Dubail Noah 15/09/2023

Gestion des frais GSB



**Lycée Condorcet BTS SIO – SLAM Bloc 2**

# **Tables des matières**

[**Tables des matières** 2](#_Toc156061686)

[**Historique du document** 4](#_Toc156061687)

[**Description du sujet de projet** 5](#_Toc156061688)

[**Le contexte du projet** 5](#_Toc156061689)

[**Le domaine de développement** 5](#_Toc156061690)

[**Langage de développement utilisée** 5](#_Toc156061691)

[**Le fonctionnement de l’application web** 5](#_Toc156061692)

[**Le périmètre du système** 7](#_Toc156061693)

[**Le logo de l’application** 7](#_Toc156061694)

[**Les extensions possibles** 7](#_Toc156061695)

[**Le Modèle de dictionnaire** 8](#_Toc156061696)

[**Le Modèle entité association** 10](#_Toc156061697)

[**La modélisation logique** 13](#_Toc156061698)

[**Script de création de la base de données** 15](#_Toc156061699)

[**Modélisation UML** 17](#_Toc156061700)

[**La modélisation de l’IHM et présentation de l’interface** 22](#_Toc156061701)

[**Structure de l’application web** 22](#_Toc156061702)

[**Page de connexion** 23](#_Toc156061703)

[**Page accueil d’un utilisateur** 23](#_Toc156061704)

[**Page d’ajout d’une mission** 24](#_Toc156061705)

[**Page de la mission** 25](#_Toc156061706)

[**Page d’ajout d’un frais** 26](#_Toc156061707)

[**Page de liste des missions** 27](#_Toc156061708)

[**Page d’accueil d’un comptable** 28](#_Toc156061709)

[**Page liste des utilisateurs pour le comptable** 28](#_Toc156061710)

[**Validation ou refus de mission** 29](#_Toc156061711)

[**Page d’accueil pour comptable** 29](#_Toc156061712)

[**Page d’ajout d’un utilisateur** 30](#_Toc156061713)

[**Présentation des principales requêtes développées.** 31](#_Toc156061714)

[**Création d’un nouveau frais** 32](#_Toc156061715)

[**Création d’une nouvelle mission** 33](#_Toc156061716)

[**Présentation du code développé dans l’application** 34](#_Toc156061717)

[**Les modèles dans laravel (Eloquent)** 35](#_Toc156061718)

[**Les vues** 36](#_Toc156061719)

[**Les contrôleurs** 39](#_Toc156061720)

[**Schéma du modèle MVC :** 39](#_Toc156061721)

# **Historique du document**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date de mise à jour | Auteur | Modification essentielles |
| 1.0 | 22/09/2023 | Noah Dubail | Création du document |
| 1.1 | 29/01/2023 | Noah Dubail | Mise à jour version E2 |
| 1.1.1 | 04/01/2024 | Noah Dubail | Mise à jour des informations du cahier des charges | Mise à jour des diagrammes UML. |
| 1.2 | 11/01/2024 | Noah Dubail | Mise à jour version E3 |
| 1.3 | 12/01/2024 | Noah Dubail | Version finale |

# **Description du sujet de projet**

## **Le contexte du projet**

GSB (Galaxy Swiss Bourdin) est née de la fusion des deux sociétés pharmaceutiques : le géant américain Galaxy et le conglomérat européen Swiss Bourdin. Cette fusion a pour but l’optimisation de l’activité du groupe, en réalisant des économies d’échelle dans la production et la distribution des médicaments, tout en prenant le meilleur des deux laboratoires sur les produits concurrents.

Mon objectif est de concevoir un prototype de logiciel de gestion des frais de déplacement pour aider l'entreprise à économiser et simplifier cette tâche, particulièrement complexe au sein de grandes entreprises. En effet les déplacements et actions de terrain menées par les utilisateurs engendrent des frais qui doivent être pris en charge par la comptabilité. GSB dépense chaque année plus de 25 millions d’euro dans les frais de déplacement.

## **Le domaine de développement**

Ce projet sera développé avec le framework laravel 10.31.0 sur l’IDE Visual Studio Code pour l’application web, et sera couplé à une base de données MySQL avec laquelle il pourra interagir. Une historisation des fichiers sera effectuée grâce à git.

Lien GitHub : <https://github.com/99Noah99/Projet_GSB>

## **Langage de développement utilisée**

* PHP 8.2.0
* HTML
* CSS
* JavaScript
* SQL

## **Le fonctionnement de l’application web**

Le fonctionnement de l’application sera simple : une fois leur déplacement effectué, les utilisateurs vont pouvoir créer la « mission » correspondant à leur déplacement/démarchage et ajouter les différentes dépenses qu’ils ont effectuées durant cette mission afin d’être remboursés. De leur côté, les comptables pourront consulter les missions des utilisateurs et leurs frais correspondants.

Ils pourront donc choisir de valider la demande de remboursement de la mission et ses frais ou non.

**Les différents frais :**

Il existe différents types de frais forfaitisé :

* Le repas du midi sera géré par un forfait à 20€
* Le relais étape (nuit + repas) sera géré par un forfait à 120€
* Une nuit dans un hôtel sera gérée par un forfait à 90€
* Les kilomètres seront remboursés à hauteur de 0.20€ par kilomètres.
* Les billets d’avions seront entièrement remboursés
* Les billets SNCF seront entièrement remboursés

**Etat des demandes :**

Une fois que l’utilisateur à créer sa mission, il pourra consulter l’évolution de sa demande, dont voici les différents statuts :

* En attente de déclaration (l’utilisateur a créé la mission mais n’a pas fini d’ajouter ses frais, le comptable ne voit pas la mission sur son tableau de bord)
* En attende de validation (l’utilisateur à déclarer sa mission et attend la validation ou le refus d’un comptable)
* Validée (le comptable a validé la demande, l’utilisateur à le statut Validée sur sa liste des missions)
* Refusée (le comptable a refusé la demande par manque d’information, ou de justificatif… l’utilisateur à le statut Refusée sur sa liste des missions)

**Ordre des actions :**

Premièrement l’utilisateur se connecte via son identifiant et son mot de passe, il peut ensuite crée sa mission. Il arrive ensuite sur la mission qu’il a créé et peut ajouter des frais liés à cette mission. Pour chaque frais une pièce justificative sera demandée. Une fois tous les frais ajoutés il peut déclarer la mission.

Le comptable peut à son tour se connecter via un identifiant et un mot de passe et consulter le(s) la liste de tous les utilisateurs, il clique sur un utilisateur et peut visionner toutes les missions que celui si à déclarer. Il peut maintenant valider ou refuser une mission.

S’il refuse, l’utilisateur pourra voir dans la liste de ses missions que le statut de sa mission a changée en « Refusée », s’il accepte le statut changera en « Validée ».

**Les différents comptes :**

L’application aura donc 2 interfaces, une interface pour les utilisateurs qui peuvent ajouter leurs missions, frais et pièces justificatives, et une interface pour les comptables qui vont pouvoir consulter la liste de tous les utilisateurs et leurs missions.

Les utilisateurs pourront donc :

* Ajouter des missions
* Ajouter les frais qu’ils ont dépenser durant cette mission et leur justificatif
* Déclarer leur mission une fois tous les frais ajoutés

Les comptables pourront :

* Visionner la liste de tous les utilisateurs
* Visionner les missions d’un utilisateur
* Visionner les frais d’une mission et télécharger les pièces justificatives.
* Valider ou Refusée la demande de remboursement d’une mission

## **Le périmètre du système**

Une application qui prend un charge une interface pour les utilisateurs, et une interface pour les comptables, une interface admin permettant d’ajouter des nouveaux utilisateurs ou comptable dans l’application.

Une base de données MySQL qui communiquera avec l’application web afin de stocker les demandes de remboursement (missions et frais associées), les comptes utilisateurs et comptables…

## **Le logo de l’application**



## **Les extensions possibles**

Ajouter une date de visite ce qui permettrait de laisser un délai de 10 jours après la date de visite pour faire la demande de remboursement. A la fin du mois, toute les demandes de remboursement non clôturés seront automatiquement clôturées par l’application.

