisVersFrancais($lesInfosFicheFrais['dateModif']);
    }

    include 'vues/v_validationEntete.php';
    include 'vues/v_validationForfait.php';
    include 'vues/v_validationHorsForfait.php';
    include 'vues/v_validationFiche.php';
    break;
```

La plupart de ces méthodes faisaient partie de l'existant. J'ai dû ajouter simplement des méthodes de calcul de frais. Pour les frais forfait, il faut pouvoir multiplier la quantité des frais de la table lignefraisforfait par les montants de la table fraisforfait :

```
public function calculFraisForfait($idVisiteur, $mois)
    {
        $requetePrepare = PdoGsb::$monPdo->prepare(
            "SELECT montant, quantite "
                . "FROM gsb_lignefraisforfait "
                . "JOIN gsb_fraisforfait "
                . "ON gsb_lignefraisforfait.idfraisforfait = gsb_fraisforfait.id "
                . "WHERE gsb_lignefraisforfait.idvisiteur = :unVisiteur "
                . "AND gsb_lignefraisforfait.mois = :unMois"
        );
        $requetePrepare->bindParam(':unVisiteur', $idVisiteur, PDO::PARAM_STR);
        $requetePrepare->bindParam(':unMois', $mois, PDO::PARAM_STR);
        $requetePrepare->execute();
        $totalForfait = 0.00;
        while ($donnees = $requetePrepare->fetch()) {
            $totalForfait = $totalForfait + floatval($donnees['montant']) * floatval($donnees['quantite']);
        }
        return $totalForfait;
    }
```

Pour les frais hors forfait, c'est plus simple parce qu'il s'agit simplement d'une addition … il faut simplement ne pas oublier que, par la suite, on voudra refuser pouvoir refuser des frais.
Donc, au préalable, il nous faut une requête qui retourne un booléen, qui dit si le frais a été refusé ou non : puisque le libellé du frais est modifié en cas de refus, il suffit de vérifier si la ligne de libellé commence par REFUSE :

```
public function estRefuse($idFrais)
    {
        $requetePrepare = PdoGsb::$monPdo->prepare(
            "SELECT libelle "
                . "FROM gsb_lignefraishorsforfait "
                . "WHERE id= :idFrais"
        );
        $requetePrepare->bindParam(':idFrais', $idFrais, PDO::PARAM_STR);
        $requetePrepare->execute();
        $libelle = $requetePrepare->fetch();
        $libelleDebut = substr($libelle['libelle'], 0, 6);
        if ($libelleDebut == 'REFUSE') {
            return true;
        } else {
            return false;
        }
    }
```

Ainsi, dans la méthode qui fait l'addition des frais hors forfait, ceux qui sont refusés ne sont pas additionnés :

```
public function calculFraisHorsForfait($idVisiteur, $mois)
    {
        $requetePrepare = PdoGsb::$monPdo->prepare(
            "SELECT id, montant "
                . "FROM gsb_lignefraishorsforfait "
                . "WHERE idvisiteur = :unVisiteur "
                . "AND mois = :unMois"
        );
        $requetePrepare->bindParam(':unVisiteur', $idVisiteur, PDO::PARAM_STR);
        $requetePrepare->bindParam(':unMois', $mois, PDO::PARAM_STR);
        $requetePrepare->execute();
        $totalHorsForfait = 0.00;
        while ($donnees = $requetePrepare->fetch()) {
            if (!($this->estRefuse($donnees['id']))) {
                $totalHorsForfait = $totalHorsForfait + floatval($donnees['montant']);
            }
        }
        return $totalHorsForfait;
    }
```

3) La Correction des Frais (Forfait et Hors Forfait)

Les frais forfait et hors forfait sont traités indépendamment, mais finalement de la même manière.
Dans les 2 cas, lors du clic de l'utilisateur sur le bouton 'corriger', le programme récupère les informations du formulaire concerné, et les utilise pour la mise à jour de la bdd. Par exemple pour les frais hors forfait :

