**DOCUMENTATION TECHNIQUE : APPLICATION GESTION AGENCE DE VOYAGE VOY’IN**

[I – Adéquation du besoin et de la persistance des données 2](#_Toc497750019)

[II – Accès distant à l’Architecture applicative 3](#_Toc497750020)

[III – Architecture applicative 3](#_Toc497750021)

[IV – Spécification des classes 4](#_Toc497750022)

[A – Les classes métiers 4](#_Toc497750023)

[1 – La classe guide 4](#_Toc497750024)

[2 – La classe langue 5](#_Toc497750025)

[3 – La classe pays 6](#_Toc497750026)

[4 – la classe activite 7](#_Toc497750027)

[B – Les classes conteneurs 8](#_Toc497750028)

[1 – La classe conteneurGuide 8](#_Toc497750029)

[2 – La classe conteneurPays 9](#_Toc497750030)

[3 – La classe conteneurLangue 9](#_Toc497750031)

[4 – La classe conteneurActivite 10](#_Toc497750032)

[C – la classe gestionVoyage 10](#_Toc497750033)

[C – la classe controleur 11](#_Toc497750034)

[D - Le fichier index 13](#_Toc497750035)

[D – la classe accessBD 14](#_Toc497750036)

# I – Adéquation du besoin et de la persistance des données

Compte tenu du cahier des charges référence GEST\_VOYAGE\_2020 la persistance de données suivante a été validé.

utilisateur ( id, login, password, grade)

clé primaire : id

guide (id, nom)

clé primaire : id

langue ( id, Libelle)

clé primaire : id

parler (idGuide, idLangue)

clé primaire : idGuide, idLangue

clé étrangère : idGuide en référence à id dans Guide

clé étrangère : idLangue en référence à id dans Langue.

pays (code, nom,photos,idLangue)

clé primaire : code

clé étrangère : idLangue en référence à id dans la table langue;

ville (id,nom, idPays)

clé primaire : id

clé étrangère : idPays en références code dans la table pays;

voyage (id, nom, idVilleArrivee)

clé primaire : id

clé étrangère : idVilleArrivee en reference à id dans la table ville;

guideParticipeVoyage (idGuide,idVoyage)

clé primaire : idGuide,idVoyage

clé étrangère : idGuide en référence ) id dans la table guide

clé étrangère : idVoyage en référence à id dans la table voyage;

villeDepartVoyage (idVoyage, idVille)

clé primaire : idVoyage,idVille

clé étrangère : idVoyage en référence à id dans la table voyage

clé étrangère : idVille en référence à id dans la table ville

instanceVoyage (id, idVoyage, dateDeb, dateFin, nbPlace)

clé primaire : id,idVoyage

clé étrangère : idVoyage en référence à id dans la table voyage

client (id, nom, prenom, adresse)

clé primaire : id

clientInscritVoyage (idNmVoyage, idClient, nbPlaceReserve)

clé primaire : idNmVoyage,idClient

clé étrangère : idNmVoyage en référence à id dans la table instanceVoyage,

clé étrangère : idClient en référence à id dans la table client

activite (idActivite, libelleActivite)

clé primaire l idActivite

Le script de la base de données porte le nom : ***001.3 - Script de la base Voy’in.sql***

Un jeu d’essai a été mit en place il est contenu dans le script de création de la base de données. Il tient compte des éléments fournit dans le cahier des charges.

# II – Accès distant à l’Architecture applicative

La base de données s’appelle ppe2voyage. Elle est implémentée sous MySQL.

# III – Architecture applicative

L’architecture de l’application respecte le modèle MVC (model – view – controler), c’est-à-dire que les couches traitements, données et éléments d’affichage sont séparés.

