16/11/2016

Documentation Application GSB FRAIS

Laetitia MONSERRAT / Victor DIANA / ROMAIN STEFANELLI

ACTIVITES DU REFERENTIEL MISES EN OEUVRE

A1.1.1 Analyse du cahier des charges d'un service à produire
A1.4.1 Participation à un projet
A4.1.2 Conception ou adaptation de l'interface utilisateur d'une solution applicative
A4.1.7 Développement, utilisation ou adaptation de composants logiciels
A4.1.9 Rédaction d'une documentation technique
A5.1.1 Mise en place d'une gestion de configuration

Sommaire

I/ Contexte de la situation professionnelle

A/ Le besoin et les objectifs

La force commerciale d'un laboratoire pharmaceutique est assurée par un travail de conseil et d'information auprès des prescripteurs. C'est pourquoi il existe des délégués (ou visiteurs médicaux) qui visitent les professionnels de la santé susceptibles de prescrire aux patients les produits médicaux, afin d'actualiser et rafraîchir leurs connaissances sur les produits de l'entreprise. L'intervention des visiteurs médicaux a donc un impact certain sur la prescription de la pharmacopée du laboratoire.

Ainsi, nous allons modifier une application qui a pour but uniformiser la gestion du suivi des frais.

L'application doit permettre d'enregistrer tout frais engagé, aussi bien pour l'activité directe (déplacement, restauration et hébergement) que pour les activités annexes (événementiel, conférences, autres), et de présenter un suivi daté des opérations menées par le service comptable (réception des pièces, validation de la demande de remboursement, mise en paiement, remboursement effectué).

Pour cela, nous allons rajouter une entité Administrateur qui pourra créer et modifier les visiteurs ainsi qu'une entité comptable qui pourra consulter les fiches de frais des visiteurs.

L'application Web destinée aux visiteurs, délégués et responsables de secteur sera en ligne, accessible depuis un ordinateur.

La partie utilisée par les services comptables sera aussi sous forme d'une interface Web.

Le module accessible à la force de visite sera intégré à l'application de gestion des comptes rendus de visite, sous forme d'une interface spécifique.

L'environnement sera accessible aux seuls acteurs de l'entreprise.
Une authentification préalable sera nécessaire pour l'accès au contenu.
Tous les échanges produits seront cryptés par le serveur Web.

L'application devra permettre de produire automatiquement l'équivalent de ces fiches de manière à les mettre à disposition des visiteurs pour la saisie en ligne.

B/ L'existant

Actuellement, nous sommes en possession d'une application fonctionnelle grâce à laquelle on peut se connecter en tant que visiteur ainsi que voir et renseigner nos propres fiches de frais. Au plus tard le 20 de chaque mois, le service comptable adresse aux visiteurs la fiche de demande de remboursement pour le mois en cours. L'application devra permettre de produire automatiquement l'équivalent de ces fiches de manière à les mettre à disposition des visiteurs pour la saisie en ligne.

II/ Environnement de développement

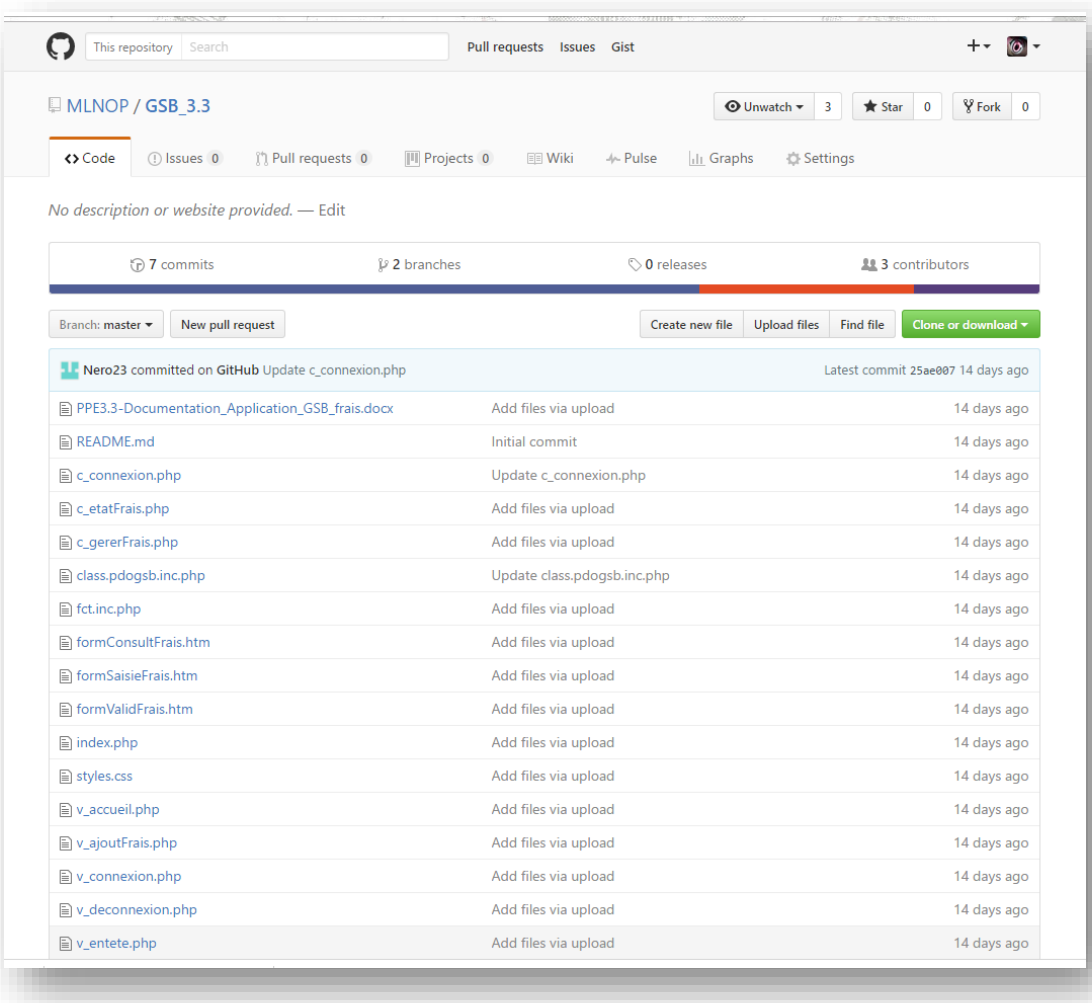
Pour notre application nous avons utilisé :

Langages :

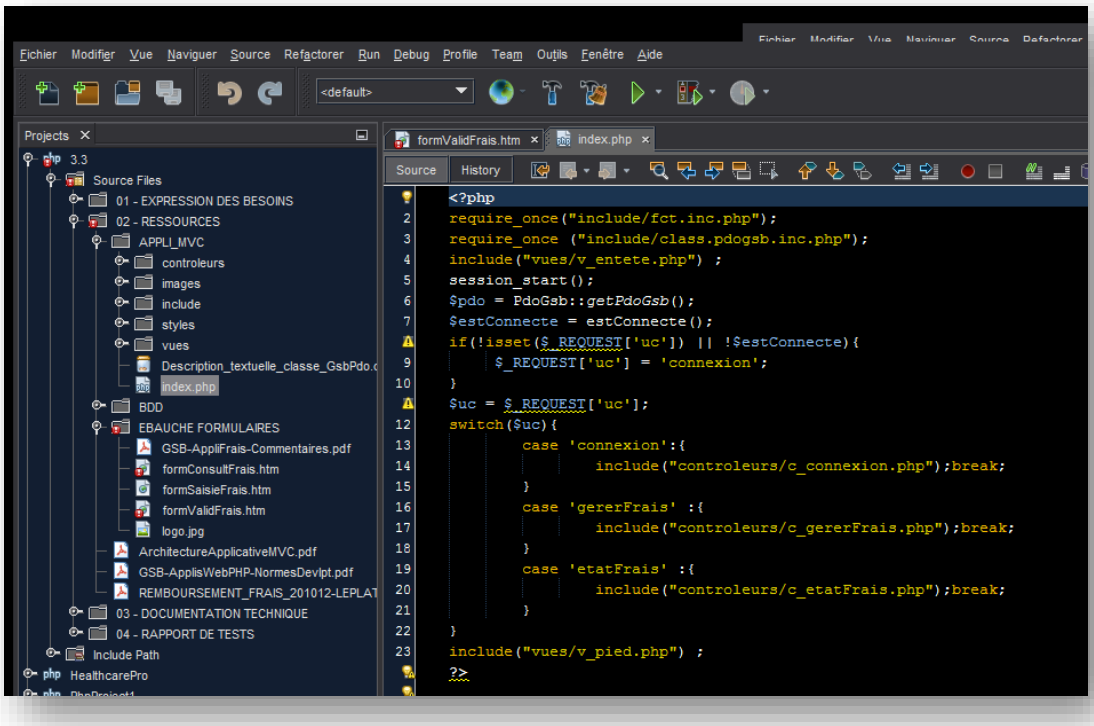
- CSS
- HTML
- PHP
- SQL

Programmes :

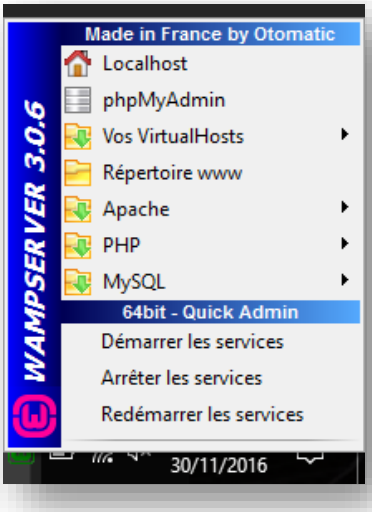
- GitHub pour créer un dépôt distant disponible en ligne pour tous les membres du projet