```
case 'corrigerHorsForfait':
    /* pour correction des frais hors forfait, on récupère l'id du frais dans
     * l'URL, et on met à jour la bdd avant de revenir à la fiche */
    $lesFraisHorsForfait = filter_input(
        INPUT_POST,
        'horsForfait',
        FILTER_DEFAULT,
        FILTER_FORCE_ARRAY
    );
    $nbJustificatifs = filter_input(INPUT_POST, 'justifs', FILTER_SANITIZE_NUMBER_INT);
    $pdo->majFraisHorsForfait($idVisiteur, $leMois, $lesFraisHorsForfait);
    $pdo->majNbJustificatifs($idVisiteur, $leMois, $nbJustificatifs);
    header('Location:index.php?uc=validation&action=valider&idVisiteur=' . $idVisiteur . '&leMois=' . $leMois);
    break;
```

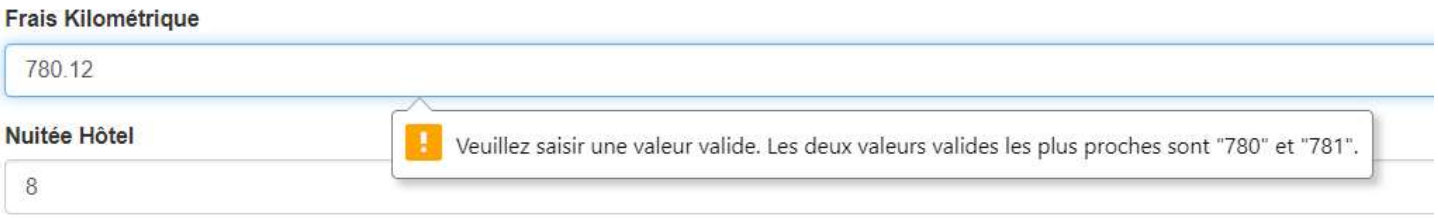
Pour la correction des frais forfaitisés, le traitement est similaire.

Une précision : dans le programme fournit, il existait une fonction qtéFraisValide(), qui vérifiait que les quantités de frais saisies par l'utilisateur étaient des nombres entiers.

Je n'ai pas utilisé cette fonction : j'ai jugé plus simple de gérer ce type de contrainte côté utilisateur, grâce aux possibilités offertes par html 5. En effet, en utilisant une balise input de type number (au lieu de text), on peut déterminer une valeur minimum, ainsi que le step c'est à dire la première valeur possible suivante.

```
<?php
        foreach ($lesFraisForfait as $unFrais) {
            $idFrais = $unFrais['idfrais'];
            $libelle = htmlspecialchars($unFrais['libelle']);
            $quantite = $unFrais['quantite']; ?>
            <div class="form-group">
                <label for="idFrais"><?php echo $libelle; ?></label>
                <input type="number" id="idFrais"
                       name="lesFrais[<?php echo $idFrais; ?>]"
                       min="0" step="1"
                       value="<?php echo $quantite; ?>"
                       class="form-control">
            </div>
            <?php
        }
        ?>
```

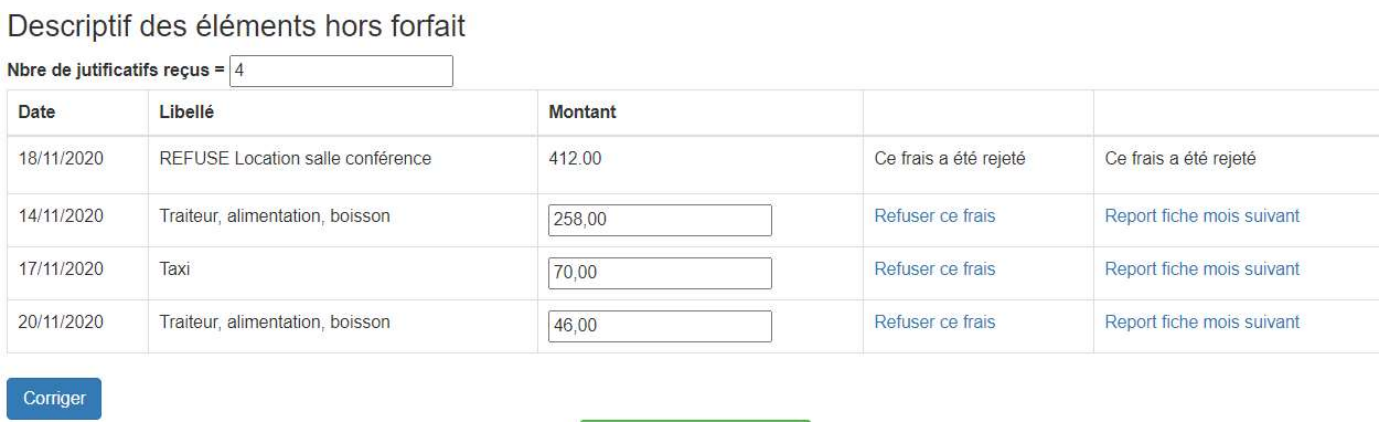
Ainsi, dans ce cas, avec un step à 1, l'utilisateur ne peut saisir que des nombres entiers :



J'ai procédé de même pour le nombre de justificatifs des frais hors forfaits, et pour les montants j'ai simplement déterminé un step de 0,01 (il peut y avoir des décimales – mais pas de montant négatif).

4) Supprimer / Reporter un frais hors forfait

Pour un affichage correcte pour l'utilisateur, il est important qu'une fois qu'un frais est refusé, on ne puisse plus accéder au montant, ni aux options de refus (en cas de report la ligne disparaît, donc pas de soucis) :



Pour ça, j'utilise de nouveau la méthode booléenne vue plus haut, qui me dit si une ligne de frais hors forfait a été refusée : de cette manière, l'affichage de la ligne est différent si le frais a été refusé ou non.