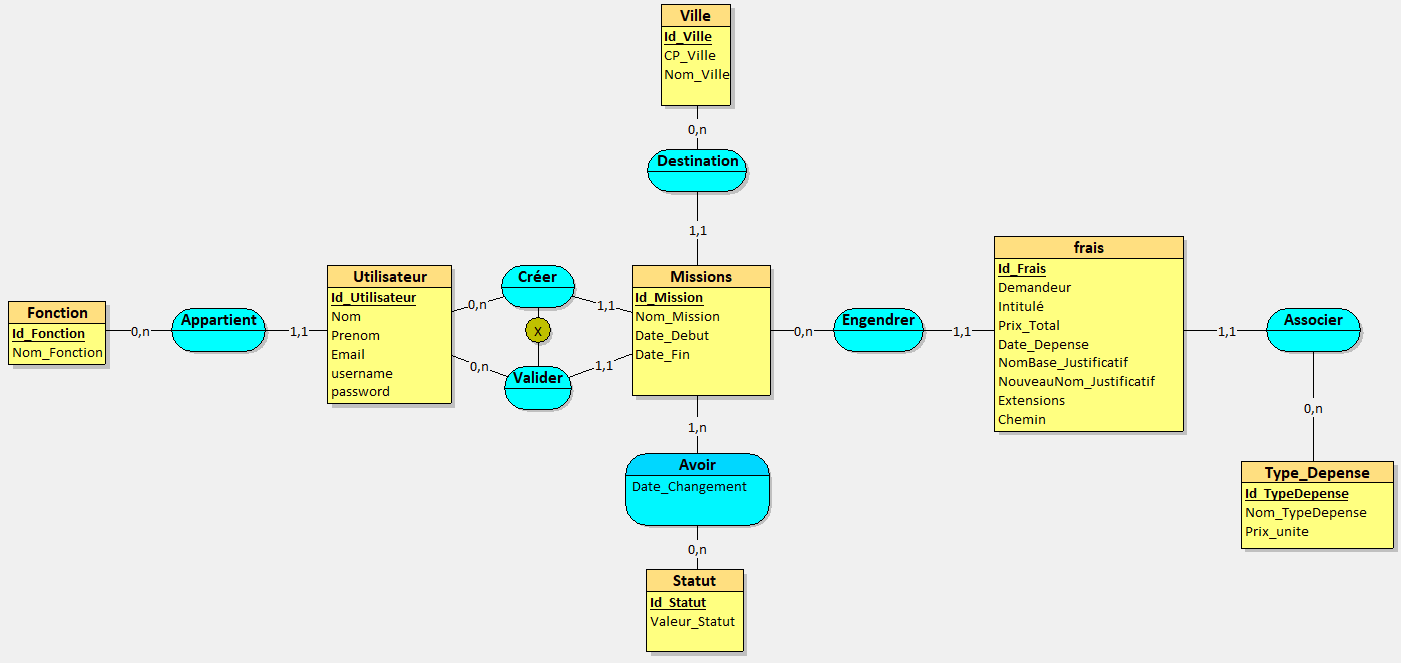
**La modélisation conceptuelle**

## **Le Modèle de dictionnaire**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Nom Conceptuel** | **Nom Logique** | **Type (E,Ca,Co)** | **Nature** | **Longueur** | **Identifiant** | **Exemple de valeur** |
| Identifiant de l’Utilisateur | Id\_Utilisateur | E | N | 4 | Oui | 251 |
| Nom de l’Utilisateur | Nom | E | A | 30 | Non | Walter |
| Prénom de l’Utilisateur | Prenom | E | A | 30 | Non | Thibaut |
| Email de l’Utilisateur | Email | E | AN | 100 | Non | [Walter.Thibaut@gmail.com](mailto:Walter.Thibaut@gmail.com) |
| Identifiant de connexion de l’Utilisateur | username | E | AN | 20 | Non | WThibaut8 |
| Mot de passe de l’Utilisateur | password | E | AN | 255 | Non | TW9$\*\*zrez45s)àsfd |
|  |  |  |  |  |  |  |
| Identifiant de la Ville | Id\_Ville | E | N | 20 | Oui | 42 |
| Code Postal de la ville | CP\_Ville | E | N | 5 | Non | 9000 |
| Nom de la Ville | Nom\_Ville | E | A | 40 | Non | Toulouse |
|  |  |  |  |  |  |  |
| Identifiant de la Mission | Id\_Mission | E | N | 5 | Oui | 23 |
| Nom de la Mission | Nom\_Mission | E | A | 250 | Non | Déplacement à Belfort |
| Date du début de la Mission | Date\_Debut | E | AN | 10 | Non | 02/11/2023 |
| Date de fin de la Mission | Date\_fin | E | AN | 10 | Non | 07/11/2023 |
|  |  |  |  |  |  |  |
| Date du changement du statut | DateChangement | E | AN | 10 | Non | 05/11/2023 |
|  |  |  |  |  |  |  |
| Identifiant du statut | Id Statut | E | N | 5 | Oui | 367 |
| Valeur du statut | Valeur\_Statut | E | A | 100 | Non | Refusé |
|  |  |  |  |  |  |  |
| Identifiant du frais | Id\_Frais | E | N | 5 | Oui | 545 |
| Demandeur du Remboursement | Demandeur | E | A | 30 | Non | Patrick |
| Intitulé de la demande de Remboursement | Intitulé | E | A | 200 | Non | Train concernant mon déplacement à Belfort |
| Prix de la demande de remboursement | Prix\_Total | E | N | 5 | Non | 250.00 |
| Date de la dépense | Date\_Depense | E | AN | 10 | Non | 20/10/2023 |
| Nom par défaut du justificatif | NomBase\_Justificatif | E | A | 150 | Non | JustificatifTransport.pdf |
| Nouveau nom du justificatif | NouveauNom\_Justificatif | E | AN | 150 | Non | PdfTransport250.pdf |
| Type d’extension | Extension | E | A | 10 | Non | .pdf |
| Chemins d’accès au Justificatif | Chemin | E | AN | 300 | Non | C:\Users\Utilisateur\Documents\BTS SIO\2ème année\SLAM |
|  |  |  |  |  |  |  |
| Identifiant du type de dépense | Id\_TypeDepense | E | N | 4 | Oui | 24 |
| Nom du type de dépense | Nom\_TypeDepense | E | A | 50 | Non | Repas midi |
| Prix de la dépense | Prix\_unite | E | N | 6 | Non | 20.00 |
|  |  |  |  |  |  |  |
| Identifiant de la fonction de l’utilisateur | Id\_Fonction | E | N | 4 | Oui | 3 |
| Nom de la fonction de l’utilisateur | Nom\_Fonction | E | A | 50 | Non | Comptable |

## **Le Modèle entité association**

Voici la modélisation du modèle entité association :



**Explications sur le choix de la modélisation :**

Entité Utilisateur :

Un utilisateur est identifié par un identifiant unique (ici id\_Utilisateur) et possède les attributs suivants : Nom, Prenom, Email, username (identifiant de connexion), password.

Un utilisateur a une fonction : utilisateur, comptable, admin.

Un utilisateur ayant la fonction utilisateur peut créer 0 ou plusieurs missions.

Un utilisateur (fonction comptable) peut valider 0 ou plusieurs missions. Un utilisateur ne peut pas créer ET valider une mission il peut faire sois l’un sois l’autre, en effet les utilisateurs ayant la fonction utilisateur peuvent créer des missions, et les utilisateurs ayant la fonction comptable valide ou refuse les missions.

Une mission est créée ou validée que par un seul utilisateur.

Nous avons relié la table Utilisateur à la table Mission par deux associations, cependant une contrainte d'exclusivité a été ajouté sur ses associations afin qu’il soit impossible pour un utilisateur à la fois de crée et de valider une mission.

Entité Missions :

Une mission est identifiée par un identifiant unique (ici id\_Missions) et possède les attributs suivants : Nom\_Mission, Date\_Debut, Date\_Fin.

Le nom de la mission permet de savoir à quoi correspond la mission, et les dates de début et de fin permettent de déterminer la durée de la mission.

Une mission a lieu dans une seule ville.

De plus, une mission peut avoir un ou plusieurs statuts, car celui-ci va évoluer au fil du temps (validé, refusé…).

Pour finir, une mission entraîne 0 ou plusieurs dépenses (frais).

Lors de la mission, l’utilisateur va dépenser de l'argent pour des repas, des trajets, des nuits d’hôtel, etc. Toutes ces dépenses sont enregistrées dans la table "frais" associée à la mission. Les frais sont calculés et classés par type de dépense. En effet, la table "Type\_Depense" permet de catégoriser les frais, par exemple, trajet, repas… Dans la table "Type\_Depense", on retrouve également le prix unitaire de chaque catégorie. La catégorie du frais est définie par la propriété "Nom\_TypeDepense".

Entité Ville :

Une ville est identifiée par un identifiant unique (ici id\_Ville) et possède les attributs suivants : CP\_Ville, Nom\_Ville, Rue.

Le CP\_Ville correspond au code postal, le Nom\_Ville au nom de la ville.

Dans une ville peut avoir lieu 0 ou plusieurs missions.

La table Ville permet d’avoir plus d’information concernant le lieu de la mission.

Entité Statut :

Un statut est identifié par un identifiant unique (ici id\_Statut) et possède l’attribut suivant : Valeur\_Statut.

La Valeur\_Statut va contenir le statut d’une mission, si elle est validée par le comptable, refusée…

Le statut est affecté à 0 ou plusieurs missions.

L’association reliant mission et statut possède l’attribut DateChangement qui nous permet de connaitre la date à laquelle le statut a changé (permet l’historisation des statuts).

Nous avons ajouté la table Statut afin de suivre les différentes étapes de la mission, de plus dans l’association reliant mission et statut se trouve la propriété DateChangement permettant de savoir à quel moment le statut a été modifié.

Entité Frais :

Un frais correspond à une dépense effectuée pendant une mission. Il est identifié par un identifiant unique (ici id\_Remboursement) et possède les attributs suivants : Demandeur, Intitulé, Prix\_Total, Date\_Depense, NomBase\_Justificatif, NouveauNom\_Justificatif, Extensions, Chemin. Le Demandeur est la personne faisant la demande de remboursement pour la dépense effectuée pendant la mission. L’Intitulé permet d’expliquer la nature de cette dépense. Le Prix\_Total correspond au montant que le demandeur demande à se faire rembourser. Par exemple, s'il a pris trois repas de midi, le Prix\_Total correspond à la somme de ces trois repas. Le NomBase\_Justificatif est le nom du fichier joint fourni par le demandeur en tant que justificatif.

Le NouveauNom\_Justificatif correspond au nom de la pièce jointe une fois renommer aléatoirement dans l’application et stocker dans celle-ci.

L’extension correspond à l’extension de la pièce jointe et le chemin correspond au chemin permettant d’accéder au justificatif.

Un frais appartient à une seule mission, et est associé à un seul type de dépense.

En effet un frais peu sois être un repas par exemple, ou une demande de remboursement de billet de train. Si on doit faire rembourser un repas et un billet de train il faudra créer deux frais.

Entité Type\_Depense :

Un type de dépense est identifié par un identifiant unique (ici id\_TypeDepense) et possède les attributs suivants :

Nom\_TypeDepense et Prix\_Unite.

Le nom du type de la dépense correspondra au type de dépense, par exemple : repas midi, nuit dans un hôtel, billet de train…

Le Prix\_Unite correspond au prix à l’unité d’une dépense, par exemple le prix d’une nuit dans un hôtel, d’un kilomètre, d’un repas de midi…

Un type de dépense est rattachée à 0 ou plusieurs frais.

**Succession des actions :**

Dans ce modèle entité-association on retrouve donc un utilisateur qui va partir en missions dans une ville. Cette mission va avoir un statut qui va évoluer au cours du temps en fonction des actions menée par l’utilisateur ou le comptable.

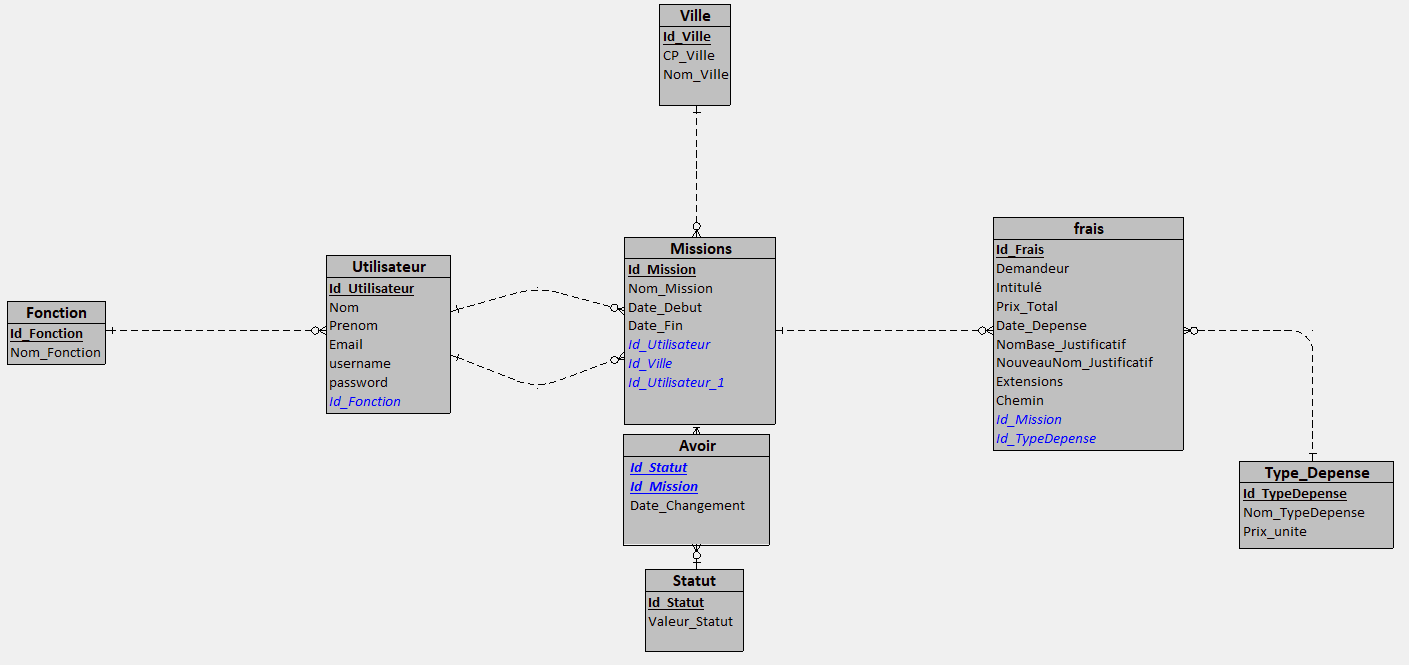
Lors de cette mission l’utilisateur va engendrer différents frais, dont il pourra demander le remboursement en contrepartie d’un justificatif. Ces frais sont rattachés à un type de dépense qui permet de les identifiés et de définir à quelle catégorie de dépense ils appartiennent.