L’organisation des répertoires dans le projet est la suivante :

* Répertoire vues :
  + Contenant l’ensemble des écrans de l’application. La nomenclature de nommage est la suivante :
    - Action suivi du nom de l’objet sur lequel on applique l’action. Exemple écran ayant pour objectif l’ajout d’un guide s’appelle ajouterGuide…
* Répertoire modeles :
  + Contenant l’ensemble des traitements de l’application. Il contient les sous-repertoires suivants :
    - Metiers : contenant l’ensemble des classes métiers. Elle porte le nom de la entité au singulier : activite.php ; guide.php ; langue.php ; pays.php. Chacune des classes contient un constructeur permettant l’implémentation d’un objet puis les getteurs.
    - Conteneurs : contenant l’ensemble des classes arrayObject des classes métiers. Ainsi la classe métier guide possède une classe conteneurGuide ayant un attribut lesGuides de type arrayObject. Chacune des classes possèdent sa classe conteneur.
    - Enfin, à la racine du répertoire méthode nous pouvons trouver le fichier accesBD contenant la classe accesBD permettant la connexion à la base de données et l’implémentation des conteneurs, ainsi que la mise à jour de la base de données. On peut aussi trouver le fichier gestionVoyage.php qui contient l’ensembles des méthodes permettant les traitements spécifiques.
* Le fichier contrôleur : il est à la racine de l’application, c’est notre aiguilleur. Tous les messages passent par lui.

Nous pouvons schématiser l’architecture applicative par le modèle en ***annexe 1***.

La description détaillée de chacune des fonctionnalités est dans les différents fichiers explicatifs.

# IV – Spécification des classes

## A – Les classes métiers

## 1 – La classe guide

Cette classe a pour objectif d’être le générateur des objets de type guide. Elle va permettre l’implémentation d’objets qui auront pour spécificité de contenir l’id, le nom du guide, ainsi que l’ensemble des langues qu’il pratique et les activités qu’il est capable de faire. Pour l’instant seul les getteurs sont en place.

Le constructeur permet de créer le guide en faisant passer le nom et l’id du guide. Dans un premier temps les langues et les activités sont vides.

N’oublions pas que nous devons pouvoir ajouter des langues et des activités, nous auront donc deux méthodes le permettant.

<?php

Class guide

{

private $idGuide;

private $nomGuide;

private $sesLangues;

private $sesActivites;

public function \_\_construct($unIdGuide, $unNomGuide)

{

$this->idGuide = $unIdGuide;

$this->nomGuide = $unNomGuide;

$this->sesLangues = new arrayObject();

$this->sesActivites = new arrayObject();

}

public function getIdGuide()

{

return $this->idGuide;

}

public function getNomGuide()

{

return $this->nomGuide;

}

public function getSesLangues()

{

$liste = '';

foreach ($this->sesLangues as $uneLangue)

{

$liste = $liste.$uneLangue->getLibelleLangue().' - ';

}

return $liste;

}

public function ajouteUneLangue($uneLangue)

{

$this->sesLangues->append($uneLangue);

}

public function ajouteUneActivite($uneActivite)

{

$this->sesActivites->append($uneActivite);

}

}

?>

## 2 – La classe langue

Cette classe a pour objectif d’être le générateur des objets de type langue. Elle va permettre l’implémentation d’objets qui auront pour spécificité de contenir l’id, le libellé et les différents guides parlant cette langue. Les getteurs permettent l’accès aux différents éléments.

Le constructeur permet de créer des objets en faisant passer en paramètre l’id et le libellé de la langue. Dans un premier temps la liste des guides est vide.

N’oublions pas que nous devons pouvoir ajouter des guides, d’où l’attributs sesGuides avec la méthode ajouteUnGuide(unGuide).

<?php

Class langue

{

private $idLangue;

private $libelleLangue;

private $sesGuides;

public function \_\_construct($unIdLangue,$unLibelleLangue)

{ $this->idLangue = $unIdLangue;

$this->libelleLangue = $unLibelleLangue;

$this->sesGuides = new arrayObject();

}

public function getIdLangue()

{

return $this->idLangue;

}

public function getLibelleLangue()

{

return $this->libelleLangue;

}

public function ajouteUnGuide($unGuide)

{

$this->sesGuides->append($unGuide);

}

}

?>

## 3 – La classe pays

Cette classe a pour objectif d’être le générateur des objets de type pays. Elle va permettre l’implémentation d’objets qui auront pour spécificité de contenir l’id, le nom, la photo, ainsi que la langue parlé d’origine du pays. De manière à simplifier le système on part du principe qu’il n’y a qu’une langue parlé par pays. Les getteurs permettent l’accès aux différents éléments.

Le constructeur permet de créer des objets de type Pays.

L’attribut saLangue va contenir un objet de type langue.