```
<?php
        foreach ($lesFraisHorsForfait as $unFraisHorsForfait) {
            $libelle = verifInput($unFraisHorsForfait['libelle']);
            $date = $unFraisHorsForfait['date'];
            $montant = $unFraisHorsForfait['montant'];
            $id = $unFraisHorsForfait['id'];
            $estRefuseFrais = $pdo->estRefuse($id);
            ?>
            <tr>
                <td> <?php echo $date ?></td>
                <td> <?php echo $libelle ?></td>
                <td><?php if ($estRefuseFrais) {
                    echo $montant;
                    } else { ?>
                    <input type="number" step="0.01" name="horsForfait[<?php echo $id; ?>]"
                        value="<?php echo $montant ?>">
                </td>
                    <?php
                }
                ?>
                <td><?php if ($estRefuseFrais) {?>
                    <p>Ce frais a été rejeté</p>
                    <?php
                    } else {
                        ?>
                    <a href="index.php?uc=validation&action=supprimerFrais&idVisiteur=<?php echo $idVisiteur; ?
>&leMois=<?php echo $leMois; ?>&idFrais=<?php echo $id;?>"
                        onclick="return confirm
                            ('Voulez-vous vraiment refuser ce frais ?');">
                        Refuser ce frais</a>
                    </td>
                    <?php } ?>
                <td><?php if ($estRefuseFrais) {?>
                    <p>Ce frais a été rejeté</p>
                    <?php
                    } else {
                        ?>
                    <a href="index.php?uc=validation&action=reporterFrais&idVisiteur=<?php echo $idVisiteur; ?>&leMois=<?
php echo $leMois; ?>&idFrais=<?php echo $id;?>"
                        onclick="return confirm
                            ('Voulez-vous vraiment reporter ce frais ?');">
                        Report fiche mois suivant</a></td>
                        <?php
                    }
                    ?>
            </tr>
            <?php
        }
        ?>
```

Au niveau du contrôleur, pour refuser un frais, conformément aux instructions données, il faut donc ajouter 'REFUSE' au début du libellé de la ligne :

```
case 'supprimerFrais':
    $idFrais = filter_input(INPUT_GET, 'idFrais', FILTER_SANITIZE_STRING);
    $pdo->refuserFraisHorsForfait($idFrais);
    header('Location:index.php?uc=validation&action=valider&idVisiteur=' . $idVisiteur . '&leMois=' . $leMois);
    include 'vues/v_validation.php';
    break;
```

Pour ce faire, j'ai donc créé la méthode appropriée, refuserFraisForfait : celle-ci extrait de la base le libellé du frais dont l'id est passé en paramètre, puis modifie le libellé en ajoutant 'REFUSE', et enfin met à jour la bdd avec ce nouveau libellé :

```
public function refuserFraisHorsForfait($idFrais)
    {
        $requetePrepare = PdoGsb::$monPdo->prepare(
            "SELECT libelle "
                . "FROM gsb_lignefraishorsforfait "
                . "WHERE id= :idFrais"
        );
        $requetePrepare->bindParam(':idFrais', $idFrais, PDO::PARAM_STR);
        $requetePrepare->execute();
        $libelleOriginal = $requetePrepare->fetch();