**Les cardinalités :**

* Un utilisateur a une seule fonction
* Une fonction est rattachée à 0 ou plusieurs utilisateurs
* Une mission est créée ou validée par 1 seul utilisateur
* Un utilisateur (fonction utilisateur) peut créer une ou plusieurs missions
* Un utilisateur (fonction comptable) peut valider une ou plusieurs missions
* L’utilisateur qui crée et qui valide la mission ne peut pas être le même grâce à la contrainte d’exclusivité
* Une mission à pour destination une seule ville.
* Une ville peut accueillir 0 ou plusieurs missions.
* Une mission à un ou plusieurs statuts.
* Un statut est rattaché à 0 ou plusieurs missions.
* Une mission engendre 0 ou plusieurs frais.
* Un frais appartient à une seule mission.
* Un frais est associé à un seul type de dépense.
* Un type de dépense est associé à 0 ou plusieurs frais.

## **La modélisation logique**

**Voici le modèle logique des données relationnel schématiser :**



**Voici le modèle logique des données relationnel contextuelle :**

**Type\_Depense = (Id\_TypeDepense *COUNTER***, Nom\_TypeDepense *VARCHAR(50)*, Prix\_unite *DECIMAL(4,2)***);**

**Statut = (Id\_Statut *COUNTER***, Valeur\_Statut *VARCHAR(100)***);**

**Ville = (Id\_Ville *COUNTER***, CP\_Ville *INT*, Nom\_Ville *VARCHAR(100)***);**

**Fonction = (Id\_Fonction *VARCHAR(50)***, Nom\_Fonction *VARCHAR(50)***);**

**Utilisateur = (Id\_Utilisateur *COUNTER***, Nom *VARCHAR(30)*, Prenom *VARCHAR(30)*, Email *VARCHAR(100)*, username *VARCHAR(20)*, password *VARCHAR(255), #Id\_Fonction***);**

**Missions = (Id\_Mission *COUNTER***, Nom\_Mission *VARCHAR(250)*, Date\_Debut *DATE*, Date\_Fin *DATE, #Id\_Utilisateur, #Id\_Ville, #Id\_Utilisateur\_Comptable***);**

**frais = (Id\_Remboursement *COUNTER***, Demandeur *VARCHAR(30)*, Intitulé *VARCHAR(200)*, Prix\_Total *DECIMAL(15,2)*, Date\_Depense *DATE*, NomBase\_Justificatif *VARCHAR(150)*, NouveauNom\_Justificatif *VARCHAR(150)*, Extensions *VARCHAR(10)*, Chemin *VARCHAR(300), #Id\_Mission, #Id\_TypeDepense***);**

**Avoir = (*#Id\_Statut, #Id\_Mission***, Date\_Changement *DATE***);**

**Note** : Pour plus de compréhension et de facilité dans la programmation, la table ‘Avoir’ sera renommée Historique\_Statut dans la BDD et sera équipée d’une clé primaire unique Id\_Historique. En effet, cela permet l’utilisation des modèles Eloquent qui jouent un rôle essentiel dans le développement avec le framework Laravel en simplifiant et rationalisant l'interaction avec la base de données. En utilisant le langage de programmation PHP, Eloquent permet de représenter les tables de la base de données sous forme de classes, où chaque instance de la classe correspond à une ligne de la table. Cette abstraction permet de manipuler les données de manière orientée objet, facilitant ainsi la compréhension et la gestion du code. Les modèles Eloquent offrent une syntaxe expressive pour effectuer des opérations courantes telles que la récupération, la création, la mise à jour et la suppression de données, ce qui permet de réduire la quantité de code nécessaire pour interagir avec la base de données. De plus, Eloquent facilite la mise en œuvre de relations complexes entre les différentes tables de la base de données, améliorant la lisibilité et la maintenance du code. En résumé, l'utilisation des modèles Eloquent dans Laravel simplifie le développement d'applications en fournissant une abstraction efficace de la couche de persistance des données, contribuant à rendre le code plus propre, modulaire et facile à maintenir.

**Justification du niveau de normalisation du modèle relationnel :**

Mon modèle relationnel respecte le premier niveau de normalisation (1FN), en effet tous ses attributs possèdent des valeurs atomiques, par exemple dans la table Utilisateur, le nom et prénom sont enregistrés dans deux colonnes différentes. De plus les valeurs des données d’une colonne sont identiques, par exemple dans la table Type\_Depense, la propriété Prix\_unite est toujours en euro.

Mon modèle respecte le deuxième niveau de normalisation (2FN), en effet les propriétés de chaque table sont identifiées par la clé primaire de la table en question. Par exemple chaque utilisateur de la table Utilisateur est identifié par la clé primaire Id\_Utilisateur. Il en est de même pour la ville ou chaque ville est identifié par la clé primaire Id\_Ville.

Les propriétés des tables Missions et Ville sont séparées, car sinon les propriétés auraient été identifié par le nom de la ville et celui de la mission ce qui aurai fait un clé primaire composite, et cela ne respecte pas le deuxième niveau de normalisation. Il n’y a donc pas de dépendance partielle.

Mon modèle relationnel respecte le troisième niveau de normalisation (3FN), en effet aucune propriété hors clé primaire ne dépend d’une autre propriété hors clé primaire. Il n’y a donc pas de dépendance transitive.

Mon modèle relationnel respecte le niveau de normalisation BOYCE CODD, en effet aucune propriété ne détermine la clé primaire, et toutes les propriétés sont dépendantes de la clé primaire. Cela est valable pour chacune des tables de mon modèle de données relationnel.

## **Script de création de la base de données**

CREATE TABLE Type\_Depense(

Id\_TypeDepense COUNTER,

Nom\_TypeDepense VARCHAR(50),

Prix\_unite DECIMAL(4,2),

PRIMARY KEY(Id\_TypeDepense)

);

CREATE TABLE Statut(

Id\_Statut COUNTER,

Valeur\_Statut VARCHAR(100),

PRIMARY KEY(Id\_Statut)

);

CREATE TABLE Ville(

Id\_Ville COUNTER,

CP\_Ville INT,

Nom\_Ville VARCHAR(100),

PRIMARY KEY(Id\_Ville)

);

CREATE TABLE Fonction(

Id\_Fonction VARCHAR(50),

Nom\_Fonction VARCHAR(50),

PRIMARY KEY(Id\_Fonction)

);

CREATE TABLE Utilisateur(

Id\_Utilisateur COUNTER,

Nom VARCHAR(30),

Prenom VARCHAR(30),

Email VARCHAR(100),

username VARCHAR(20),

password VARCHAR(255),

Id\_Fonction VARCHAR(50) NOT NULL,

PRIMARY KEY(Id\_Utilisateur),

FOREIGN KEY(Id\_Fonction) REFERENCES Fonction(Id\_Fonction)

);

CREATE TABLE Missions(

Id\_Mission COUNTER,

Nom\_Mission VARCHAR(250),

Date\_Debut DATE,

Date\_Fin DATE,

Id\_Utilisateur INT NOT NULL,

Id\_Ville INT NOT NULL,

Id\_Utilisateur\_1 INT NOT NULL,

PRIMARY KEY(Id\_Mission),

FOREIGN KEY(Id\_Utilisateur) REFERENCES Utilisateur(Id\_Utilisateur),

FOREIGN KEY(Id\_Ville) REFERENCES Ville(Id\_Ville),

FOREIGN KEY(Id\_Utilisateur\_comptable) REFERENCES Utilisateur(Id\_Utilisateur)

);

CREATE TABLE frais(

Id\_Frais COUNTER,

Demandeur VARCHAR(30),

Intitulé VARCHAR(200),

Prix\_Total DECIMAL(15,2),

Date\_Depense DATE,

NomBase\_Justificatif VARCHAR(150),

NouveauNom\_Justificatif VARCHAR(150),

Extensions VARCHAR(10),

Chemin VARCHAR(300),

Id\_Mission INT NOT NULL,

Id\_TypeDepense INT NOT NULL,

PRIMARY KEY(Id\_Frais),

FOREIGN KEY(Id\_Mission) REFERENCES Missions(Id\_Mission),

FOREIGN KEY(Id\_TypeDepense) REFERENCES Type\_Depense(Id\_TypeDepense)

);

CREATE TABLE Avoir(

Id\_Statut INT,

Id\_Mission INT,

Date\_Changement DATE,

PRIMARY KEY(Id\_Statut, Id\_Mission),

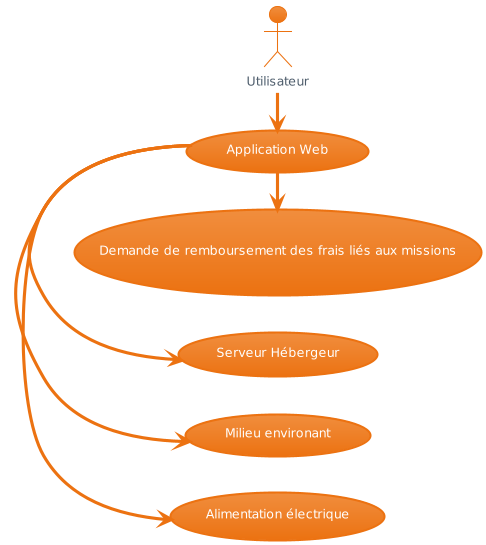
FOREIGN KEY(Id\_Statut) REFERENCES Statut(Id\_Statut),

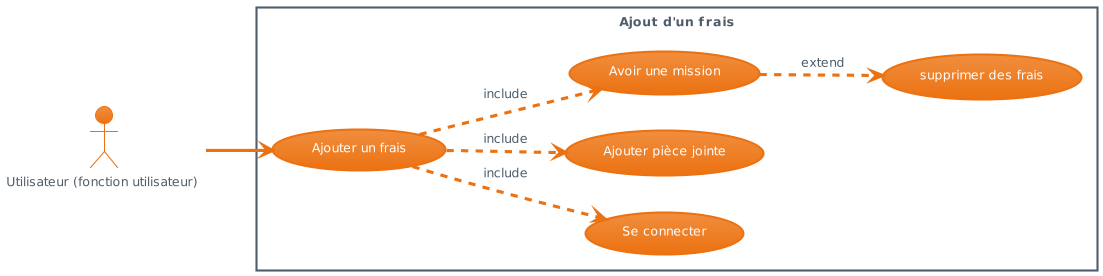
FOREIGN KEY(Id\_Mission) REFERENCES Missions(Id\_Mission)