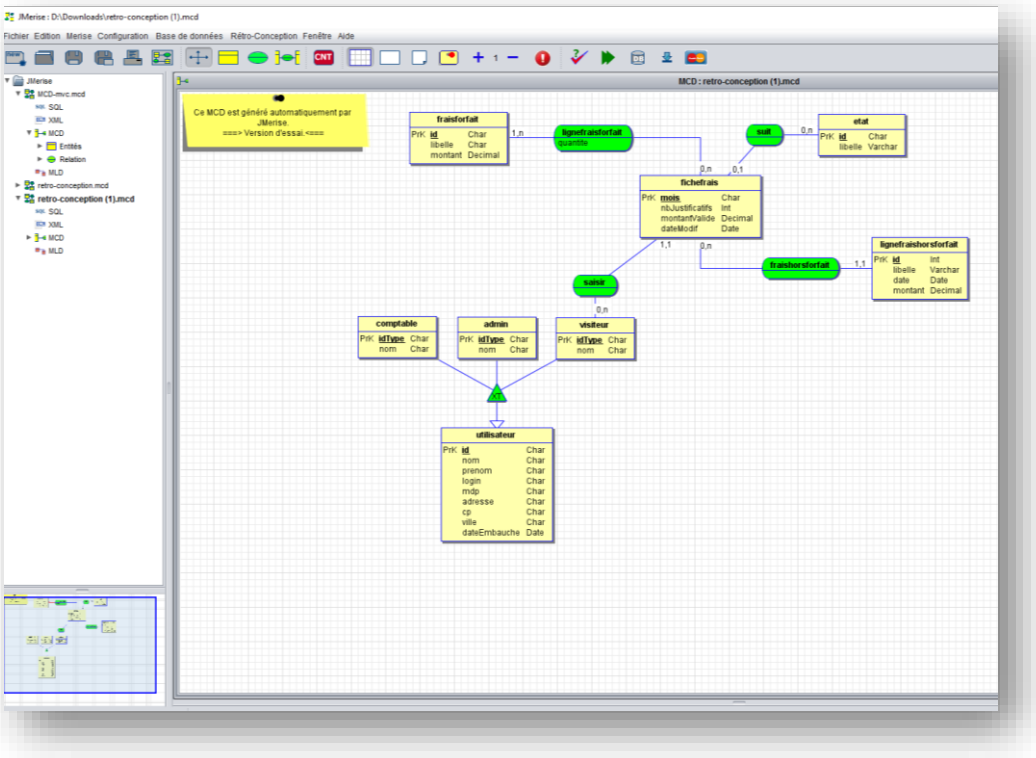
- l'IDE NetBeans pour créer le code source



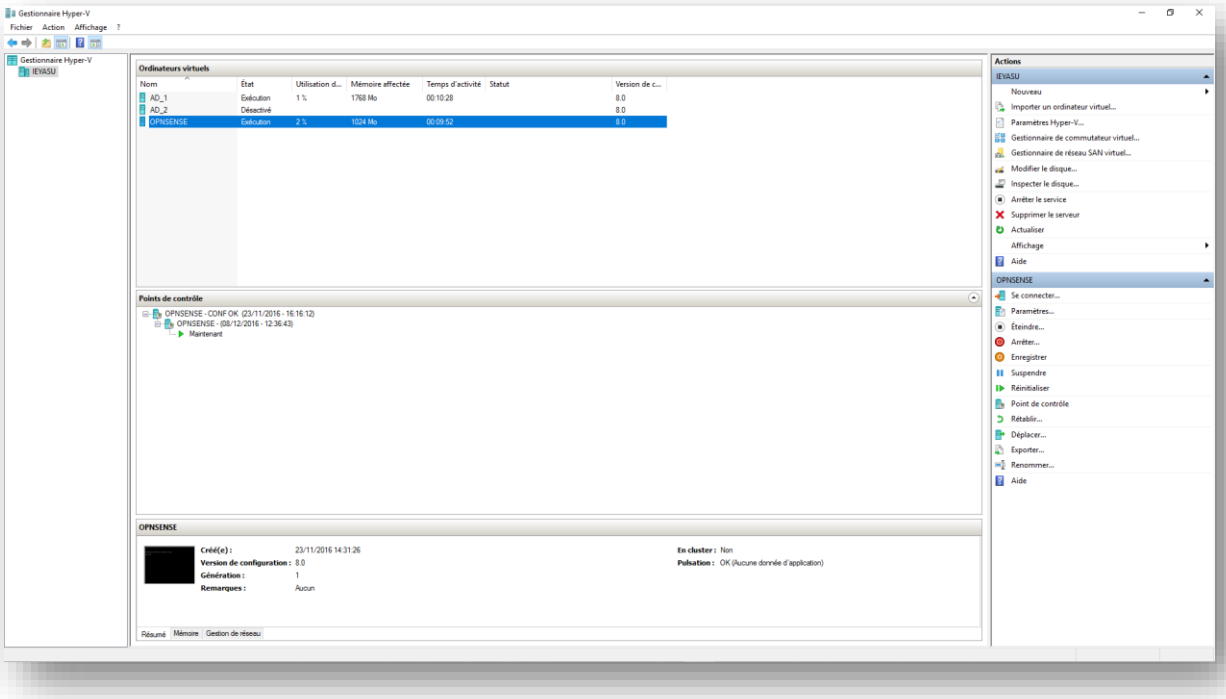
- Wampserver en tant qu'environnement afin de pouvoir visualiser l'application :



- JMerise pour modéliser les bases de données



-HyperV pour créer une machine virtuelle afin d'héberger l’application :



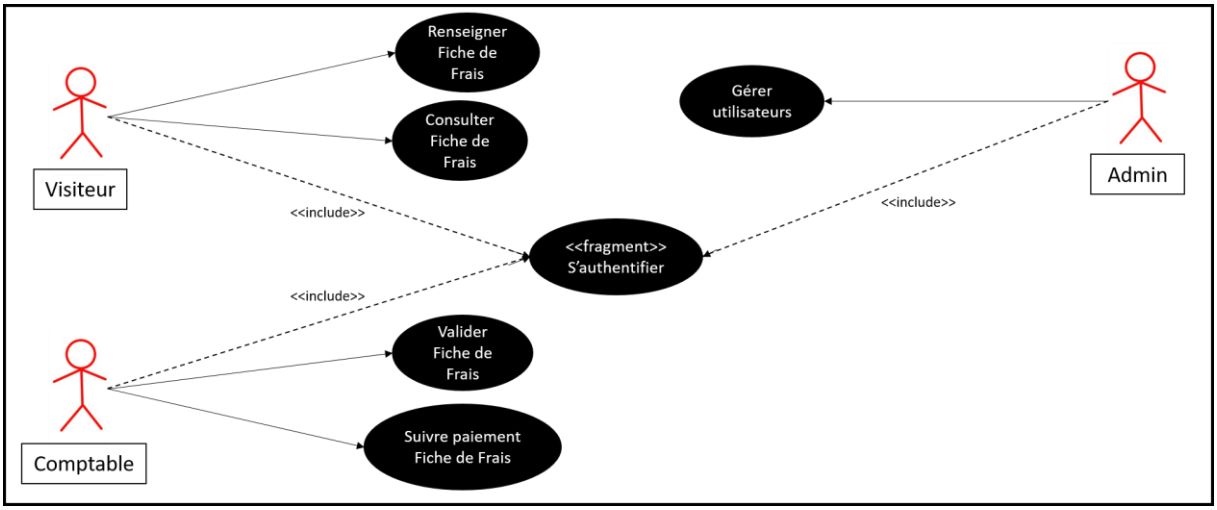
- Visio professional pour créer le diagramme de classe PdoGsb :

Nature du projet :

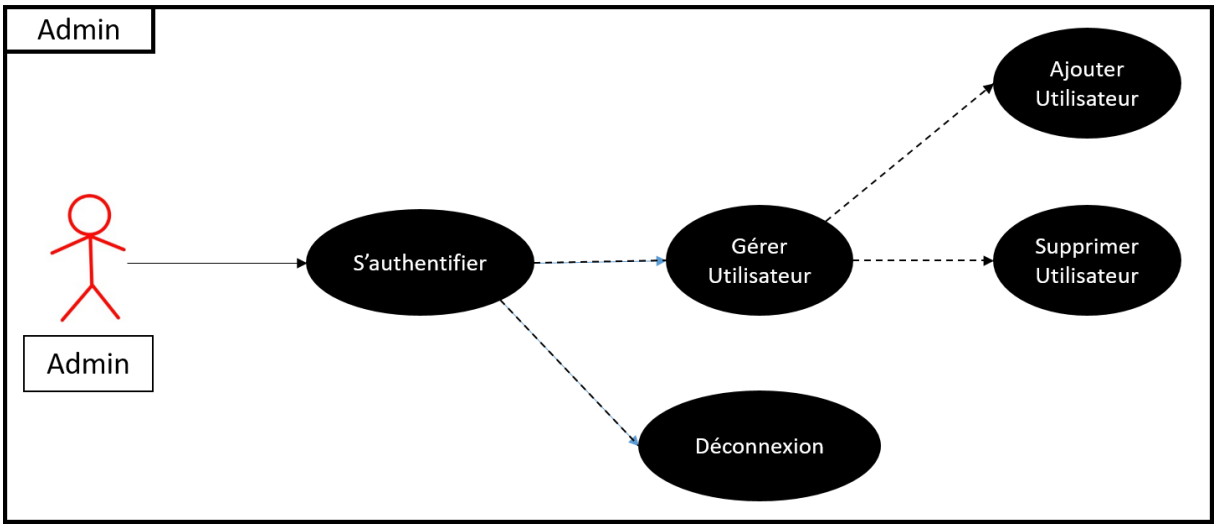
Le Projet consiste à modifier une application client léger qui a pour but uniformiser la gestion du suivi des frais.
L'environnement sera accessible aux seuls acteurs de l'entreprise.
Nous allons développer une interface comptable dans une architecture Modèle-Vue-Contrôleur, afin qu'une fois authentifiés, ils puissent valider les fiches de frais et en suivre le paiement.
Nous intégrerons la bibliothèque de classes Bootstrap dans le code source de l'application.