        $libelleModifie = 'REFUSE  ' . $libelleOriginal['libelle'];
        $requeteModifie = PdoGsb::$monPdo->prepare(
            "UPDATE gsb_lignefraishorsforfait "
                . "set libelle = :nouveauLibelle "
                . "WHERE id = :idFrais"
        );
        $requeteModifie->bindParam(':nouveauLibelle', $libelleModifie, PDO::PARAM_STR);
        $requeteModifie->bindParam(':idFrais', $idFrais, PDO::PARAM_STR);
        $requeteModifie->execute();
    }
```

Pour le report du frais cette fois, puisqu'il faut le reporter sur la fiche du mois suivant (qui existe dans 100 % des cas, puisque dès qu'une fiche est clôturée la fiche du mois suivant est créée), il m'a fallu d'abord faire une fonction qui retourne le mois suivant d'un mois passé en paramètre : un joli petit algorithme, puisqu'il prend le mois en format aaaamm, découpe pour obtenir l'année et le mois, convertit ces variables en format integer pour pouvoir ajouter 1 mois, puis reconvertir en format string, ajoute un 0 si c'est un mois à 1 seul chiffre, et enfin concatène le résultat obtenu :

```
function getLeMoisSuivant($mois)
{
    $numAnnee = substr($mois, 0, 4);
    $numMois = substr($mois, 4, 2);
    $numAnnee = intval($numAnnee);
    $numMois = intval($numMois);
    if ($numMois === 12) {
        $numMois = 1;
        $numAnnee += 1;
    } else {
        $numMois += 1;
    }
    $numMois = strval($numMois);
    $numAnnee = strval($numAnnee);
    if (strlen($numMois === 1)) {
        return $numAnnee . '0' . $numMois;
    } else {
        return $numAnnee . $numMois;
    }
}
```

Une fois qu'on a ainsi obtenu le mois suivant, il est aisé de développer la méthode qui reportera un frais hors forfait :

```
public function reporterFraisHorsForfait($idFrais, $mois)
    {
        $moisSuivant = getLeMoisSuivant($mois);
        $requetePrepare = PdoGsb::$monPdo->prepare(
            "UPDATE gsb_lignefraishorsforfait "
                . "SET mois = :unMois "
                . "WHERE id = :idFrais"
        );
        $requetePrepare->bindParam(':unMois', $moisSuivant, PDO::PARAM_STR);
        $requetePrepare->bindParam(':idFrais', $idFrais, PDO::PARAM_STR);
        $requetePrepare->execute();
    }
```

5) Validation de la fiche de frais

Une fois que l'utilisateur a corrigé tout ce qu'il souhaitait, il valide la fiche concernée.
Dans le contrôleur :