);

## **Modélisation UML**

**Diagramme de contexte :**



**Diagramme cas d’utilisation d’Ajout d’un frais :**

Objectif :

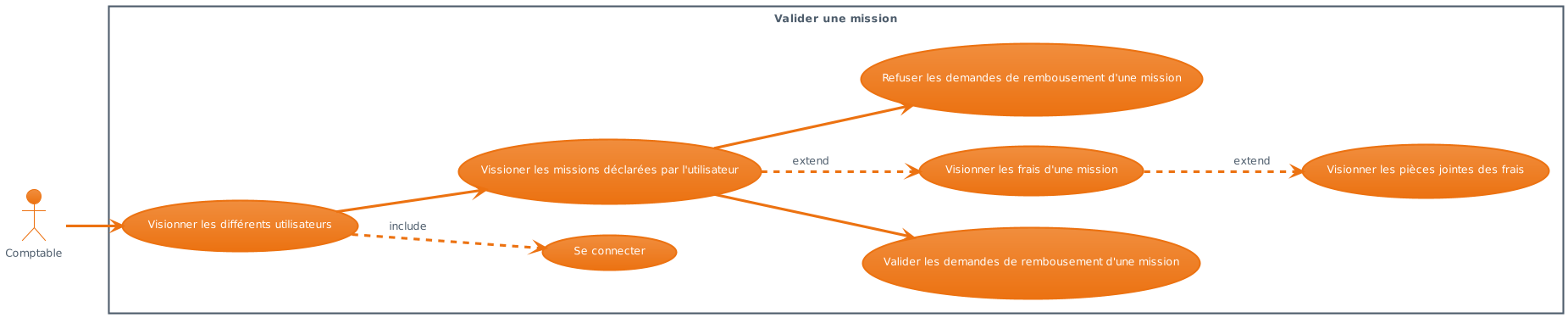
Le but de ce diagramme d’utilisation est de permettre de visualiser comment un utilisateur peut via le site internet, ajouter les frais qu’il a effectué durant la mission afin de demander le remboursement de celle-ci.

Prérequis :

Afin de soumettre une demande de remboursement, l'utilisateur doit disposer d'une connexion Internet pour accéder au site web et se connecter en utilisant son identifiant et son mot de passe. Cette démarche contribue à garantir la sécurité de l'accès au site. De plus, pour inclure des frais, il est nécessaire que l'utilisateur ait préalablement ajouté la mission pour laquelle il souhaite enregistrer ces frais.

Scénario :

Lorsque l’utilisateur accède au site web, la première étape est de se connecter en utilisant son identifiant et son mot de passe. Sans cette démarche, il ne sera pas possible d'effectuer l'ajout d'un frais sur le site. En premier lieu, il doit créer une mission à laquelle il souhaite associer le frais. Une fois cette étape accomplie, il peut cliquer sur la mission pour y accéder. À ce stade, il a la possibilité d'ajouter un frais à la mission en fournissant obligatoirement une pièce justificative de la dépense.

**Diagramme cas d’utilisation de validation d’une mission :**

Objectif :

Le diagramme d'utilisation a pour objectif de fournir une représentation visuelle du processus par lequel un comptable peut observer les divers utilisateurs et leurs missions. Il illustre également la manière dont le comptable peut approuver ou rejeter les demandes de remboursement associées à ces missions.

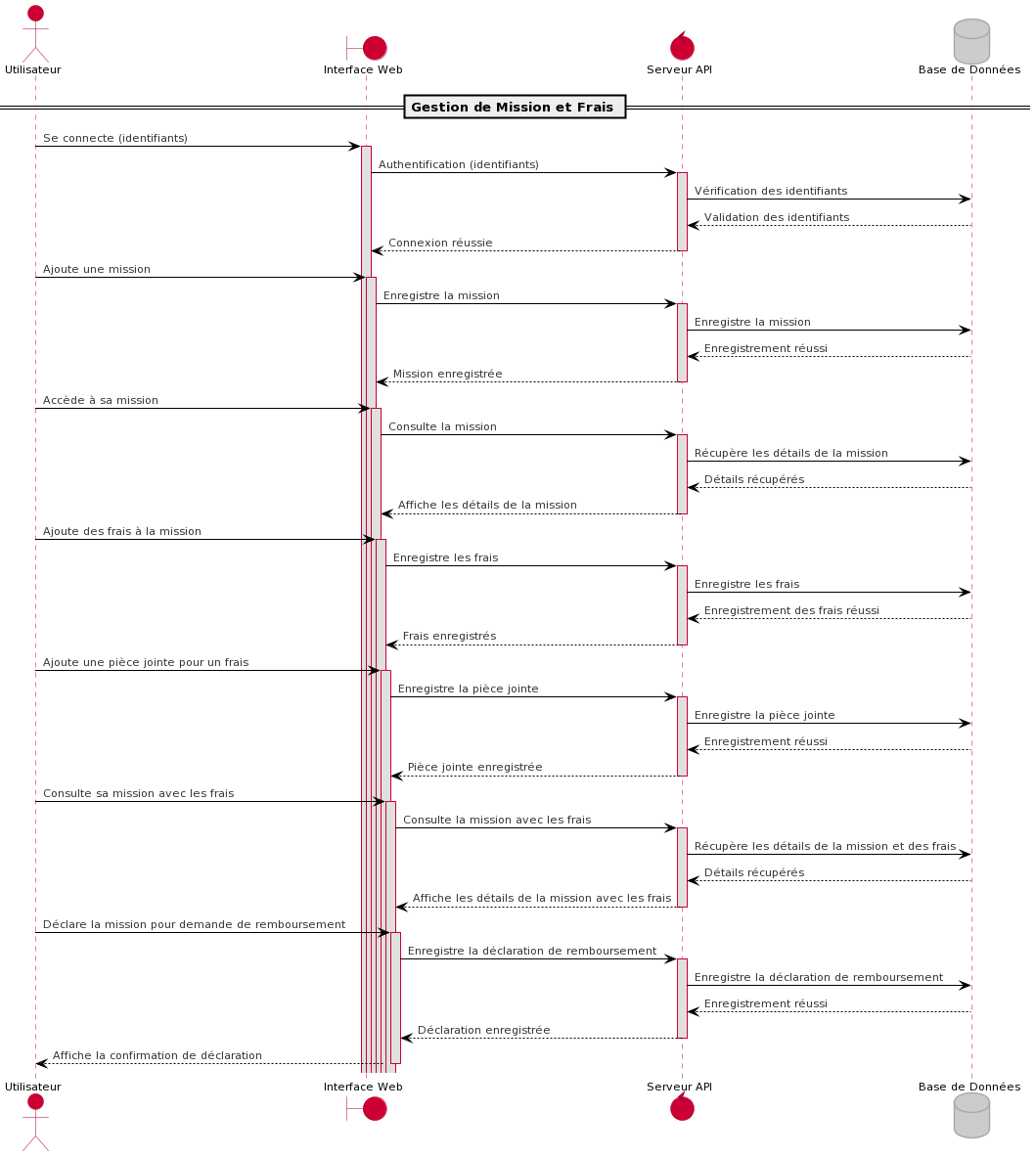
Prérequis :

Afin que le comptable puisse effectuer ces différentes actions il faut qu’il dispose d’une connexion internet et qu’il se connecte via son identifiant et son mot de passe. Cette démarche contribue à garantir la sécurité de l'accès au site.

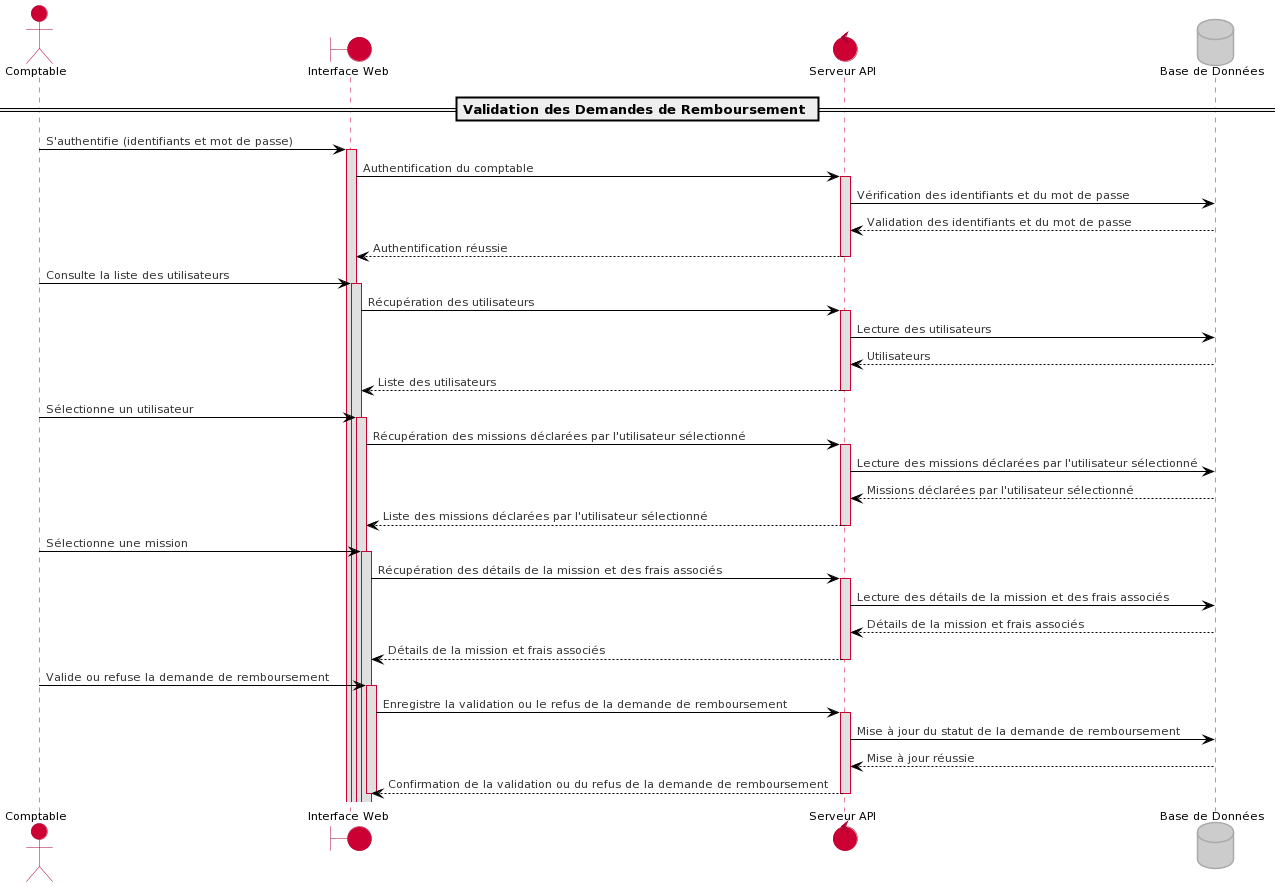
Scénario :

Quand le comptable arrive sur le site web il devra dans un premier temps se connecter via son identifiant et son mot de passe. Sans cette étape, l'accès au site et à ses fonctionnalités est impossible. Une fois connecté, il peut explorer la liste des utilisateurs et choisir un utilisateur spécifique. En sélectionnant cet utilisateur, le comptable peut examiner les missions qu'il a déclarées pour solliciter un remboursement. En accédant à une mission particulière, il a la possibilité de consulter les frais associés, télécharger les pièces justificatives et prendre la décision de valider ou de refuser la demande de remboursement associée à cette mission.