<?php

Class pays

{

private $idPays;

private $nomPays;

private $photoPays;

private $saLangue;

public function \_\_construct($unIdPays, $unNomPays, $photoPays,$saLangue)

{

$this->idPays = $unIdPays;

$this->nomPays = $unNomPays;

$this->photoPays = $photoPays;

$this->saLangue = $saLangue;

}

public function getIdPays()

{

return $this->idPays;

}

public function getNomPays()

{

return $this->nomPays;

}

public function getPhotoPays()

{

return $this->photoPays;

}

public function getSaLangue()

{

return $this->saLangue;

}

}

?>

## 4 – la classe activite

Cette classe a pour objectif d’être le générateur des objets de type activite. Elle va permettre l’implémentation d’objets qui auront pour spécificité de contenir l’id, le libellé, ainsi que le l’ensemble des guides pouvant effectuer ses activités. Les getteurs permettent l’accès aux différents éléments.

Nous avons un constructeur qui va permettre de créer des objets de type activite. Dans un premier temps la liste des guides sera vide.

<?php

Class activite

{

private $idActivite;

private $libelleActivite;

private $sesGuides;

public function \_\_construct($unIdActivite, $unLibelleActivite)

{

$this->idActivite = $unIdActivite;

$this->libelleActivite = $unLibelleActivite;

$this->sesGuides = new arrayObject();

}

public function getIdActivite()

{

return $this->idActivite;

}

public function getLibelleActivite()

{

return $this->libelleActivite;

}

public function ajouteUnGuide($unGuide)

{

$this->sesGuides->append($unGuide);

}

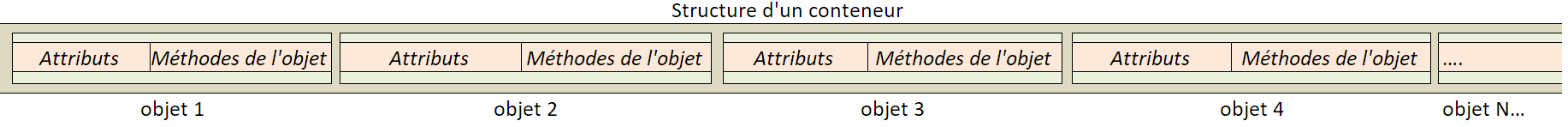
}

?>

## B – Les classes conteneurs

Les classes conteneurs ont pour objectif de permettre l’implémentation d’une collection d’objet du même type.

Si on devait faire une représentation schématique des objets conteneur on obtiendrait la représentation suivante :



Sachant que les attributs peuvent eux-mêmes être des objets, voir des collections d’objets…

Les méthodes correspondent aux besoins de traitement que l’on va avoir sur les objets.

## 1 – La classe conteneurGuide

On créer les objets de type GUIDE grâce à la classe GUIDE puis on stocke les différentes GUIDE dans un objet de type collection de GUIDE qui aura été implémenté grâce à la classe conteneurGUIDE.

<?php

include '/../Metiers/guide.php';

Class conteneurGuide

{

private $lesGuides;

public function \_\_construct()

public function ajouteUnGuide($unIdGuide,$unNomGuide)

public function nbGuides()

public function donneObjetGuideDepuisNumero($unIdGuide)

public function listeLesGuides()

public function lesGuidesAuFormatHTML()

public function donneToutesLeslanguesDUnGuide($unNumeroGuide)

public function ajouterUneLangueAuGuide($unNumeroGuide,$uneLangue)

}

?>

**Remarque** : le corps des fonctions n’est pas ici repris pour des raisons de visibilités.

## 2 – La classe conteneurPays

On créer les objets de type PAYS grâce à la classe PAYS puis on stocke les différents PAYS dans un objet de type collection de PAYS qui aura été implémenté grâce à la classe conteneurPAYS.

<?php

include '/../Metiers/pays.php';

Class conteneurPays

{

private $lesPays;

public function \_\_construct()

public function ajouteUnPays($unIdPays, $unNomPays, $unePhotoPays, $uneLanguePays)

public function nbPays()

public function listeDesPays()

public function lesPaysAuFormatHTML()

}

?>

## 3 – La classe conteneurLangue

On créer les objets de type LANGUE grâce à la classe LANGUE puis on stocke les différentes LANGUES dans un objet de type collection de LANGUE qui aura été implémenté grâce à la classe conteneurLANGUE.