```
case 'succesValidation':
    /* maintenant que tout est OK, on valide, donc :
        * - on met à jour le montant validé dans la fiche de frais
        * - on passe l'état de la fiche de frais à VA pour validée
     */
    $montantForfait = $pdo->calculFraisForfait($idVisiteur, $leMois);
    $montantHorsForfait = $pdo->calculFraisHorsForfait($idVisiteur, $leMois);
    $enCours = $montantForfait + $montantHorsForfait;
    $pdo->validerFrais($idVisiteur, $leMois, $enCours);
    $pdo->majEtatFicheFrais($idVisiteur, $leMois, 'VA');
    include 'vues/v_succesValidation.php';
    header('Refresh:5 ; URL=index.php?uc=validation&action=selectionVisiteurMois');
    break;
default:
    include 'vues/v_accueil.php';
}
```

Rien de vraiment nouveau, la plupart des méthodes ont déjà été utilisées, et la méthode validerFrais consiste simplement à ajouter dans la bdd, dans la table fichefrais, le montant à valider.

VI LE PAIEMENT DES FICHES DE FRAIS

1) Vérification et correction des informations de la base de données

L'exécution de la création d'un jeu test, donc, insère dans la base de données des fiches en cours de création, et des fiches déjà validées et qui sont donc à payer.
Cependant, il s'avère que ces fiches, déjà validées, comportent des incohérences, qu'il me fallait corriger :
- le montant total validé dans la fiche de frais n'est pas égal au frais forfait + frais hors forfait (il semble que ce soit un montant aléatoire qui soit généré), il faut donc vérifier et corriger la base en conséquences :

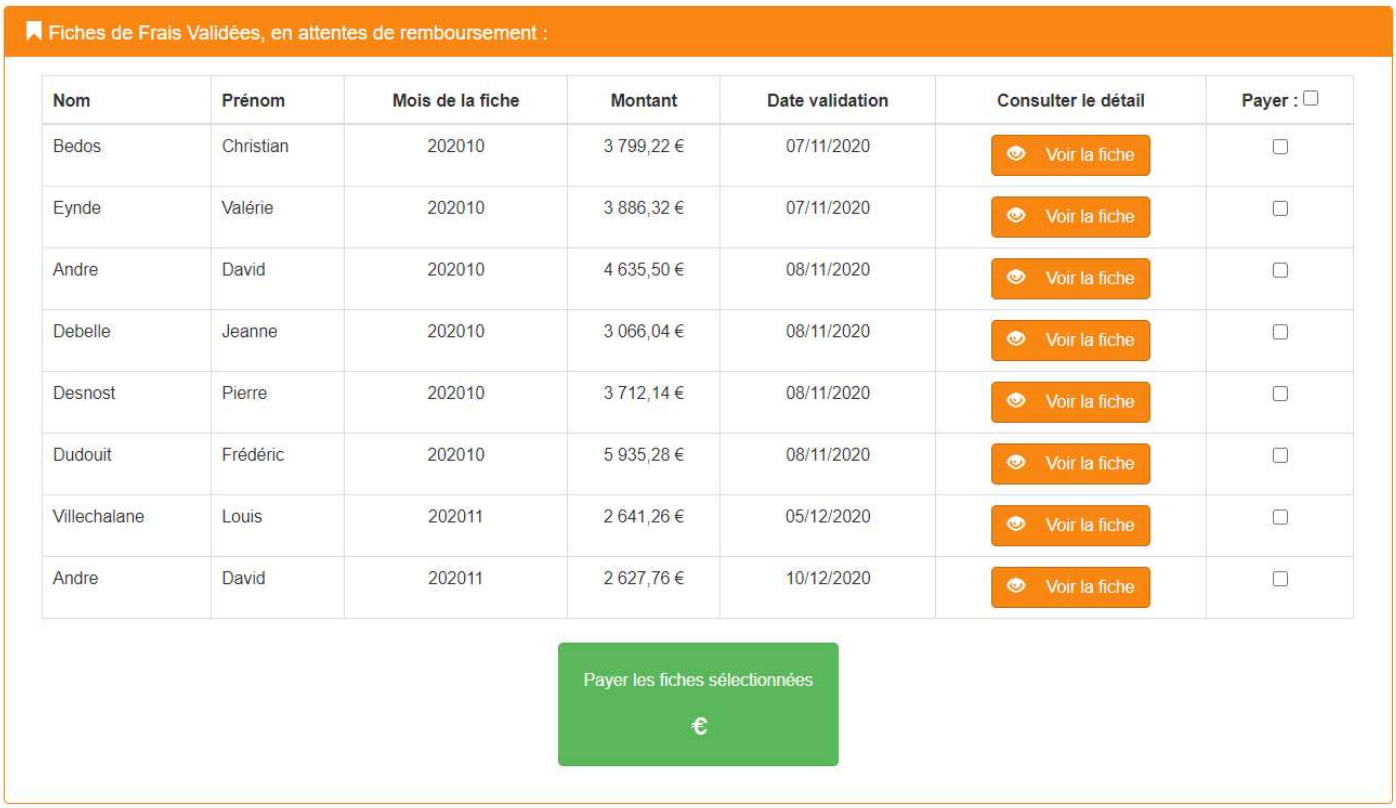
```
public function correctionAutomatiqueMontantValide()
    {
        $lesFichesVisiteurs = $this->getLesFichesVisiteursAPayer();
        foreach ($lesFichesVisiteurs as $uneFicheVisiteur) {
            $idVisiteur = $uneFicheVisiteur['id'];
            $mois = $uneFicheVisiteur['mois'];
            $montantValide = $uneFicheVisiteur['montant'];
            $montantForfait = $this->calculFraisForfait($idVisiteur, $mois);
            $montantHorsForfait = $this->calculFraisHorsForfait($idVisiteur, $mois);
            $montantTotal = $montantForfait + $montantHorsForfait;

            if ($montantValide !== $montantTotal) {
                $requetePrepare = PdoGsb::$monPdo->prepare(
                    "UPDATE gsb_fichefrais "
                    . "SET montantvalide = :montant "
                    . "WHERE idvisiteur = :idVisiteur AND mois = :unMois"
                );
                $requetePrepare->bindParam(':montant', $montantTotal, PDO::PARAM_STR);
                $requetePrepare->bindParam(':idVisiteur', $idVisiteur, PDO::PARAM_STR);
                $requetePrepare->bindParam(':unMois', $mois, PDO::PARAM_STR);
                $requetePrepare->execute();
            }
        }
    }
```

- de la même manière, le nombre de justificatifs est aussi inséré de manière aléatoire : on peut trouver 11 justificatifs pour 3 lignes de frais hors forfaits. J'ai donc procédé à la vérification et à la correction de la base :