Méthodologie choisie :

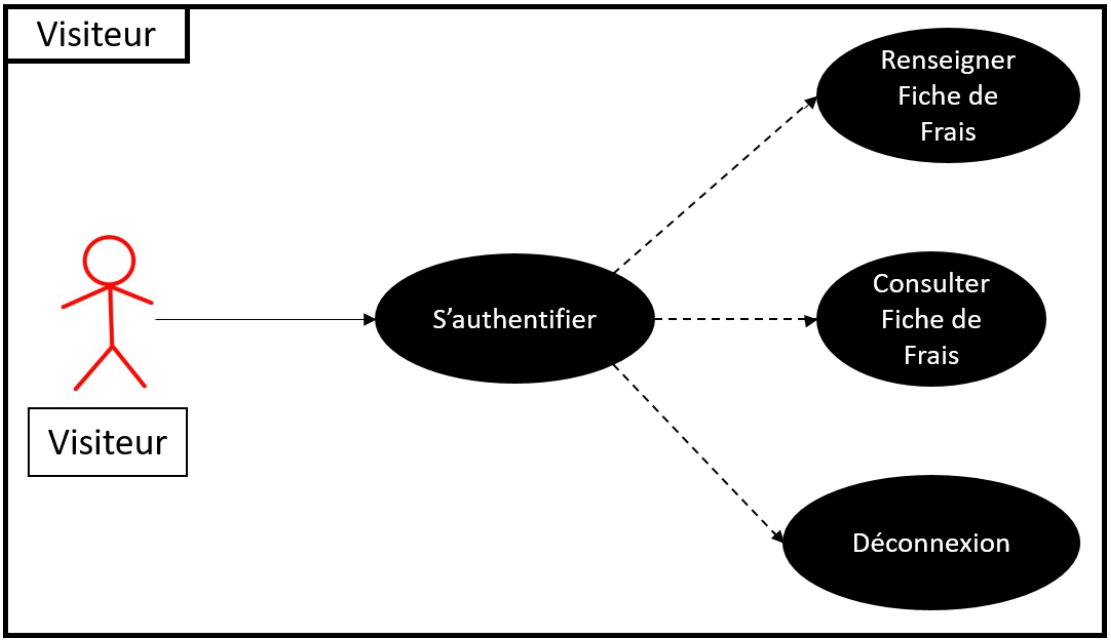
- Use case global :



- Use case Admin :

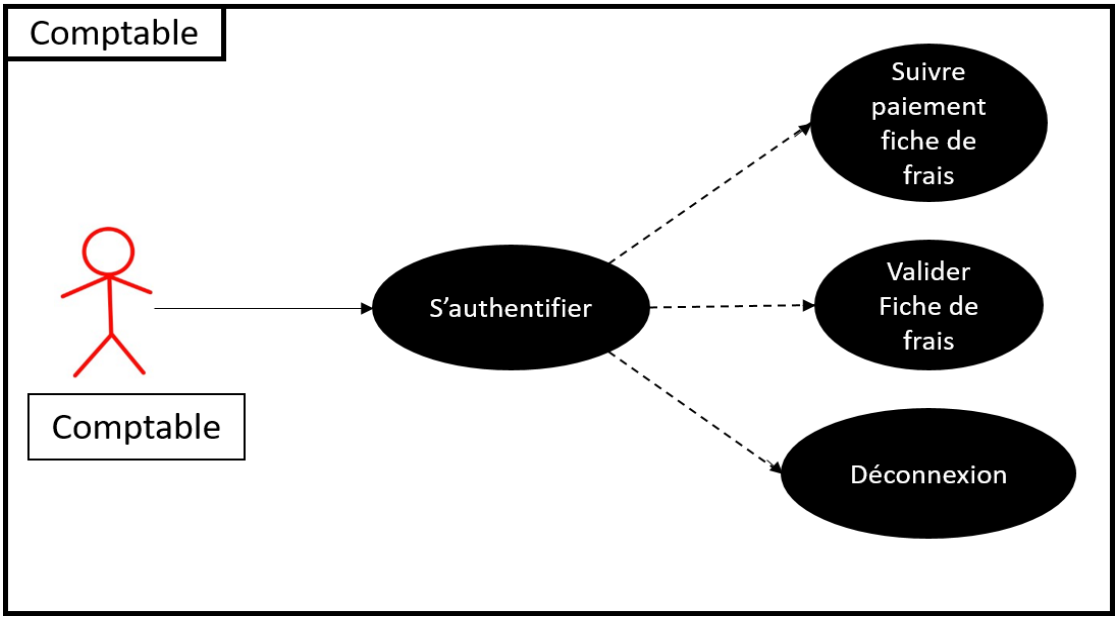


-Use case Visiteur :



Après authentification grâce aux identifiants à leur disposition, les visiteurs saisissent les quantités de frais forfaitisés et les frais hors forfait engagés pour le mois écoulé.
Ils ont accès en modification à la fiche tout au long du mois et peuvent y ajouter de nouvelles données ou supprimer des éléments saisis.
Les frais saisis peuvent remonter jusqu'à un an en arrière.

-Use case Comptable :

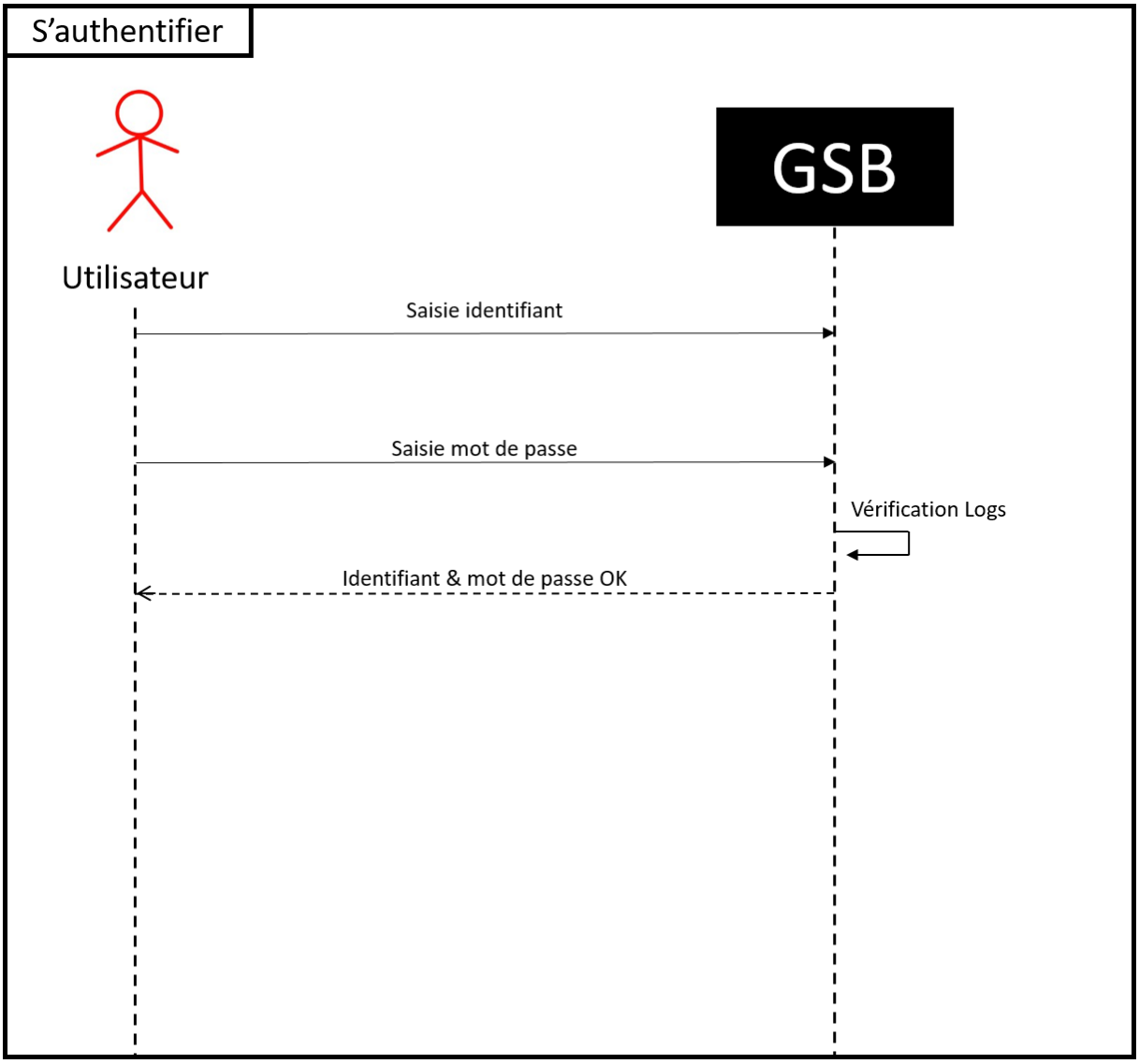


III/ Modélisation fonctionnelle

Méthodologie mise en œuvre pour l'implémentation du code :

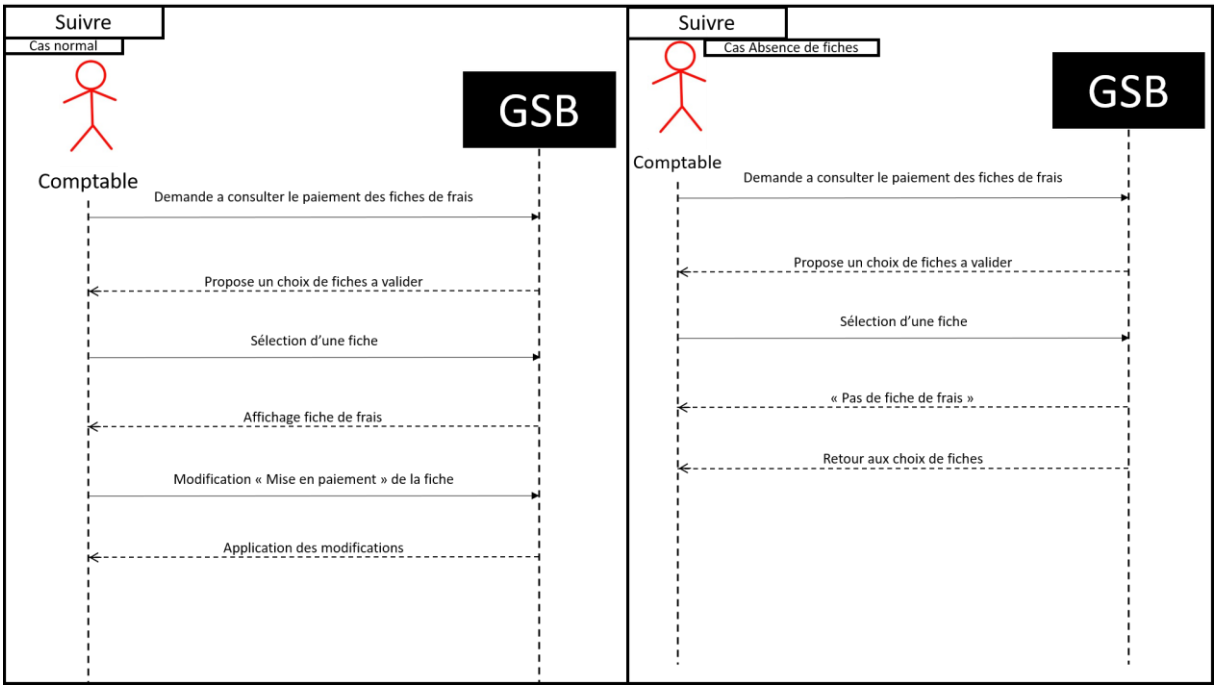
On réalise un USE CASE de la situation pour exprimer les besoins et la modélisation fonctionnelle.

Ici on démontre l'authentification d'un utilisateur sur le site de GSB.