**Diagramme de séquence de création mission jusqu’à déclaration :**



**Diagramme de séquence de validation d’une demande de remboursement par un comptable :**



# **La modélisation de l’IHM et présentation de l’interface**

## **Structure de l’application web**

Pour mon site internet, j’ai opté pour une structure de base qui sera présente dans toute l’application web. Cela facilitera la tâche à l’utilisateur qui pourra très vite prendre l’application web en main et retrouver ses habitudes dans les différentes pages de celle-ci.

Cette structure ce veut simple et épurer afin que l’utilisateur puisse facilement comprendre comment effectuer les actions de base sur l’application web (exemple : ajout de mission pour l’utilisateur, accès à la liste des utilisateurs pour le(s) comptable(s)…).

Voici donc l’IHM de la structure principale de l’application :



On retrouve donc un menu de navigation à gauche permettant à l’utilisateur de facilement naviguer entre les différentes pages. Les boutons de se menu seront propre à la fonction de l’utilisateur, les comptable et les utilisateurs n’auront donc pas les mêmes boutons affichés.

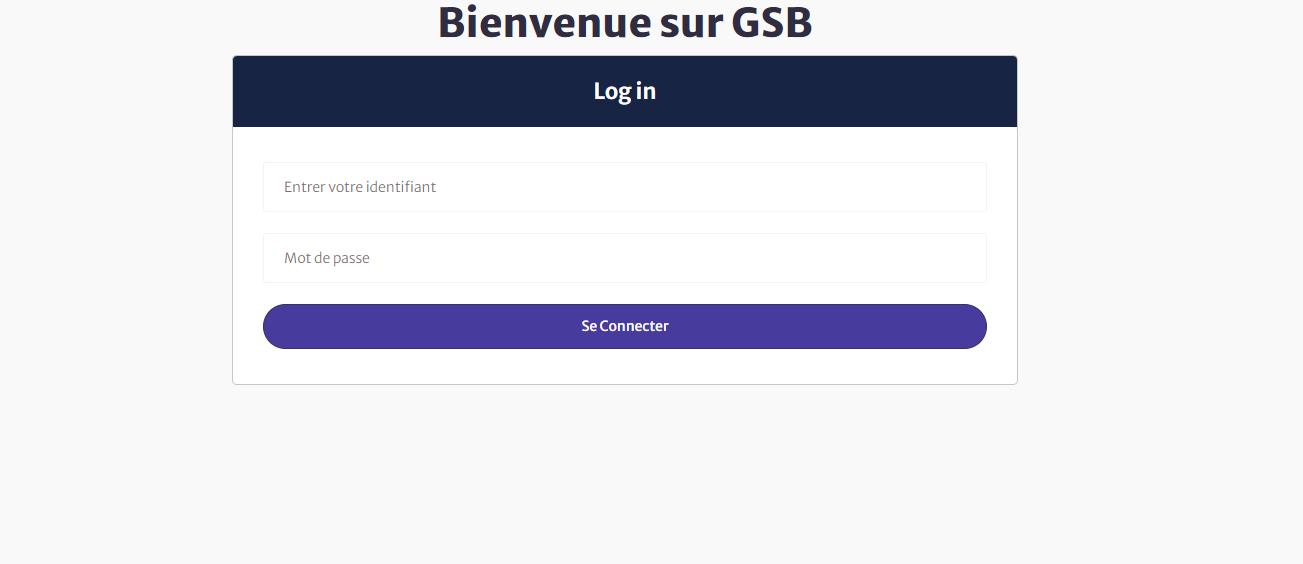
Il y aura également une barre en haut et une en bas, la bar du haut permet l’affichage des informations de l’utilisateur connecté, et de se déconnecter.

Seul le contenu de l’application changera, le reste restera identique de toutes les pages de l’utilisateur.

## **Page de connexion**

Avant d’accéder au différentes pages l’utilisateur devra se connecter afin de garantir la sécurité de l’application web.

Voici l’IHM de la page de connexion :



On pourra y retrouver deux champs permettant à l’utilisateur d’inscrire son identifiant et son mot de passe afin de se connecter.

## **Page accueil d’un utilisateur**

Voici l’IHM de la page d’accueil lorsqu’un utilisateur se connecte :



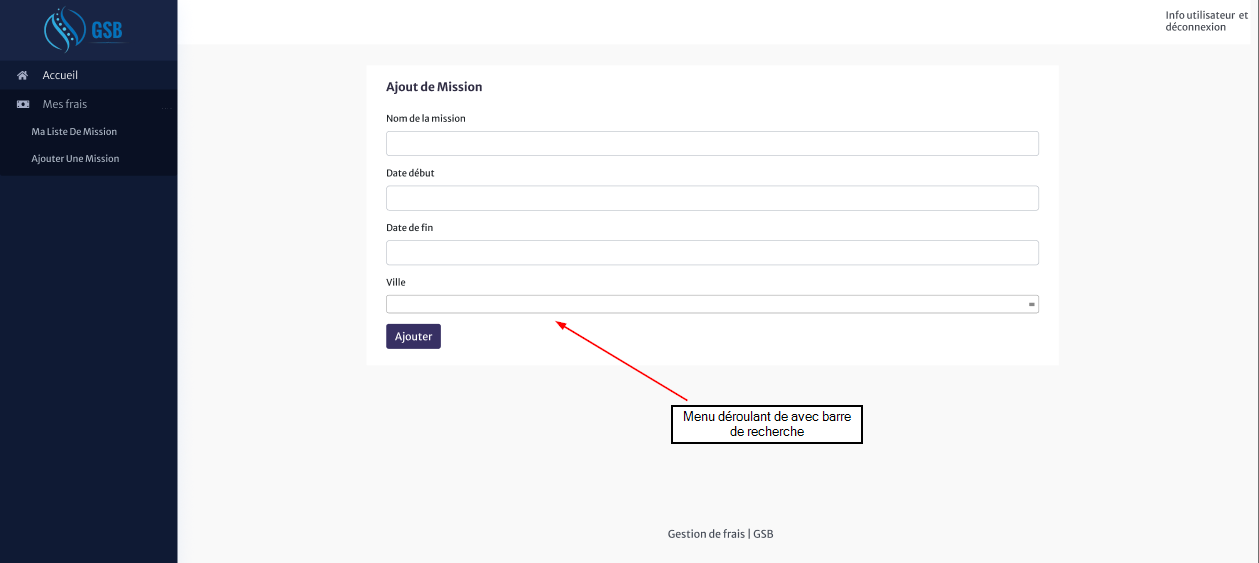
On peut retrouver la structure énoncée précédemment avec le menu déroulant de navigation à gauche, il va permettre à l’utilisateur de naviguer parmi les différentes pages.

Cependant d’afficher les pages importantes directement à l’utilisateur, deux boutons sont également présents au milieu de la section « contenue ». Cela permet plus d’ergonomie et une facilité de compréhension et d'utilisation.

Le bouton accueil du menu de gauche va permettre à l’utilisateur de revenir à tout moment sur cette page d’accueil.

## **Page d’ajout d’une mission**

Si l’utilisateur clique sur le bouton « Ajouter une mission » depuis la page d’accueil ou depuis le menu déroulant, il sera dirigé vers un formulaire d’ajout d’une mission, voici l’IHM de cette page :

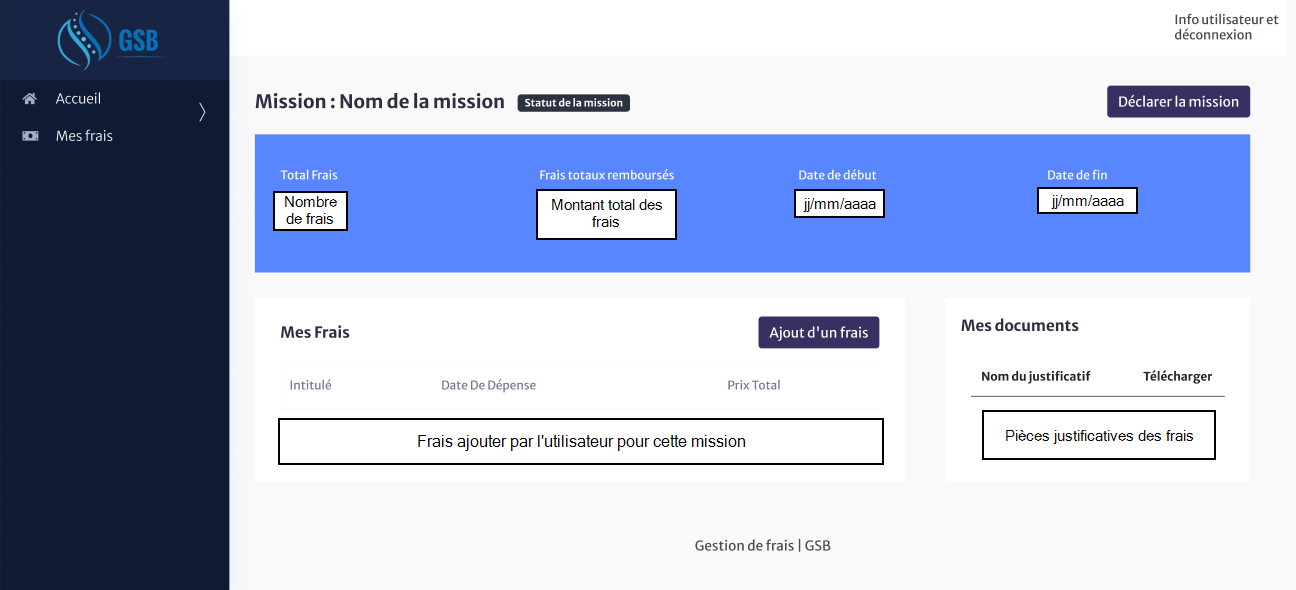


On peut donc voir un formulaire comportant différents champs qui devront obligatoirement être rempli pour ajouter une mission.

Une fois le bouton « Ajouter » cliquer, l’utilisateur sera directement redirigé vers la mission qu’il vient de créer, cela va lui permettre une simplicité de prise en main et d’utilisation.

## **Page de la mission**

Voici l’IHM de la page de la mission :



On peut rapidement y voir toutes les informations de la mission, on va pouvoir y retrouver le nom donner à la mission ainsi que son statut.