```
public function correctionAutomatiqueNbJustificatifs()
    {
        $lesFichesVisiteurs = $this->getLesFichesVisiteursAPayer();
        foreach ($lesFichesVisiteurs as $uneFicheVisiteur) {
            $idVisiteur = $uneFicheVisiteur['id'];
            $mois = $uneFicheVisiteur['mois'];
            $nbJustificatifs = $uneFicheVisiteur['nbjustificatifs'];
            $fraisHorsForfait = $this->getLesFraisHorsForfait($idVisiteur, $mois);

            if (count($fraisHorsForfait) < $nbJustificatifs) {
                $requetePrepare = PdoGsb::$monPdo->prepare(
                    "UPDATE gsb_fichefrais "
                    . "SET nbjustificatifs = :nbJustificatifs "
                    . "WHERE idvisiteur = :idVisiteur AND mois = :unMois"
                );
                $requetePrepare->bindParam(':nbJustificatifs', count($fraisHorsForfait), PDO::PARAM_STR);
                $requetePrepare->bindParam(':idVisiteur', $idVisiteur, PDO::PARAM_STR);
                $requetePrepare->bindParam(':unMois', $mois, PDO::PARAM_STR);
                $requetePrepare->execute();
            }
        }
    }
```

2) Le paiement des fiches

Côté utilisateur, j'ai choisi de lui proposer un tableau, qui reprend toutes les fiches de la base qui sont validées, avec un bouton pour visualiser la fiche de frais concernée :



Pour un meilleur confort d'utilisation, la case à cocher à côté de " Payer " permet de cocher/décocher toutes les cases. Pour cela, j'ai utilisé une petite fonction JavaScript qui fait le travail :

```
function cocher_tout()
{
   var chb = document.getElementById('monform').getElementsByTagName("input");
   if (chb.length > 1) { // s'il y a d'autres input que "Sélectionner tout" et "Supprimer"
      // si la case "Sélectionner tout" est cochée
      if (document.getElementById('tout').checked === true) {
         for (var i = 0; i < chb.length; i++) {
            if (chb[i].name.substr(0, 6) === "aPayer") {
               chb[i].checked = true;
            } // si le name du checkbox commence par "aPayer"
         }
      } else {  // si la case "Sélectionner tout" est décochée
         for (var i = 0; i < chb.length; i++) {
            if (chb[i].name.substr(0, 6) === "aPayer") {
               chb[i].checked = false;
            }
         }
      }
   } else { // il n'y a pas de checkbox
      return;
   }
}
```

Côté contrôleur, pour la visualisation de la fiche de frais les méthodes utilisées sont similaires à ce qui est fait dans la partie validation, je ne vais donc pas les répéter ici.

Enfin, une fois que l’utilisateur clique sur le bouton “ Payer les fiches sélectionnées ” :