L'utilisateur saisit son identifiant puis son mot de passe, l'application va vérifier que les identifiants saisis sont inscrits dans la base de données.

Cas d'utilisation du comptable :

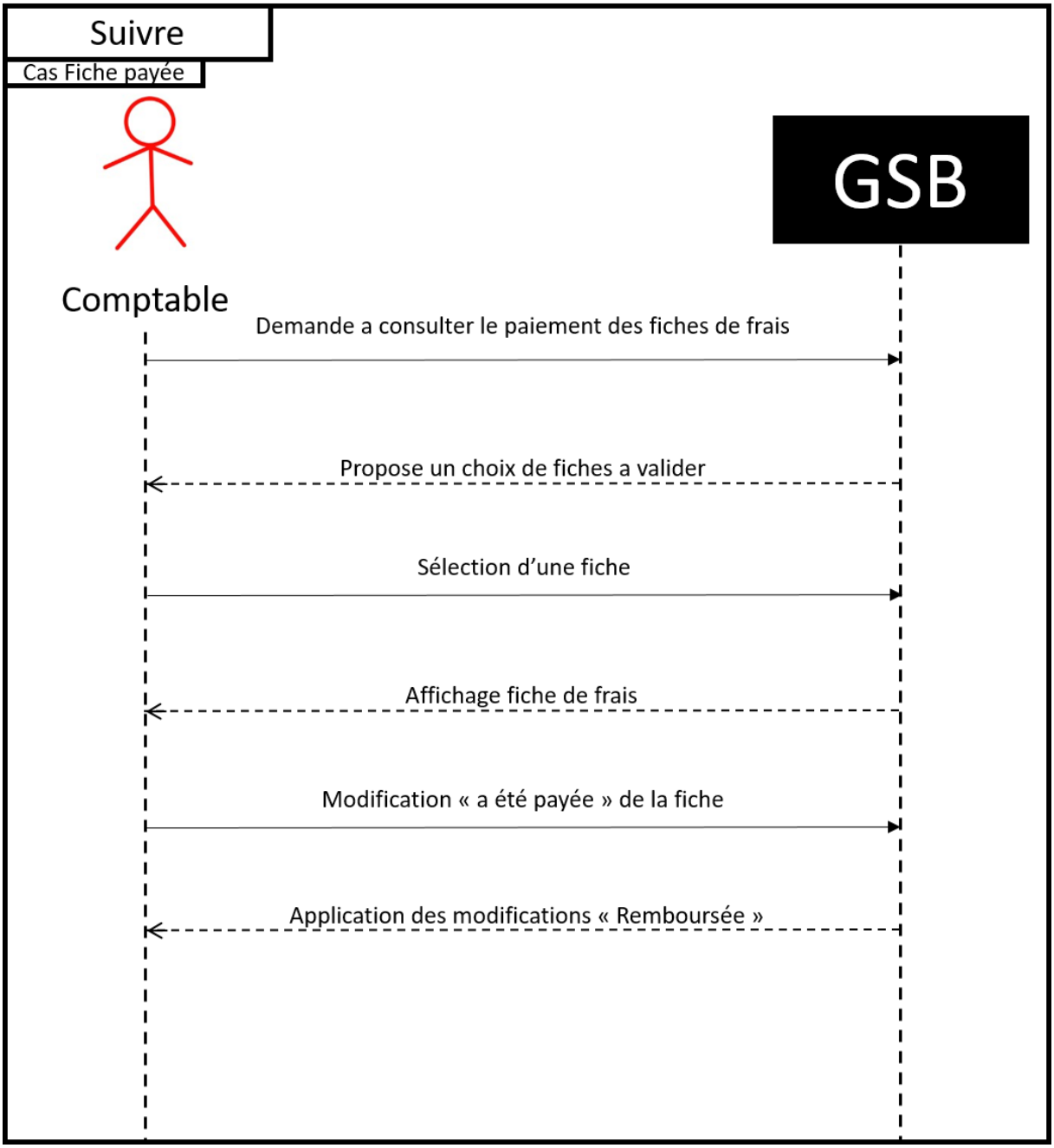


Suivre (Cas normal) :

Le comptable veut consulter les fiches. Il en choisit une parmi celles proposées et choisi de la modifier (changement à "mise en paiement" de la fiche).

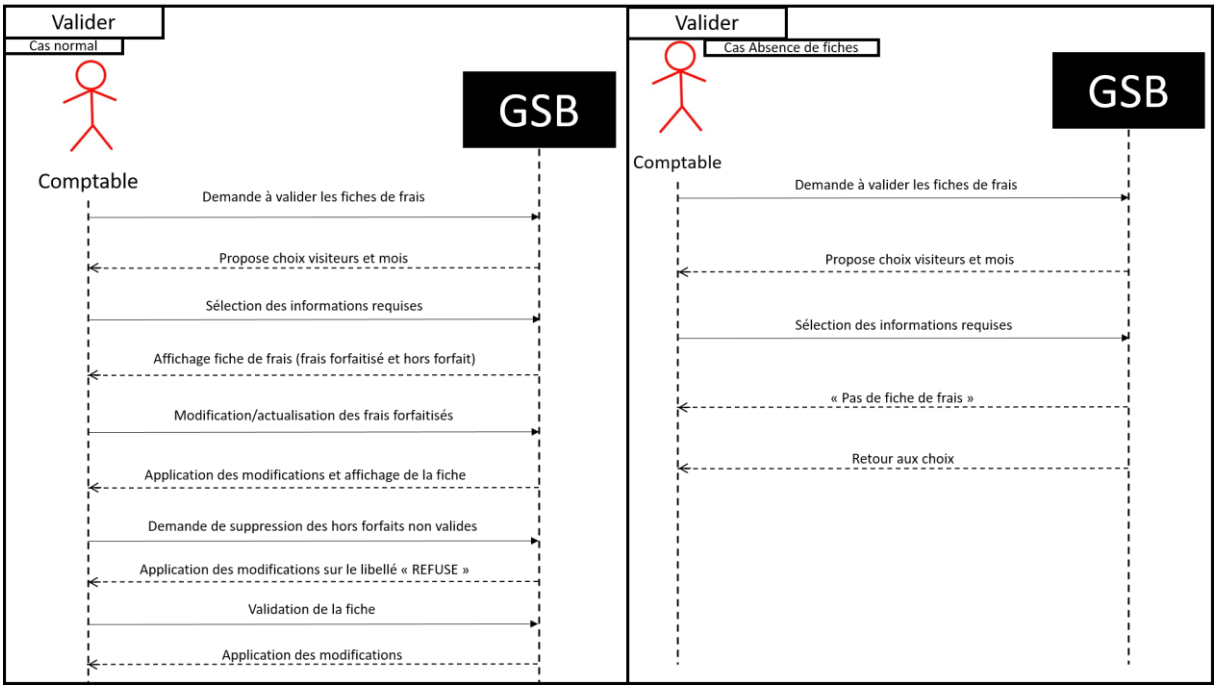
Suivre (Cas absence de fichiers) :

Le comptable veut consulter les fiches. Aucune fiche ne lui est proposé, il n'a donc d'autre choix que de revenir à la page d'accueil ou de se déconnecter.



Suivre (Cas normal) :

Le comptable veut consulter les fiches. Il en choisit une parmi celles proposées et choisi de la modifier (changement du statut à “a été payé”).

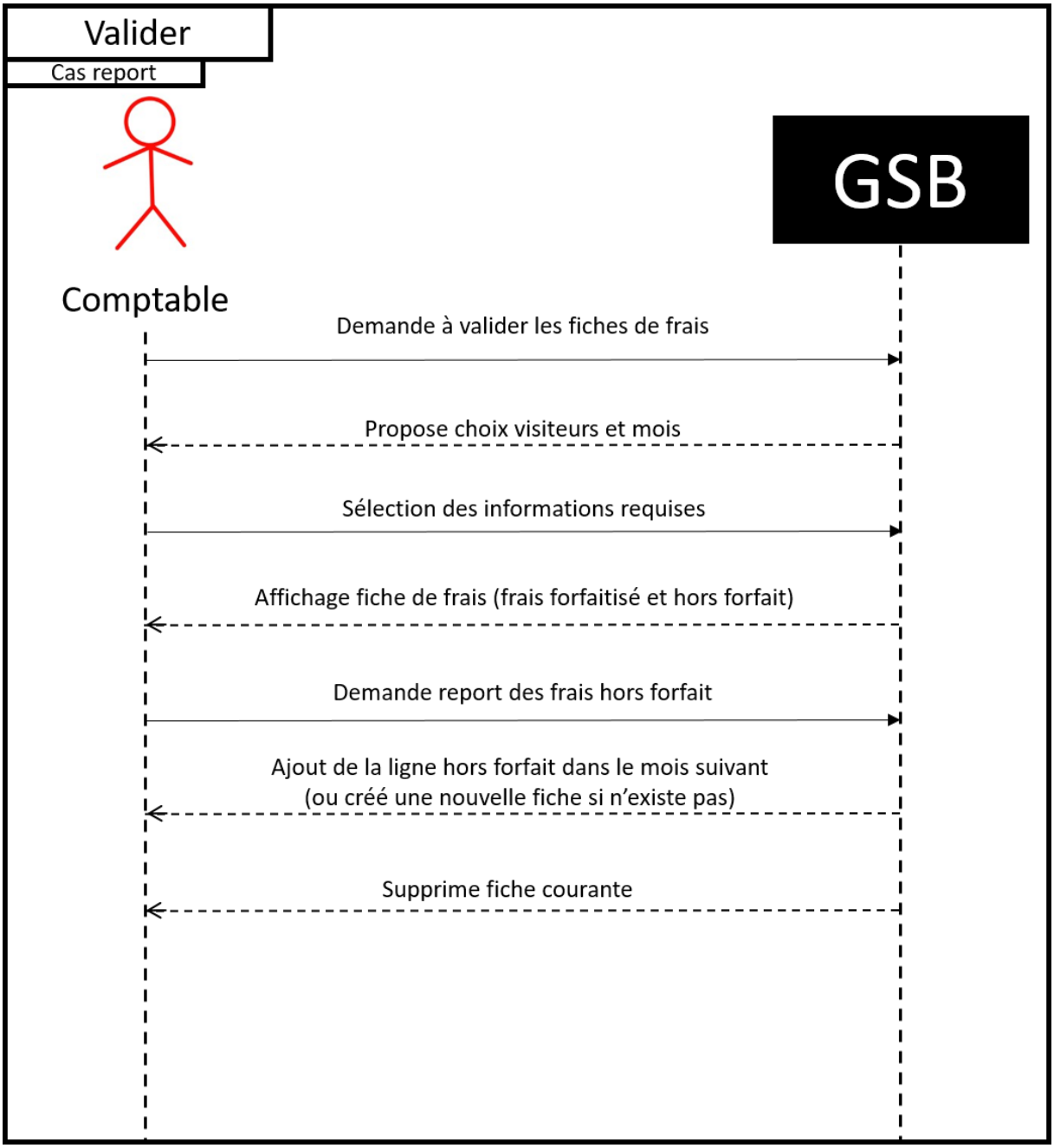


Valider (Cas normal) :

Le comptable veut consulter les fiches. Il en choisit une parmi celles proposées, sélectionne le visiteur et le mois. Modifie les frais forfaitisés et validation de la fiche.