Dans le bandeau on pourra retrouver les dates de début et fin de la mission ainsi que des informations sur les frais liés à celle-ci.

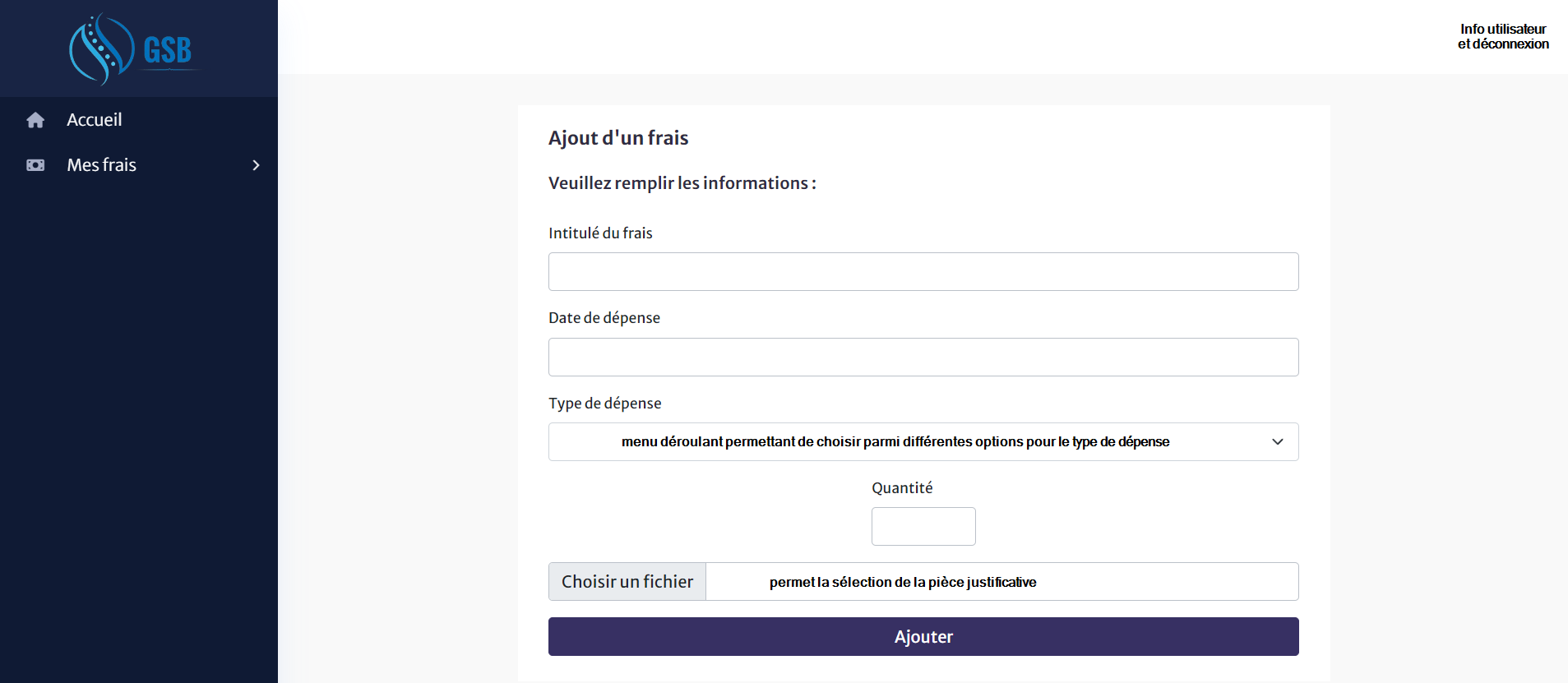
Puis on pourra retrouver le tableau comportant les frais de la mission et un tableau comportant les justificatifs des frais.

Cette disposition permet une lecture rapide de toutes les informations.

On peut également y retrouver un bouton « Ajout d’un frais » permettant d’ajouter un nouveau frais à la mission et un bouton « Déclarer la mission » permettant de passer la mission au statut « en attente de validation » qui permettra à la mission d’apparaitre dans le visuel du comptable.

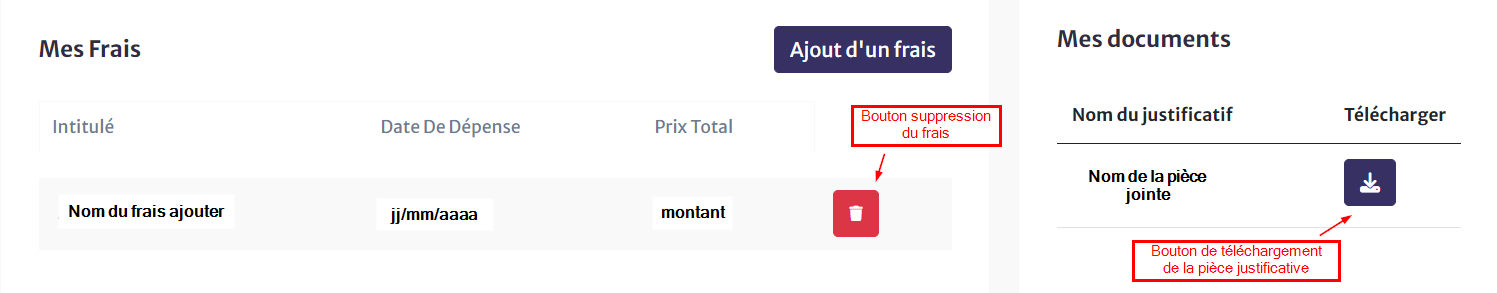
## **Page d’ajout d’un frais**

Lorsque depuis la page d’affiche de la mission l’utilisateur clique sur le bouton « Ajout d’un frais » il est redirigé vers un formulaire donc voici l’IHM :



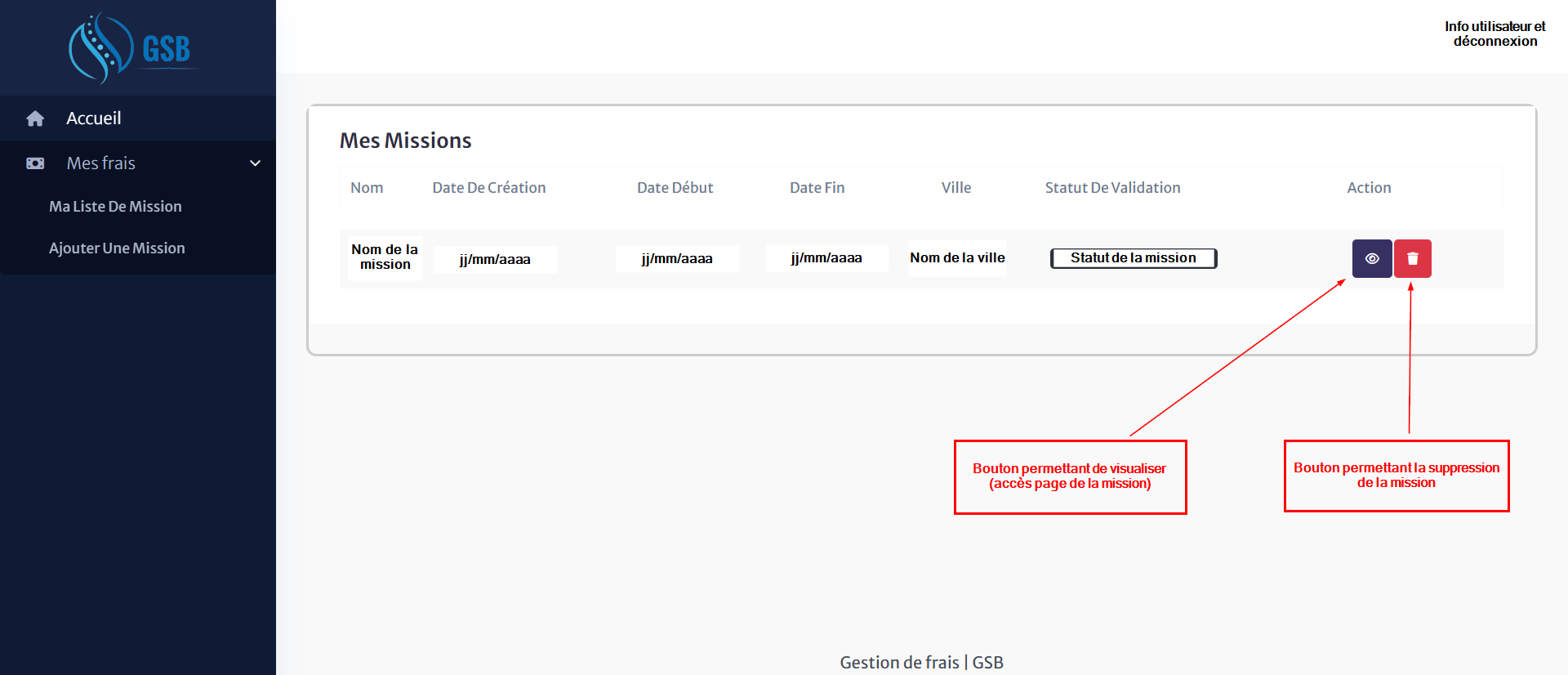
Sur cette page, on peut retrouver différents champs qui devront être obligatoirement remplis, ainsi que l’ajout d’une pièce justificative afin d’ajouter un frais.

Une fois l’ajout du frais effectuer, l’utilisateur sera redirigé vers la page mission ou de nouvelles informations apparaitront :



## **Page de liste des missions**

Lorsque l’utilisateur clique sur le bouton « Ma Liste De Mission » présent sur le menu de navigation de gauche ou sur la page d’accueil, il sera redirigé vers une liste de toute ses missions créer dont voici l’IHM :



Cette liste sera vide si l’utilisateur n’a pas encore ajouter de mission.

Cette liste permettra simplement de visualiser l’avancement des différentes missions (avec leur statut), ainsi que d'accéder à celles-ci ou de les supprimer.