```
case 'payer':
    // récupération des cases cochées => maj bdd des fiches payées
    $fichesARembourser = $pdo->getLesFichesVisiteursAPayer();
    $montantVirement = 0;
    foreach ($_POST['aPayer'] as $virement) {
        $posId = stripos($virement, '&');
        $idVisiteur = substr($virement, 0, $posId);
        $moisPaye = substr($virement, $posId + 1);
        $montantAPayer = getLeMontantAPayer($fichesARembourser, $idVisiteur, $moisPaye);
        $montantVirement += $montantAPayer;
        $pdo->majEtatFicheFrais($idVisiteur, $moisPaye, 'RB');
    }
    include 'vues/v_succesPaiement.php';
    break;
```

VII HASHAGE DES MOTS DE PASSE

Comme précisé en préambule, je n'ai pas touché à l'existant. Je n'ai donc pas touché à l'appli visiteur, et, même si j'ai en effet développé les méthodes nécessaires au hashage des mots de passe visiteurs, je ne les ait pas intégré dans l'appli visiteurs.

J'ai donc uniquement hashé les mots de passe pour les comptables.

Pour l'algorithme de hashage, il y a de multiples possibilités. Déjà, j'ai éliminé des algorithmes comme MD5, SHA1 et SHA256 : ils sont rapides oui, mais considérant les évolutions technologiques il est devenu facile de les attaquer.

D'après la documentation de PHP, il nous reste donc les algorithmes suivants :

Les algorithmes suivants sont actuellement supportés :

- **PASSWORD_DEFAULT** - Utilisation de l'algorithme bcrypt (par défaut depuis PHP 5.5.0). Notez que cette constante est concue pour changer dans le temps, au fur et à mesure que des algorithmes plus récents et plus forts sont ajoutés à PHP. Pour cette raison, la longueur du résultat issu de cet algorithme peut changer dans le temps, il est donc recommandé de stocker le résultat dans une colonne de la base de données qui peut contenir au moins 60 caractères (255 caractères peut être un très bon choix).
- **PASSWORD_BCRYPT** - Utilisation de l'algorithme **CRYPT_BLOWFISH** pour créer la clé de hachage. Ceci va créer une clé de hachage standard crypt() utilisant l'identifiant "$2y$". Le résultat sera toujours une chaîne de 60 caractères, ou **false** si une erreur survient.
- **PASSWORD_ARGON2I** - Utilise l'algorithme de hachage Argon2i pour créer le hachage. Cet algorithme est seulement disponible si PHP a été compilé avec le support d'Argon2
- **PASSWORD_ARGON2ID** - Utilise l'algorithme de hachage Argon2id pour créer le hachage. Cet algorithme est seulement disponible si PHP a été compilé avec le support d'Argon2

Puisque PASSWORD_DEFAULT est susceptible de changer dans le temps, j'ai éliminé aussi cet algorithme.

Au départ, j'avais donc choisi Argon2i. Sauf que, une fois le code uploadé sur mon serveur distant, je me suis aperçue que celui-ci n'avait pas la configuration requise.

De fait, j'ai choisi l'algorithme CRYPT_BLOWFISH

Pour le hashage proprement dit, j'ai besoin de 2 méthodes :
- la première qui me retourne la liste des comptables et de leur mot de passe :

```
public function getLesMotsDePasseComptables()
    {
        $requetePrepare = PdoGsb::$monPdo->prepare(
            "SELECT id, mdp "
                . "FROM gsb_comptable "
        );
        $requetePrepare->execute();
        while ($donnees = $requetePrepare->fetch()) {
            $mdp[] = array(
                'id' => $donnees['id'],
                'mdp' => $donnees['mdp']
            );
        }
        return $mdp;
    }
```

- et la deuxième qui prend en paramètre un idComptable et un mot de passe, et qui hashe le mot de passe dans la base de données :

```php
public function setMdpHashComptables($idComptable, $mdp)
    {
        $requetePrepare = PdoGsb::$monPdo->prepare(
            "UPDATE gsb_comptable "
                . "SET mdp = :motHash "
                . "WHERE id = :id "
        );
        $mdp = password_hash($mdp, PASSWORD_BCRYPT);
        $requetePrepare->bindParam(':motHash', $mdp);
        $requetePrepare->bindParam(':id', $idComptable, PDO::PARAM_STR);
        $requetePrepare->execute();
    }
```

Ensuite, armée de ces méthodes, je peux faire le nécessaire dans le contrôleur de la connexion, c'est à dire : vérifier si le mdp stocké est hashé, s'il ne l'est pas alors hashage, puis contrôle que le comptable identifié avec son login a saisi le bon mot de passe, grâce à la fonction native de php password_verify, et ensuite création du cookie de session :

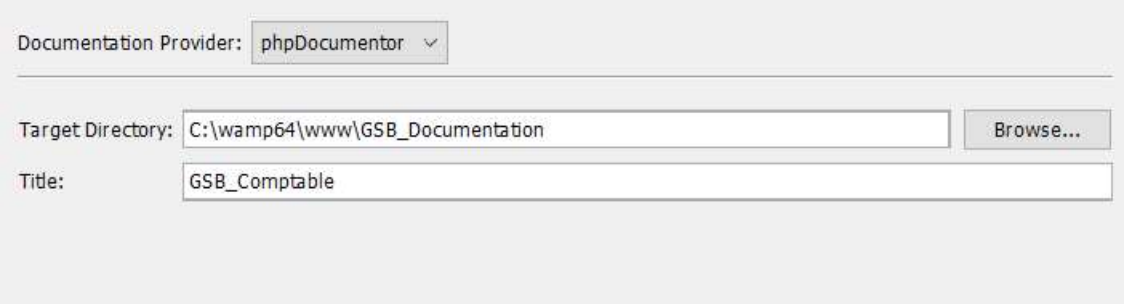
```php
    case 'valideConnexion':
        /**
         * Vérification que les mdp sont hachés. Sinon, hashage.
         */
        $motsDePasses = $pdo->getLesMotsDePasseComptables();

        foreach ($motsDePasses as $comptable) {
            $mot = $comptable['mdp'];
            $id = $comptable['id'];
            $algo = substr($mot, 0, 4);
            if ($algo !== '$2y$') {
                $pdo->setMdpHashComptables($id, $mot);
            }
        }
        // on récupère les infos saisie par l'utilisateur
        $login = verifInput($_POST['login']);
        $mdp = verifInput($_POST['mdp']);