Valider (Cas d'absence de fiche) :

Le comptable veut consulter les fiches. Il sélectionne le visiteur et le mois. Aucune fiche ne lui est proposé.

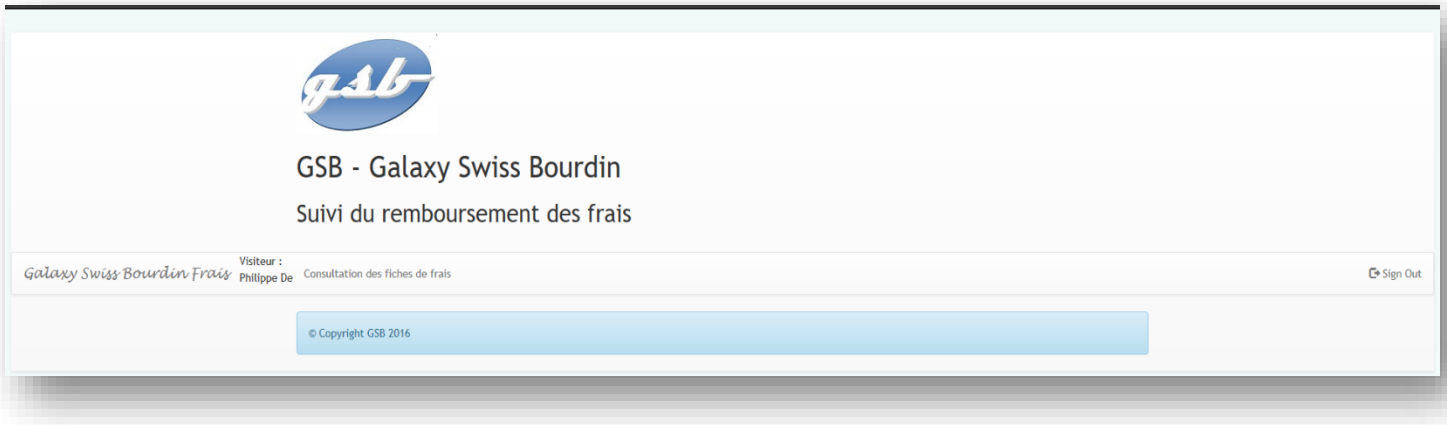


Valider (Cas d'un report de fiche) :

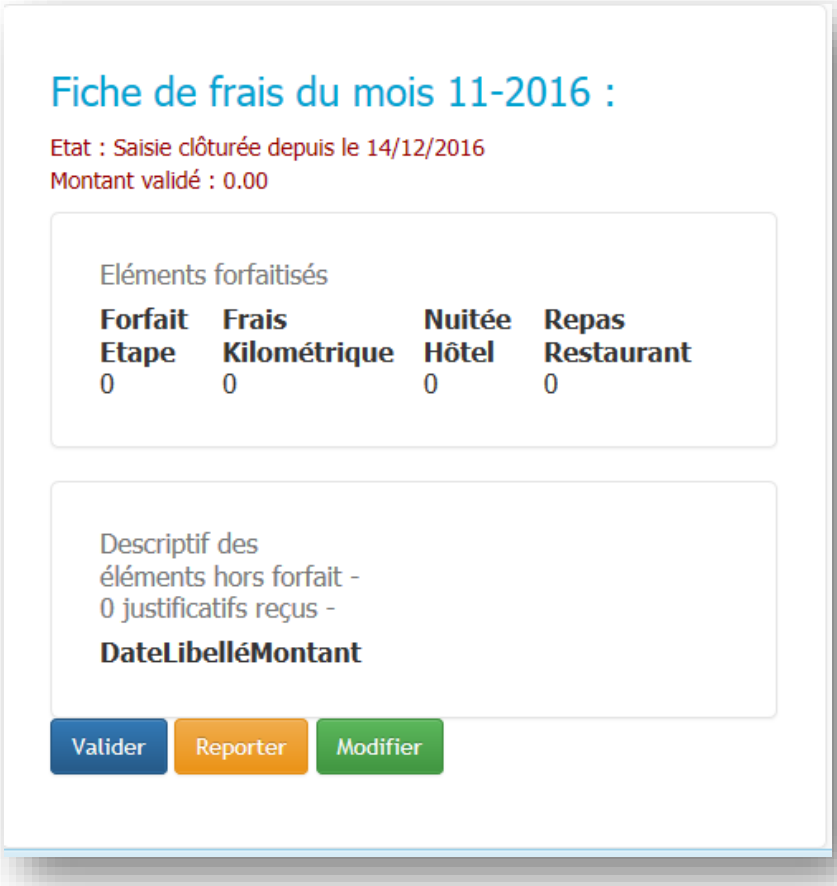
Le comptable veut consulter les fiches. Il en choisit une parmi celles proposées, sélectionne le visiteur et le mois. Envoie une demande de report de fiche. Une fiche est créée dans la page du mois suivant et bien sur la fiche dans le mois d'origine est supprimée.

IV/ IHM

L'Interaction entre Humain et Machine est l'ensemble des dispositifs matériels et logiciels permettant à un utilisateur de communiquer avec un système informatique.



Le comptable n'aura accès qu'à l'option "Consultation des fiches de frais"



```
<label for="lstMois" accesskey="n">Mois : </label>
<select id="lstMois" name="lstMois">
    <?php
    foreach ($lesMois as $unMois)
    {
        $mois = $unMois['mois'];
        $numAnnee =  $unMois['numAnnee'];
        $numMois =  $unMois['numMois'];
        if($mois == $moisASelectionner){
        ?>
        <option selected value="<?php echo $mois ?>"><?php echo  $numMois."/".$numAnnee ?> </option>
        <?php
        }
        else{ ?>
        <option value="<?php echo $mois ?>"><?php echo  $numMois."/".$numAnnee ?> </option>
        <?php
        }

    }

   ?>

</select>

  <label for="visiteur" accesskey="n">Visiteurs : </label>
<select id="visiteur" name="visiteur">
    <?php
    foreach ($nom as $prenom)
    {
        $visiteur = $nom['nom'];
        $visiteur =  $prenom['prenom'];

        if($visiteur == $nom){
        ?>
        <option selected value="<?php echo $query="SELECT * nom FROM utilisateurs WHERE idType = v"; ?>"> </option>
        <?php
        }
```

Affiche deux listes déroulantes proposant au comptable de choisir le mois et le visiteur qu'il souhaite modifier.

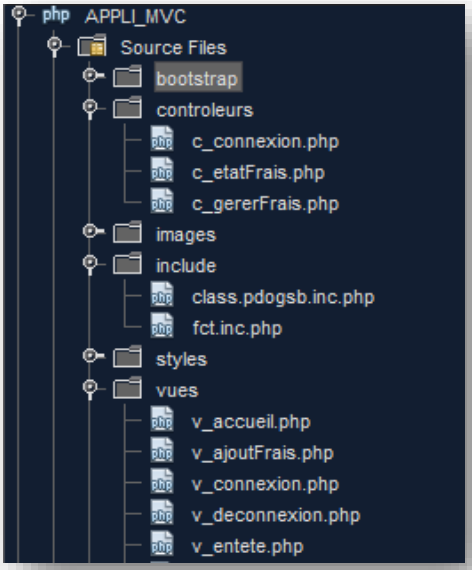
Après sélection du mois et du Visiteur le comptable se voit attribuer 3 choix : Valider, Reporter, Modifier la fiche.

V/ Architecture applicative

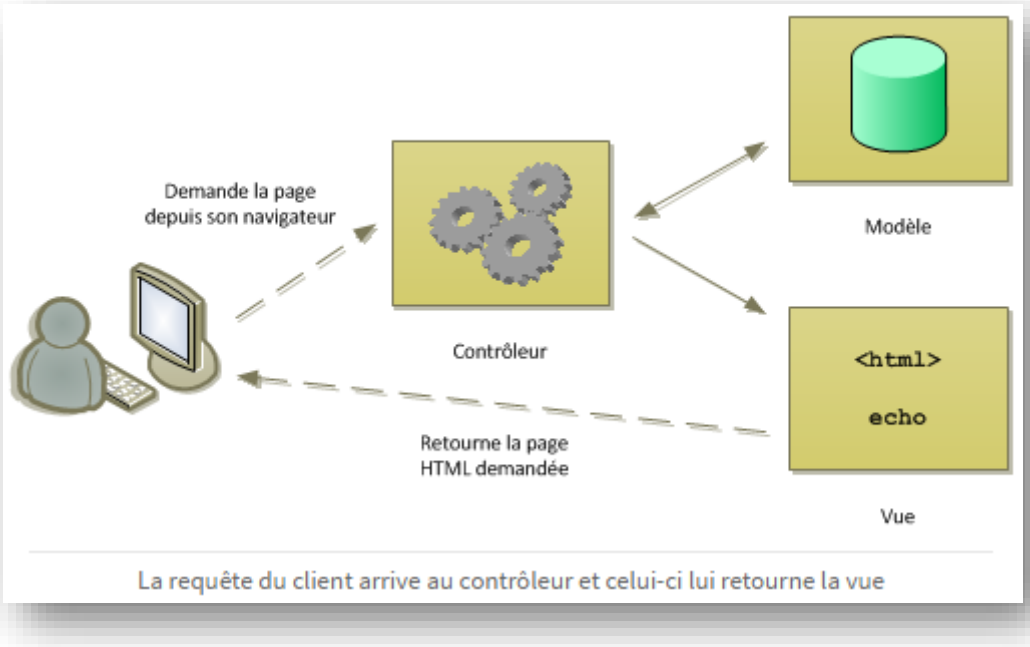
MVC est un design pattern, une bonne pratique de programmation qui recommande de découper son code en trois parties qui gèrent les éléments suivants :
Modèle : stockage des données
Vue : affichage de la page
Contrôleur : logique, calculs et décisions
Utiliser l'architecture MVC sur son site web en PHP est recommandé, car cela permet d'avoir un code plus facile à maintenir et à faire évoluer.