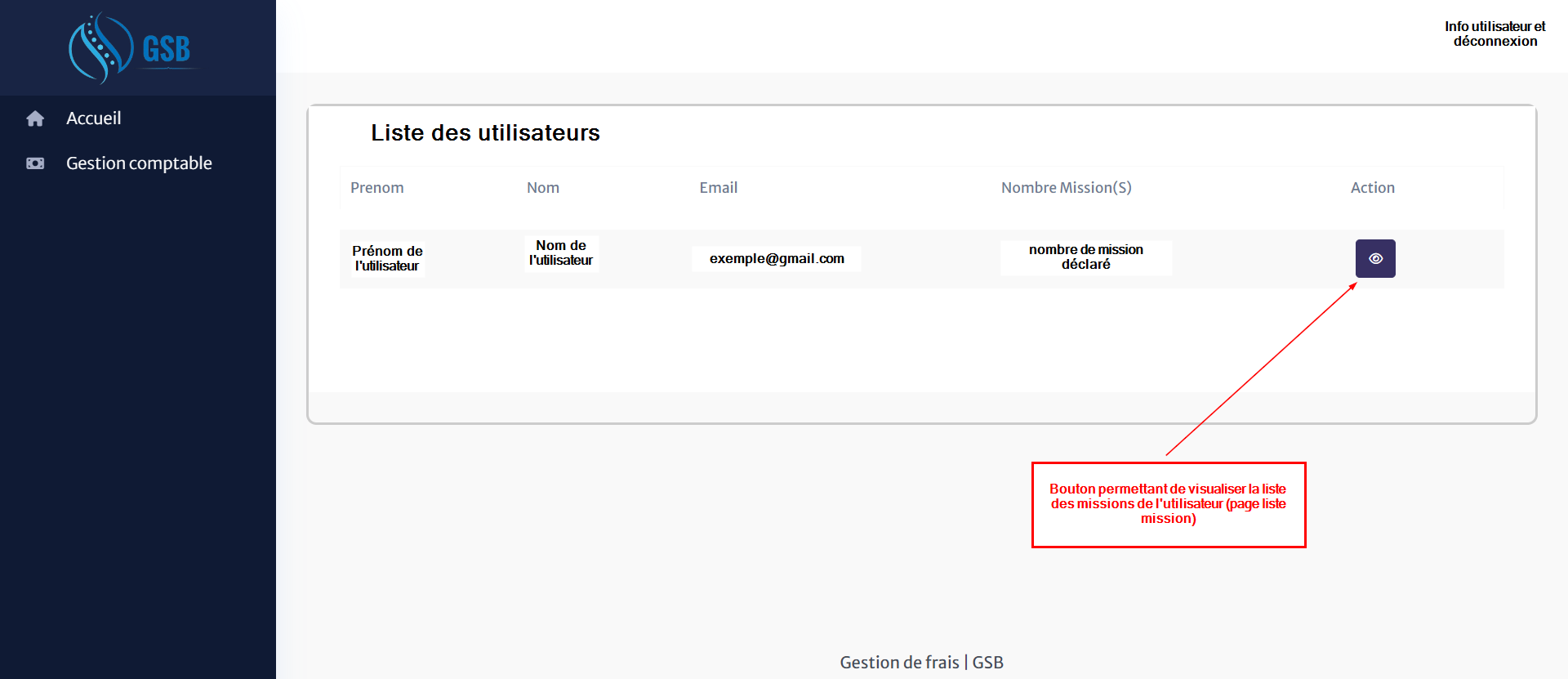
## **Page d’accueil d’un comptable**

Lorsqu’un comptable se connecte un possède un page d’accueil légèrement différente d’un utilisateur, voici l’IHM :



Le comptable retrouve une application très facile à prendre en main, qui va permettre étape par étape de valider des missions, la première étape consistant à visualiser la liste de tous les utilisateurs. On retrouve deux boutons pour accéder à cette liste sur la page d’accueil.

## **Page liste des utilisateurs pour le comptable**

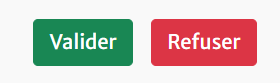
En cliquant sur le bouton « Gestion comptable » du menu de navigation ou encore « Liste des visiteurs » de la page d’accueil, le comptable sera redirigé vers cet IHM :

Cette liste permet une vue d'ensemble de tous les utilisateurs et de leur nombre de demandes de remboursement (mission(s) déclarée(s)).

## **Validation ou refus de mission**

Une fois l'utilisateur sélectionné, le comptable pourra visualiser la liste de toutes les missions qu’il a déclarées (page liste mission), puis chaque mission individuellement pour la valider ou refuser (page de la mission).

En tant que comptable sur la page de la mission, deux nouveaux boutons apparaîtront afin de valider ou refuser la demande de remboursement :



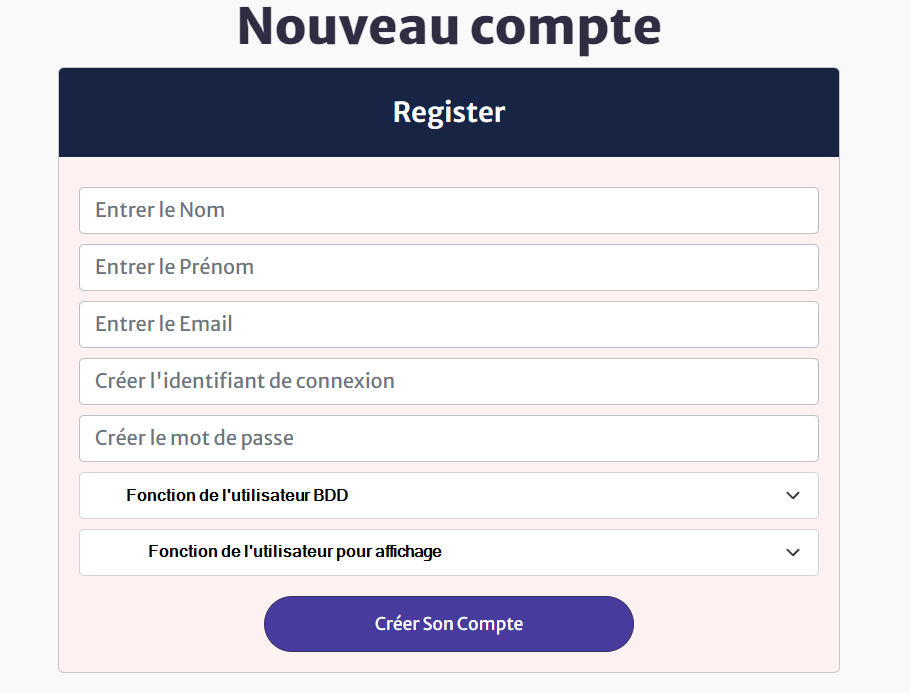
## **Page d’accueil pour comptable**

Il existe également une interface pour le développeur lui permettant d’ajouter un utilisateur et de lui attribuer sa fonction, voici la page d’accueil de l’interface développeur/admin :



## **Page d’ajout d’un utilisateur**

Voici l’IHM d’ajout d’un compte utilisateur :



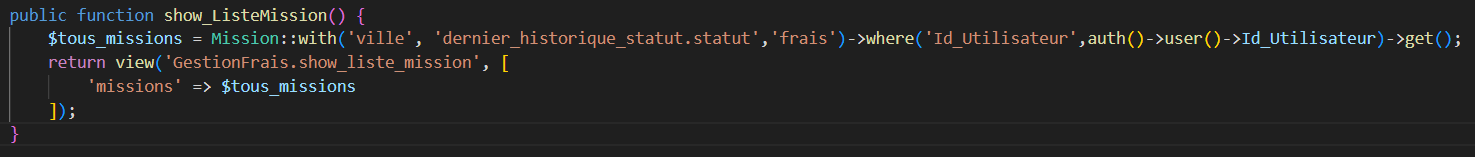
Il permet au développeur de choisir un mot de passe sécurisé pour les nouveaux utilisateurs.

La fonction de l’utilisateur pour l’affichage permet de dire à l'application le « rôle de l’utilisateur » et ainsi lui afficher uniquement les éléments correspondant à sa fonction dans la BDD.

# **Présentation des principales requêtes développées.**

**Sélection de toutes les missions d’un utilisateur**

Voici la requête permettant de sélectionner toute les missions qu’un utilisateur à créer :

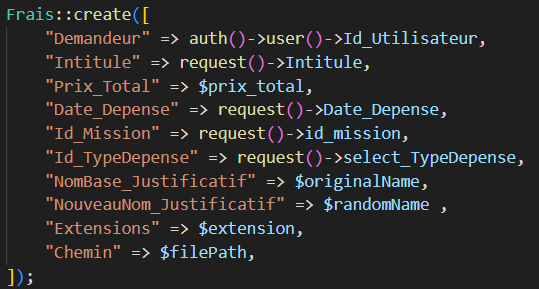


Cette requête SQL, codée dans le framework Laravel, sert à récupérer un ensemble de données liées aux missions. La variable `$tous\_missions` stocke le résultat de la requête, qui est obtenue à partir du modèle Eloquent "Mission" (table Mission de la BDD). La méthode `with` est utilisée pour spécifier les relations à charger en même temps que les missions, notamment les relations "ville", "dernier\_historique\_statut.statut" et "frais. Ensuite, la clause `where` filtre les missions en fonction de l'identifiant de l'utilisateur actuellement authentifié. Enfin, la méthode `get` exécute la requête et renvoie les résultats. Ainsi, cette requête récupère toutes les missions associées à l'utilisateur authentifié avec des informations supplémentaires sur la ville, le statut le plus récent de la mission et les frais associés.

Une fois la requête effectuée, la page d’affichage de la liste des missions est affichée a l’utilisateur, et les données récupérée avec la requête seront transférer avec la page afin d’afficher la liste des missions de l’utilisateur.

## **Création d’un nouveau frais**

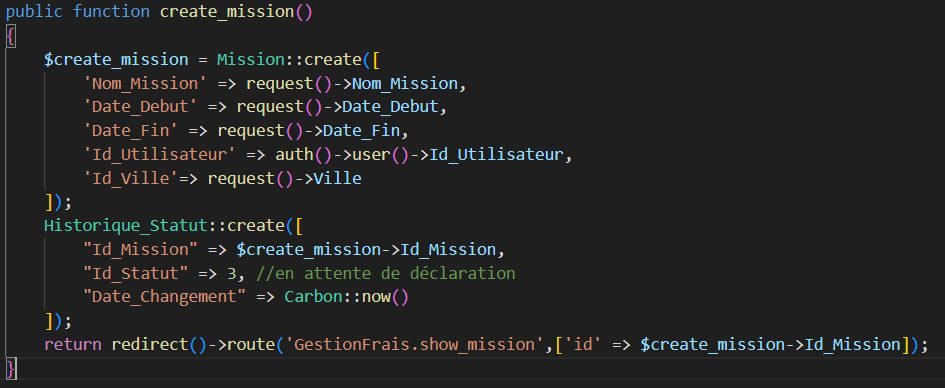
Voici la requête permettant de créer nouveau frais dans la base de données :



La requête `Frais::create` crée une nouvelle entrée dans la table "frais" de la base de données en utilisant le modèle Eloquent "Frais" dans le contexte du framework Laravel. Les données à insérer sont fournies grâce au formulaire d’ajout de frais et servent en tant que paramètre de la méthode create. Plus précisément, les données incluent le demandeur (utilisateur actuellement authentifié), l'intitulé de la dépense, le prix total calculé en fonction du type de dépense et de la quantité ou du montant, la date de la dépense, l'identifiant de la mission associée, l'identifiant du type de dépense, le nom original et le chemin du justificatif de frais, ainsi que des informations sur l'extension du fichier. En outre, la méthode de création Eloquent prend soin de l'insertion sécurisée de ces données dans la base de données, évitant ainsi les vulnérabilités potentielles liées aux attaques d'injection SQL.