        // on récupère le comptable identifié par son login
        $comptable = $pdo->getInfosComptable($login);

        // s'il n'y a pas de comptable avec ce login : msq erreur
        if (!is_array($comptable)) {
            ajouterErreur('Login ou mot de passe incorrect');
            include 'vues/v_erreurs.php';
            include 'vues/v_connexion.php';
        } else {
            // sinon, on verifie le mdp hashé
            $mdpOk = password_verify($mdp, $comptable['mdp']);
            // si le mdp est OK, création du cookie de session
            if ($mdpOk) {
                $id = $comptable['id'];
                $nom = $comptable['nom'];
                $prenom = $comptable['prenom'];
                connecter($id, $nom, $prenom);
                header('Location: index.php');
            } else {
                // sinon msq erreur + retour page accueil
                ajouterErreur('Login ou mot de passe incorrect');
                include 'vues/v_erreurs.php';
                include 'vues/v_connexion.php';
            }
        }
        break;
```

VIII PRODUCTION DE LA DOCUMENTATION

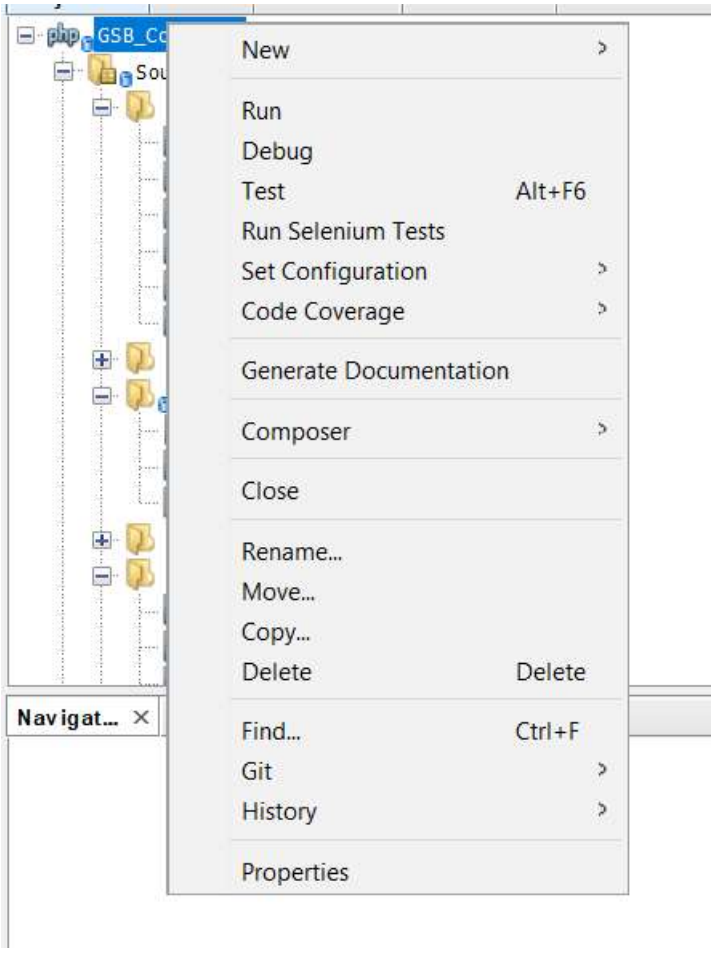
Comme conseillé dans les normes de développement, j'ai installé et utilisé phpDocumentor pour la génération automatique de la documentation.

J'ai choisi de générer cette documentation dans un répertoire à part, et donc de l'héberger sur un sous-domaine dédié.

Il suffit de configurer correctement l'IDE, et en particulier le répertoire dédié à la doc :



Ensuite … une fois le projet terminé, il n'y a plus qu'à cliquer sur Generate Documentation :



Tous les fichiers nécessaires sont alors générés automatiquement, et la page s'ouvre dans le navigateur par défaut.