L'architecture MVC permet de séparer l'affichage des informations, les actions de l'utilisateur et l'accès aux données. MVC signifie Modèle-Vue-Contrôleur. C'est un modèle qui a été conçu au départ pour des applications dites « client lourd » (dont la majorité des données sont traitées sur le poste client. Par exemple : un traitement de texte comme Word). MVC était tellement puissant pour ces

applications, qu'il a été massivement adopté comme modèle pour la création d'applications web dites « client léger ».



Dans notre application, le dossier Include rempli le rôle du Modèle : accès à la base, fonctions métier, gestion des erreurs.



Modèle : cette partie gère les données du site. Son rôle est d'aller récupérer les informations brutes dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le Contrôleur. On y trouve donc les requêtes SQL.

Vue : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples.

Contrôleur : cette partie gère la logique du code qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au Modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la Vue. Le Contrôleur contient exclusivement du PHP. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).

Voici un extrait de code source du Contrôleur de notre application :

```
switch($action){
        case 'demandeConnexion':{
                include("vues/v_connexion.php");
                break;
        }
        case 'valideConnexion':{
                $login = $_REQUEST['login'];
                $mdp = $_REQUEST['mdp'];
                $utilisateurs = $pdo->getInfosUtilisateur($login,$mdp);
                if(!is_array( $utilisateurs)){
                        ajouterErreur("Login ou mot de passe incorrect");
                        include("vues/v_erreurs.php");
                        include("vues/v_connexion.php");
                }
                else{
                        $id = $utilisateurs['id'];
                        $nom =  $utilisateurs['nom'];
                        $prenom = $utilisateurs['prenom'];
                        connecter($id,$nom,$prenom);
                        include("vues/v_sommaire.php");
                }
                break;
        }
```

Ici, le contrôleur va demander à la Vue d'afficher v_connexion.php. Il va ensuite demander au Modèle si le login et le mot de passe correspond à la base de données. Il fait le lien entre l'utilisateur, le Modèle et la Vue.

VI/ Persistance des données

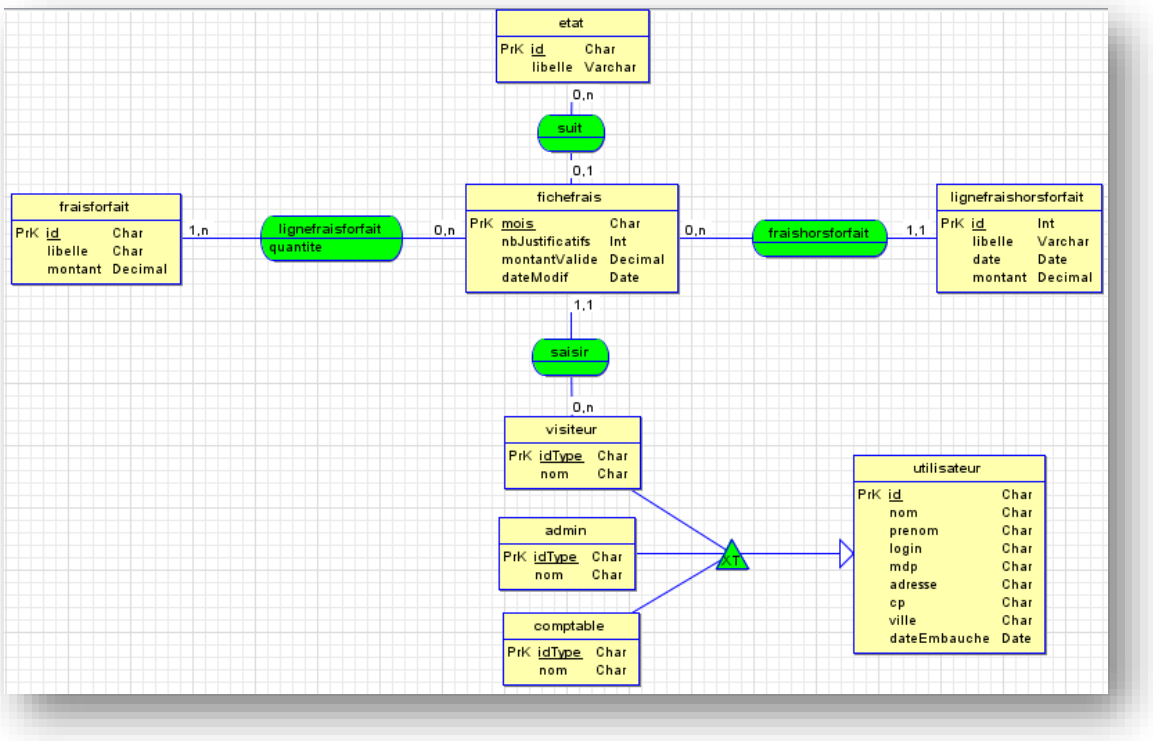
La couche Persistante est la couche d'accès aux données.

A/ Schéma de données (MCD)

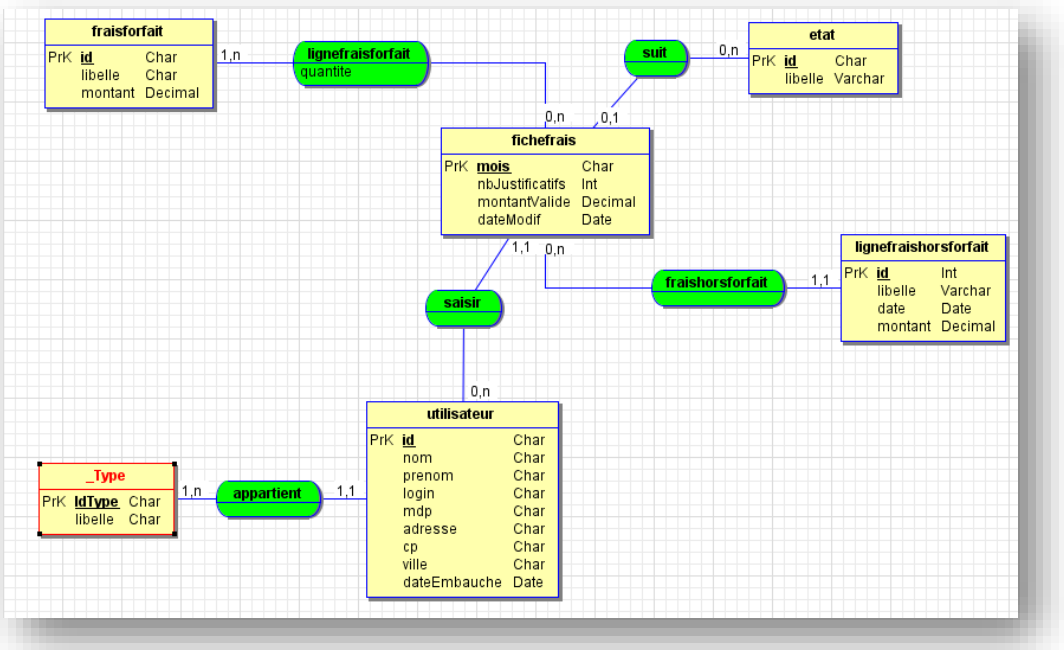
Le modèle conceptuel des données (MCD) a pour but d'écrire de façon formelle les données qui seront utilisées par le système d'information. Il s'agit donc d'une représentation des données, facilement compréhensible, permettant de décrire le système d'information à l'aide d'entités.

On peut plus ou moins optimisé notre MCD en fonction de comment on le crée (ici deux exemples avec et sans héritage)

MCD avec héritage (celui utilisé) :



MCD (version la plus optimisé) :



B/ Extrait du script SQL de création des tables

Le script SQL suivant consiste en la création de la table utilisateurs. Nous avons rajouter la colonne idType pour que chaque utilisateurs ait son type d'identification ici 'v' pour visiteur, 'c' pour comptable et 'a' pour admin.

```
-- Structure de la table `Utilisateur`
--

CREATE TABLE IF NOT EXISTS `utilisateurs` (
  `id` char(4) NOT NULL,
  `nom` char(30) DEFAULT NULL,
  `prenom` char(30)  DEFAULT NULL,
  `login` char(20) DEFAULT NULL,
  `mdp` char(20) DEFAULT NULL,
  `adresse` char(30) DEFAULT NULL,
  `cp` char(5) DEFAULT NULL,
  `ville` char(30) DEFAULT NULL,
  `dateEmbauche` date DEFAULT NULL,
  `idType` char(1) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

idType
a
c
v
v
v
v
v
v
v

Une contrainte d'intégrité est une clause permettant de contraindre la modification de tables, faite par l'intermédiaire de requêtes d'utilisateurs, afin que les données saisies dans la base soient conformes aux données attendues. Ces contraintes doivent être exprimées dès la création de la table.

Une **contrainte d'intégrité référentielle** est un (ou plusieurs) attribut(s) d'une relation qui apparaît comme clé dans une autre relation.

Le script suivant concerne la création de la table FicheFrais avec pour clé primaire "idVisiteur" et "mois" et pour contrainte d'intégrité la clé étrangère "idEtat" et "idVisiteur" des entités respectives "Etat" et "utilisateurs".