## **Création d’une nouvelle mission**

Voici la requête permettant de créer une nouvelle mission dans la base de données et de lui attribuer un statut :



La première requête, Mission::create, insère les données de la nouvelle mission dans la table "missions" de la base de données en utilisant le modèle Eloquent "Mission". Les données sont extraites des champs du formulaire d’ajout de mission, notamment le nom de la mission, les dates de début et de fin, l'identifiant de l'utilisateur actuellement authentifié, et l'identifiant de la ville associée à la mission. Ensuite, une deuxième requête Historique\_Statut::create crée une entrée dans la table "historique\_statuts", établissant le statut initial de la mission nouvellement créée. Dans cet exemple, le statut est fixé à 3, ce qui correspond à l'état "en attente de déclaration". Enfin, la fonction redirige l'utilisateur vers la page d'affichage des détails de la mission nouvellement créée. En résumé, ces requêtes permettent de créer une mission, d'initialiser son statut et de rediriger l'utilisateur vers les détails de la mission fraîchement créée.

# **Présentation du code développé dans l’application**

Tout le code de l’application peut se trouve dans mon github : <https://github.com/99Noah99/Projet_GSB>

Mon application est codée avec le framework laravel, l’application est donc codée selon le modèle MVC.

Le modèle MVC (Modèle-Vue-Contrôleur) est un motif architectural largement utilisé dans le développement de logiciels, visant à organiser le code de manière structurée et modulaire.

Dans ce modèle, le Modèle représente la logique métier et la gestion des données. Il traite des opérations sur la base de données, de la manipulation des données, et encapsule la logique métier spécifique à l'application.

La Vue est responsable de l'interface utilisateur et de l'affichage des données. Elle présente les informations au format approprié pour l'utilisateur et réagit aux événements de l'interface.

Enfin, le Contrôleur agit comme un intermédiaire entre le Modèle et la Vue. Il gère les entrées utilisateur, invoque les opérations appropriées sur le Modèle, et met à jour la Vue en conséquence.

La séparation claire de ces trois composants offre plusieurs avantages, notamment une meilleure maintenabilité, une réutilisation du code, et une facilité de collaboration entre les développeurs travaillant sur différentes parties de l'application. Le modèle MVC est largement adopté dans de nombreux frameworks, dont Laravel pour PHP, pour structurer de manière efficace le code des applications web.

Dans Laravel, le modèle MVC (Modèle-Vue-Contrôleur) est implémenté comme suit :

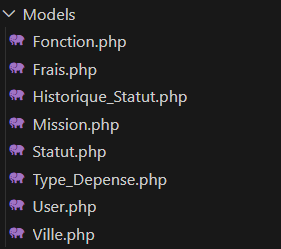
1. Modèle (Model) : Les modèles dans Laravel sont des classes Eloquent qui représentent les tables de la base de données. Ils sont responsables de l'accès aux données, de la manipulation de la logique métier et de la gestion des relations entre les tables de la base de données.

Voici mes différents modèles dans mon code laravel (correspond aux tables de ma BDD) :

1. Vue (View) : Les vues dans Laravel sont généralement des fichiers Blade. Elles sont responsables de la présentation des données et de l'interface utilisateur. Les vues affichent les informations au format HTML et peuvent utiliser le langage de templating Blade pour intégrer dynamiquement des données.
2. Contrôleur (Controller) : Les contrôleurs dans Laravel sont des classes qui gèrent la logique de l'application en traitant les requêtes HTTP. Ils agissent comme des intermédiaires entre les modèles et les vues. Les méthodes des contrôleurs sont appelées pour répondre à des actions spécifiques, comme l'affichage d'une page ou la manipulation de données.

## **Les modèles dans laravel (Eloquent)**

Les modèles permettent donc de représenter les tables de ma base donnée dans mon code. Voici donc tous mes modèles Eloquent dans mon code :



On retrouve donc toute les tables de ma BDD présentent dans le modèle entité-association.

Voici pour exemple le code du modèle correspondant à ma table mission :



Ce code représente donc un modèle Eloquent dans Laravel, spécifiquement pour l'entité "Mission". Les modèles Eloquent facilitent l'interaction avec la base de données en permettant de manipuler les enregistrements de manière orientée objet. Dans ce cas, la classe Mission étend la classe de base Model de Laravel.

Les propriétés protégées telles que $table, $primaryKey, $guarded et $dates sont utilisées pour définir le nom de la table, la clé primaire, les colonnes protégées (qui peuvent être mass-assignées), et les colonnes de dates respectivement. La clé primaire est définie sur 'Id\_Mission'.

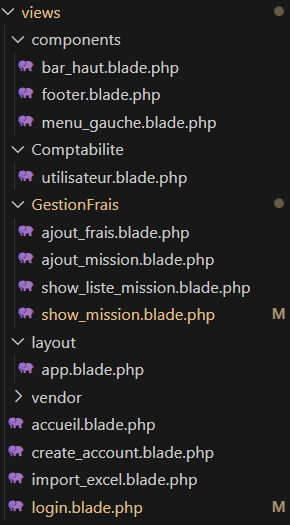
On ne retrouve pas forcément dans tous les modèles les colonnes de dates, les seules colonnes présentent dans tous les modèles Eloquent sont les propriétés protégées $table et $primaryKey.

Par la suite, la classe Mission déclare plusieurs relations avec d'autres modèles. La méthode ville() définit une relation de type "hasOne" avec le modèle Ville (table Ville), indiquant que chaque mission est associée à une seule ville. Les méthodes frais() (table Frais), historique\_statut() (table Historique\_Frais), et dernier\_historique\_statut() définissent des relations de type "hasMany" et "hasOne" respectivement avec les modèles Frais et Historique\_Statut. Ces relations reflètent la structure des données dans la base de données, facilitant la récupération d'informations liées aux missions, villes, frais, et historique des statuts. En particulier, la méthode dernier\_historique\_statut() utilise la fonction latest() pour récupérer le dernier enregistrement de l'historique des statuts lié à la mission, basé sur la date de changement.

Pour résumé, ce modèle Eloquent simplifie l'accès et la manipulation des données liées aux missions dans une application Laravel.

## **Les vues**

Voici toutes les vues présentent dans mon code :



Les vues correspondent à ce qui est affiché à l’utilisateur.

Comme on a pu le voir dans la section IHM, les pages visualiser par l’utilisateur sont construit selon une structure, avec un menu gauche, une barre du haut et une barre en bas. Ce sont ses éléments que l’on retrouve dans le dossier « components »

**La construction de la vue :**

La vue est donc constituée différents composants (menu gauche…), le composant principale est le contenu de la page affiché (contenu que l’on retrouve dans l’IHM).

Tous les fichiers autres que les composants correspondent donc à ce qui sera affiché dans la section contenu de la structure de la page vu dans l’IHM.

Voici comment la page affichée est construite :

1. Premièrement l’utilisateur appelle une vue suite à une action ou autre dans notre cas on va prendre l’exemple de l’affichage de la liste des missions
2. Au début du fichier de la vue liste mission () on retrouve ce bout de code :



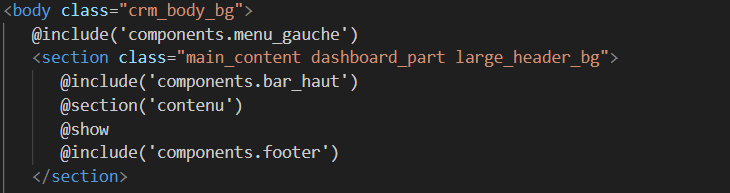
Et à la fin du fichier on retrouve ce code :



L'instruction « @extends('layout.app') » dans le fichier show\_liste\_mission.blade.php de Laravel indique que la vue actuelle (le fichier Blade en cours) étend un autre fichier de mise en page appelé "app.blade.php". En d'autres termes, cela signifie que la vue actuelle utilisera la structure définie dans le fichier "app.blade.php" en tant que mise en page principale. Le fichier "app.blade.php" contient donc la structure HTML de base, le code commun à toutes les pages de l'application, comme le menu de navigation, la barre du haut et la barre du bas.

`@extends`, permet d’éviter la duplication de code en définissant une mise en page générique que plusieurs vues peuvent utiliser. Cela favorise la maintenabilité du code en permettant des modifications centralisées de la structure commune sans avoir à les répéter dans chaque fichier de vue individuel.

1. Dans le fichier « app.blade.php » on peut donc retrouver ce code :



Tout d'abord, l'instruction `@include('components.menu\_gauche')` est utilisée pour incorporer le menu de navigation. Cela permet d'ajouter un menu gauche à la page, réutilisant du code défini dans ce fichier.

Ensuite l'instruction `@include('components.bar\_haut')` est utilisée pour inclure le contenu du fichier "bar\_haut.blade.php", ajoutant ainsi une barre en haut de la page.

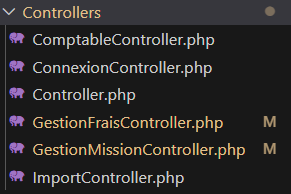
La `@section('contenu')` indique le début d'une section nommée "contenu". Dans cet exemple la section sera emplie par le fichier show\_liste\_mission.blade.php qui étendent celui-ci. La directive `@show` marque la fin de la section "contenu", et tout ce qui se trouve entre `@section('contenu')` et `@show` sera remplacé dans les fichiers qui étendent ce modèle (ici show\_liste\_mission.blade.php).

Enfin, l'instruction `@include('components.footer')` est utilisée pour inclure le contenu du fichier "footer.blade.php", ajoutant ainsi un pied de page à la page.

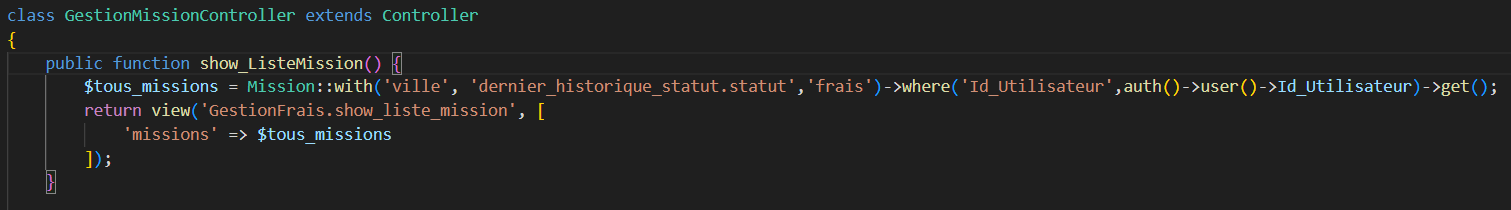
Dans l'ensemble, ce code favorise la modularité en incorporant des composants spécifiques à des endroits précis de la page. Il suit également la structure de base d'une page avec un menu gauche, une barre en haut, une section de contenu, et un pied de page.

## **Les contrôleurs**

Les contrôleurs reçoivent les requêtes de l'utilisateur, traitent ces demandes en effectuant les opérations nécessaires du point de vue des données… et passent ensuite les données nécessaires aux vues pour l'affichage. Voici les contrôleurs de mon application web :



Dans notre cas on regarde le contrôleur « GestionMissionController.php », on peut y trouver la fonction « show\_ListeMission() » :



Comme on peut le voir, la fonction effectue d’abord des traitements de données, ici la récupération de toutes les missions créées par l’utilisateur et des différentes informations liées à cette mission.

Une fois les données récupérées, le contrôleur renvoie la vue show\_liste\_mission avec les données concernant les missions.

## **Schéma du modèle MVC :**