```
-- Structure de la table `FicheFrais`
--

CREATE TABLE IF NOT EXISTS `fichefrais` (
  `idVisiteur` char(4) NOT NULL,
  `mois` char(6) NOT NULL,
  `nbJustificatifs` int(11) DEFAULT NULL,
  `montantValide` decimal(10,2) DEFAULT NULL,
  `dateModif` date DEFAULT NULL,
  `idEtat` char(2) DEFAULT 'CR',
  PRIMARY KEY (`idVisiteur`,`mois`),
  FOREIGN KEY (`idEtat`) REFERENCES Etat(`id`),
  FOREIGN KEY (`idVisiteur`) REFERENCES utilisateurs(`id`)
) ENGINE=InnoDB;
```

VII/ Accès aux données

A/ PDO

Php Data Object, est une extension PHP qui permet d'utiliser une base de données en programmant avec un style orienté objet. Il s'agit une couche d'abstraction des fonctions d'accès aux bases de données. Ce qui signifie que les fonctions d'accès seront universelles. PDO ne fournit pas une abstraction de base de données.

PDO permet l'utilisation simultanée ou alternée de plusieurs Systèmes de Gestion de Bases de Données avec le même code PHP.

Cette classe n'est pas associée au modèle MVC, mais elle allège grandement la gestion des données. Dans l'application GSB, la classe PDO est encapsulée dans une classe PdoGsb.

Quelques exemples de méthodes :

Retourne id, nom et prénom du visiteur sous la forme d'un tableau associatif

```php
public function getInfosVisiteur($login, $mdp) {
    $req = "select visiteur.id as id, visiteur.nom as nom, visiteur.prenom as prenom from visiteur
            where visiteur.login='$login' and visiteur.mdp='$mdp'";
    $rs = PdoGsb::$monPdo->query($req);
    $ligne = $rs->fetch();
    return $ligne;
}
```

Retourne les lignes de frais hors forfait sous forme d'un tableau associatif

```php
public function getLesFraisHorsForfait($idVisiteur, $mois) {
    $req = "select * from lignefraishorsforfait where lignefraishorsforfait.idvisiteur ='$idVisiteur'
            and lignefraishorsforfait.mois = '$mois' ";
    $res = PdoGsb::$monPdo->query($req);
    $lesLignes = $res->fetchAll();
    $nbLignes = count($lesLignes);
    for ($i = 0; $i < $nbLignes; $i++) {
        $date = $lesLignes[$i]['date'];
        $lesLignes[$i]['date'] = dateAnglaisVersFrancais($date);
    }
    return $lesLignes;
}
```

Retourne tous les id des FraisForfait

```php
public function getLesIdFrais() {
    $req = "select fraisforfait.id as idfrais from fraisforfait order by fraisforfait.id";
    $res = PdoGsb::$monPdo->query($req);
    $lesLignes = $res->fetchAll();
    return $lesLignes;
}
```

B/ Design Pattern Singleton

Nous avons utilisé une version de PDO implémentant le design pattern Singleton. Ce dernier, permet d'obtenir une unicité dans l'instanciation de la classe, de telle sorte que nous n'instancions pas plusieurs occurrences d'une même classe alors que nous n'avons besoin, dans notre application, que d'une seule instance.

```php
class PdoGsb {

    private static $serveur = 'mysql:host=localhost';
    private static $bdd = 'dbname=gsb_frais';
    private static $user = 'root';
    private static $mdp = '';
    private static $monPdo;
    private static $monPdoGsb = null;

    /**
     * Constructeur privé, crée l'instance de PDO qui sera sollicitée
     * pour toutes les méthodes de la classe
     */
    private function __construct() {
        PdoGsb::$monPdo = new PDO(PdoGsb::$serveur . ';' . PdoGsb::$bdd, PdoGsb::$user, PdoGsb::$mdp);
        PdoGsb::$monPdo->query("SET CHARACTER SET utf8");
    }
```

Pour le mettre en place, les attributs sont tous statiques, les 4 premiers pour la connexion. $monPdo est de type PDO et $monPdoGsb contiendra l'unique instance de la classe.

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

C'est la fonction statique qui crée l'unique instance, et qui retourne l'unique objet de la classe.

C/ Gestion des sessions

La connexion d'un utilisateur se réfère à son id, nom et son idType (si c'est un visiteur ce sera "v", un administrateur sera "a" et un comptable sera "c"). Lors de la connexion la base de donnée vérifie que tout est bien correcte.

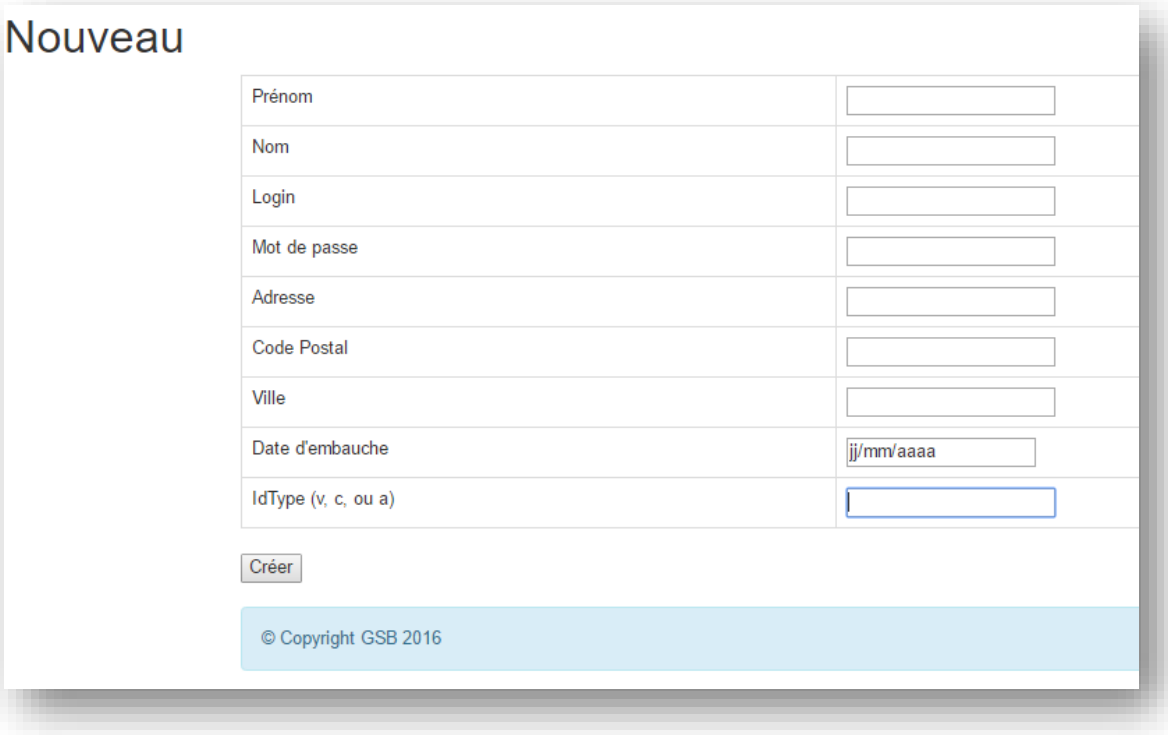
```
public function getInfosUtilisateurs($login, $mdp, $idType){
    $req = "select utilisateurs.id as id, utilisateurs.nom as nom, utilisateurs.prenom as prenom from utilisateurs
    where utilisateurs.login='$login' and utilisateurs.mdp='$mdp' and utilisateurs.idType='$idType'";
    switch($idType){
case 'v':{
    include("controleurs/c_connexion.php");break;
}
case 'a' :{
    include("controleurs/c_gererFrais.php");break;
}
case 'c' :{
    include("controleurs/c_etatFrais.php");break;
}

    $rs = PdoGsb::$monPdo->query($req);
    $ligne = $rs->fetch();
    return $ligne;
}
```

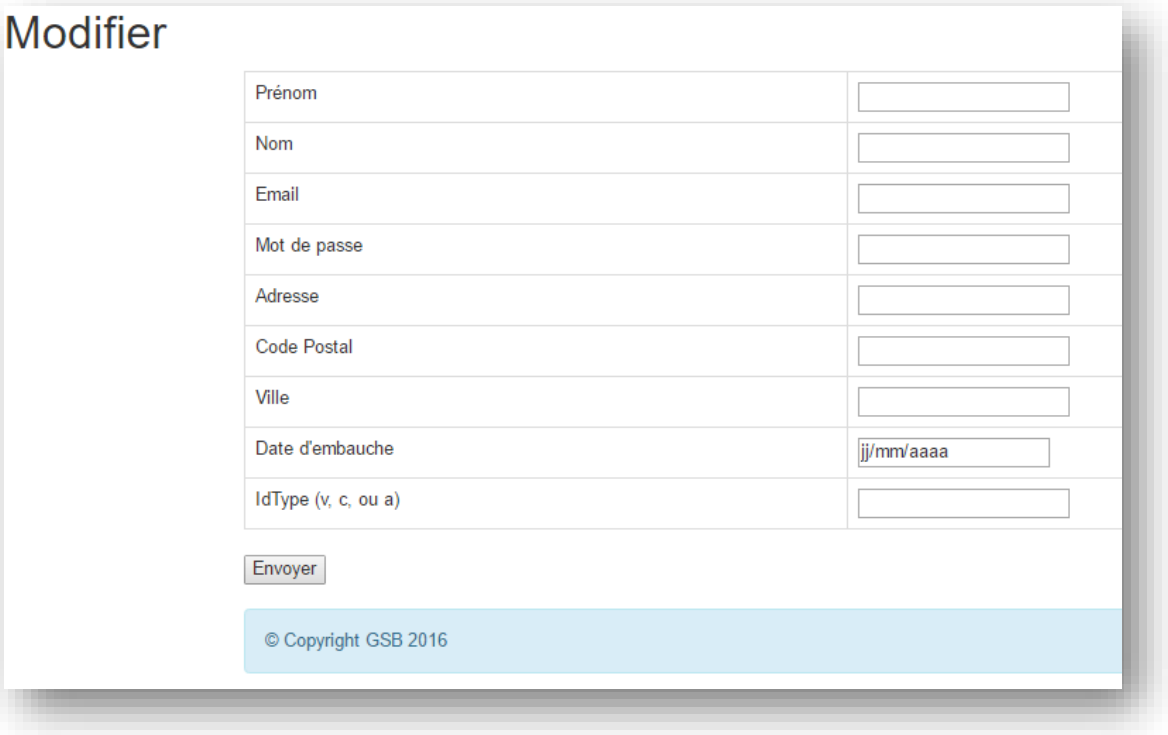
```
if(!is_array( $utilisteurs)){
    ajouterErreur("Login ou mot de passe incorrect");
    include("vues/v_erreurs.php");
    include("vues/v_connexion.php");
}
else if(is_array( $utilisateurs)){
    $id = $utilisateurs['id'];
    $nom =  $utilisateurs['nom'];
    $idType = $utilisateurs['idType'];
    connecter($id,$nom,$idType);
    include("vues/v_sommaire.php");
    if($idType == v){
        return $v;
    }
    else if($idType == a){
        return $a;
    }
    else if($idType == c){
        return $c;
    }
     }
break;
```

L'administrateur peut Créer, Modifier et Supprimer les sessions.

- Créer

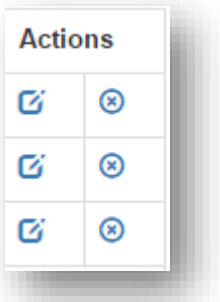


- Modifier :



- Supprimer :

Pour chaque Utilisateur, l'administrateur peut le modifier ou le supprimer à l'aide de ces petites icônes à la fin de chaque ligne générée :



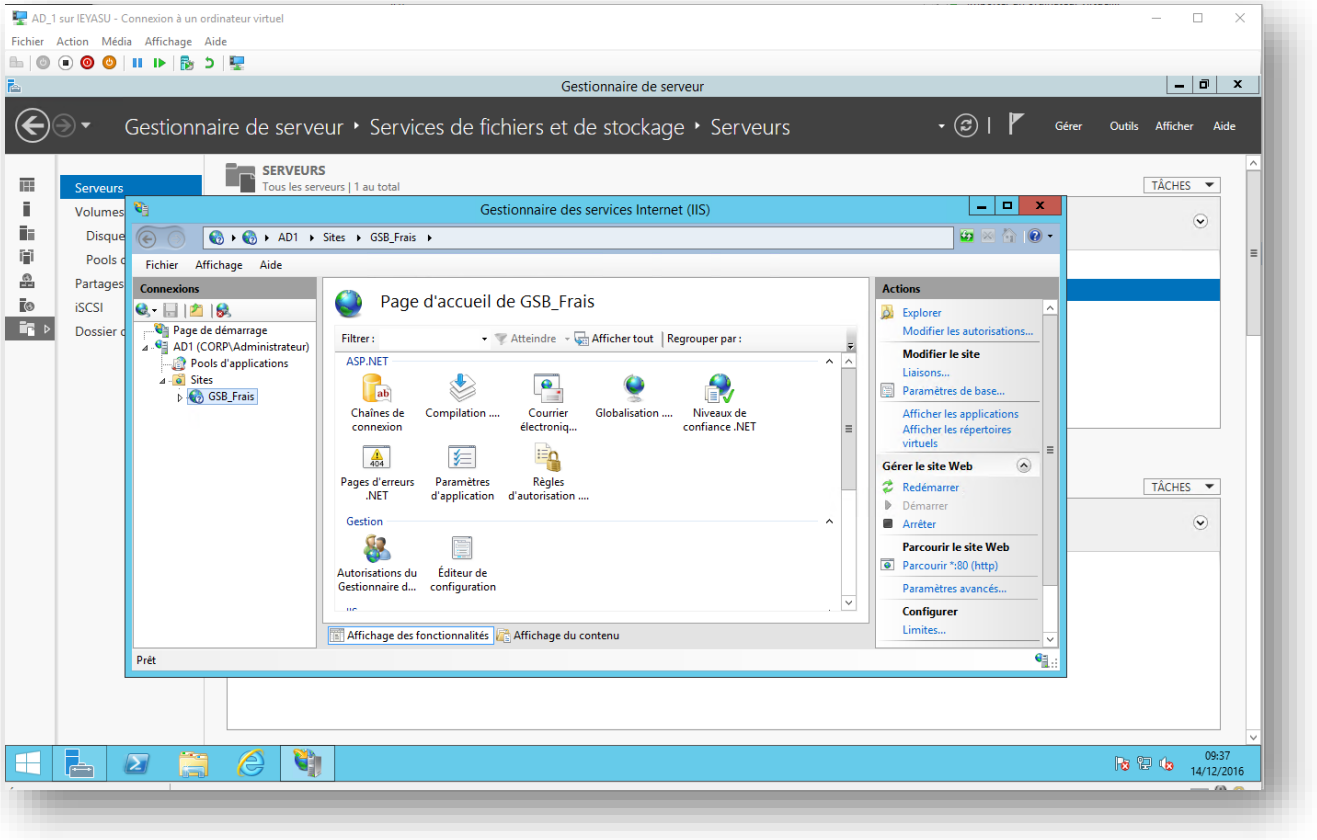

```
<div class="container">
    <a href="class.pdogsb.inc.php" class="btn btn-large btn-info">
        <i class="glyphicon glyphicon-plus"></i>   Nouveau Contact
    </a>
    <a href="class.pdogsb.inc.php" class="btn btn-large btn-info">
        <i class="glyphicon"></i>   Modifier contact
    </a>
    <a href="class.pdogsb.inc.php" class="btn btn-large btn-info">
        <i class="glyphicon"></i>   Supprimer contact
        <onclick <?php DELETE 'id,nom,prenom,login,mdp,adresse,cp,ville,dateEmbauche,idType' FROM 'utilisateurs' WHERE 1; ?>
    </a>
</div>
<br/>

<div class="container">
    <table class="table table-bordered table-responsive">
        <tr>
            <th>id</th>
            <th>nom</th>
            <th>prénom</th>
            <th>login</th>
            <th>mdp</th>
            <th>adresse</th>
            <th>code postal</th>
            <th>ville</th>
            <th>dateEmbauche</th>
            <th colspan = "2" align="center">Actions</th>
        </tr>
        <?php
        $query = "SELECT * FROM utilisateurs";
        $monPdo->dataview($newquery);
        ?>
        <td colspan="7" align="center"><?php $monPdo->pagingLink($query, $records_per_page) ?></td>
    </table>
</div>
```

A l'aide de l'interface l'administrateur choisit d'ajouter, modifier ou supprimer un utilisateur.

VIII/ Déploiement, mise en production de l'application

L'application est donc déployée sur une VM Windows server 2012 avec le service IIS installé dessus.



IX/ Tests fonctionnels

N°	Action	Résultat attendu	OK / KO
1	L'utilisateur lance l'application	Le système affiche la page d'authentification	OK
2	L'utilisateur saisit le login et le mot de passe puis clique sur bouton "Valider".	Le système affiche la page d'accueil	OK
3	Le visiteur clique sur saisie fiche de frais	Le système affiche la page pour renseigner les fiches de frais	OK
4	Le visiteur clique sur consultation des fiches de frais	Le système affiche la page pour consulter ses fiches de frais en fonction du mois	OK
5	Le visiteur choisit un mois pour consulter ses fiches de frais	Le système affiche la fiche de frais du mois sélectionné	OK
6	Le comptable clique sur "Consultation des fiches de frais"	Le système affiche la page pour consulter les fiches de frais	OK
7	Le comptable renseigne le mois et L'utilisateur afin de consulter ses fiches de frais	Le système affiche la fiche en question	KO
8	Le comptable souhaite modifier la fiche	Le système lui donne les droit d'édition de la fiche	KO
9	Le comptable souhaite reporter la fiche à un autre mois	Le système crée une nouvelle fiche dans le mois sélectionné et la supprime dans le mois actuel	KO
10	Le comptable souhaite Valider une fiche de frais	Le système change le statut de la fiche	KO
11	L'administrateur clique sur créer	Le système affiche la page Créer Session	OK
12	L'administrateur clique sur modifier	Le système affiche la page Modifier Session	OK
13	L'administrateur clique sur supprimer	Le système supprime l'utilisateur de la base de données.	OK
14	L'utilisateur (Administrateur, Comptable ou Visiteur) clique sur "Sign out"	Le système renvoie l'utilisateur à la page de Login	OK

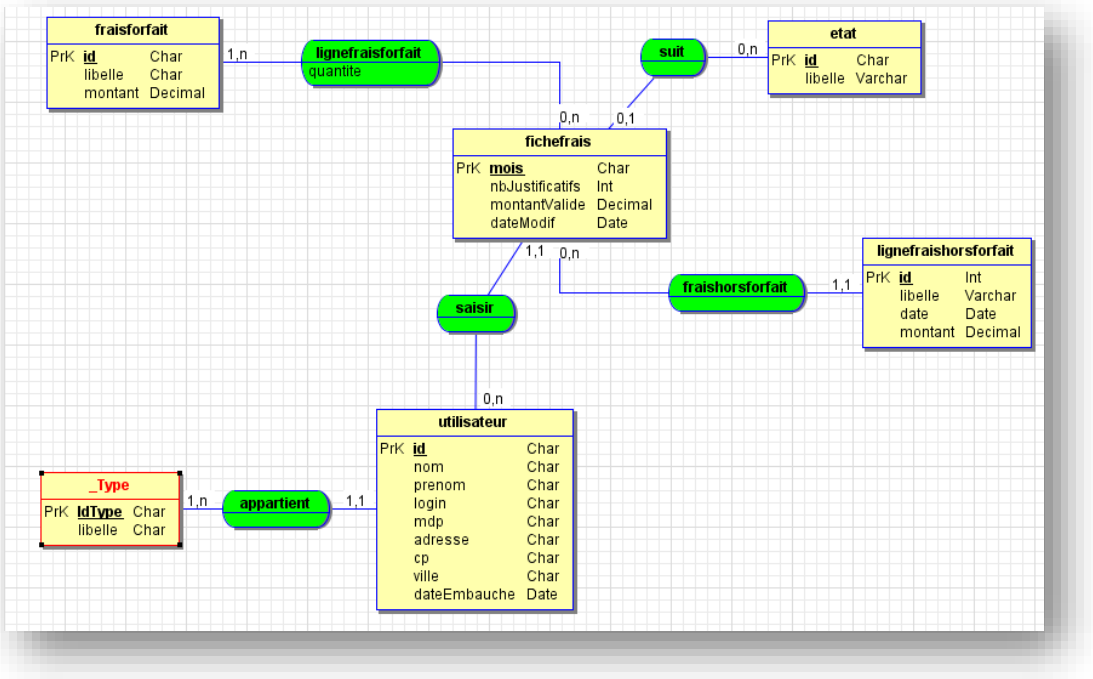
X/ Difficultés rencontrées

- Comprendre les objectifs du projet et ce que nous devions modifier.
- Réaliser l'UML.
- Manque de temps

XI/ Améliorations possibles

Nous aurions pu :

- Utiliser une version plus optimisée du MCD :



- Crypter le mot de passe dans la base de données

- Au niveau du Cas d'Utilisation "Consulter fiche frais", rendre la fiche de frais facilement imprimable, par exemple proposer la génération d'un fichier PDF.

- Au niveau des Cas d'Utilisation "Saisir fiche frais" et "Consulter fiche frais", prévoir l'affichage des éléments intermédiaires de calcul permettant d'apprécier le montant correspondant à chaque ligne de frais forfaitisé, ainsi que les totaux des éléments forfaitisés et hors forfait, de sorte à être plus proche du document relatif à la fiche de demande de remboursement.

- Modifier la base de données (et l'application) pour que l'on connaisse le statut d'une ligne de frais hors forfait (point 8 de l'UC « Valider Frais » qui ajoute le texte « REFUSE » en début de libellé.

-Enregistrer et montrer la dernière date de modification d'une fiche.

XII/ Conclusion

De nombreux frameworks PHP, tels que Symfony et le Zend Framework, permettent de mettre rapidement en place les bases d'une architecture MVC sur un site. Ce sont des outils appréciés des professionnels qui nécessitent cependant un certain temps d'adaptation. C'est pourquoi nous n'avons pas utilisez de frameworks.

L'implémentation de bootstrap dans l'application est une marque de professionnalisation et de qualité qui a pour but d'établir ou de garder le standing de l'entreprise.