<?php

include '/../Metiers/langue.php';

Class conteneurLangue

{

private $lesLangues;

public function \_\_construct()

public function ajouteUneLangue($unIdLangue,$unLibelleLangue)

public function nbLangues()

public function donneObjetLangueDepuisNumero($unIdLangue)

public function listeLesLangues()

public function lesLanguesAuFormatHTML()

}

?>

## 4 – La classe conteneurActivite

On créer les objets de type ACTIVITE grâce à la classe ACTIVITE puis on stocke les différents ACTIVITES dans un objet de type collection de ACTIVITES qui aura été implémenté grâce à la classe conteneurACTIVITE.

<?php

include '/../Metiers/activite.php';

Class conteneurActivite

{

private $lesActivites;

public function \_\_construct()

public function ajouteUneActivite($unIdActivite,$unNomActivite)

public function nbActivites()

public function donneObjetActiviteDepuisNumero($unIdActivite)

public function listeLesActivites()

public function lesActivitesAuFormatHTML()

}

?>

## C – la classe gestionVoyage

Comme son nom l’indique cette classe à pour objectif de faire la gestion de notre agence de voyage. C’est elle qui va faire le lien entre le contrôleur et les différents classes conteneurs.

Elle contient l’ensemble des conteneurs, ainsi que le lien vers la connexion à la base de données.

<?php

include 'Conteneurs/conteneurGuide.php';

include 'Conteneurs/conteneurPays.php';

include 'Conteneurs/conteneurActivite.php';

include 'Conteneurs/conteneurLangue.php';

include 'accesBD.php';

Class gestionVoyage

{

private $toutesLesActivites;

private $toutesLesLangues;

private $TousLesPays;

private $tousLesGuides;

private $maBD;

public function \_\_construct()

private function chargeLesActivites()

private function chargeLesGuides()

private function chargeLesPays()

public function ajouteUneLangue($unLibelleLangue)

public function ajouteUneActivite($unLibelleActivite)

public function ajouteUnGuide($unNomGuide)

public function ajouteUnPays($unNomPays, $unePhotoPays, $unNumeroLangue)

public function donneNbGuides()

public function donneNbPays()

public function donneNbActivites()

public function donneNbLangues()

public function listeLesGuides()

public function listeLesLangues()

public function listeLesActivites()

public function listeLesPays()

public function lesGuidesAuFormatHTML()

public function lesActivitesAuFormatHTML()

public function lesPaysAuFormatHTML()

public function lesLanguesAuFormatHTML()

public function donneDetails($unNumeroGuide)

}

?>

## C – la classe controleur

Comme son nom l’indique, elle fait le lien entre les vues et la partie traitement. C’est le contrôleur des transferts d’informations. En effet cette classe à pour objectif d’aiguiller les messages en transit entre les vues et la classe gestion qui permet d’atteindre les classes méthodes.

<?php

include 'Modeles/gestionVoyage.php';

class Controleur

{

private $monAgence;

public function \_\_construct()

public function afficheEntete()

public function affichePiedPage()

public function afficheMenu()

public function affichePage($action,$vue)

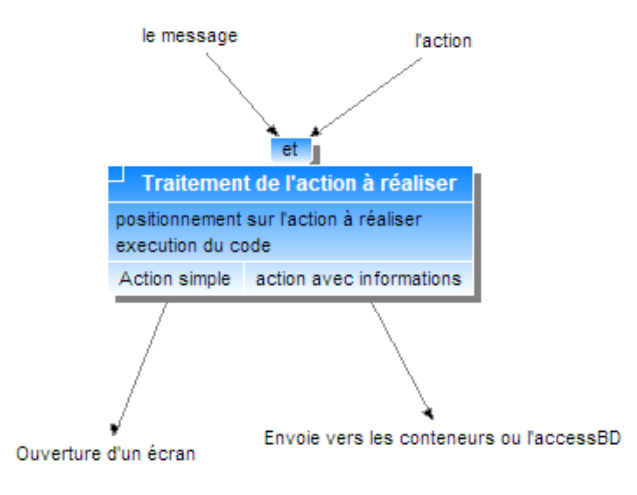
private function vuePays($action)

}

?>

Penchons nous particulièrement sur la méthode afichePage (action, vue) : elle est notre véritable aiguilleur.

Si on devait représenter schématiquement son fonctionnement on pourrait avoir le schéma suivant :



**private function vueGuide($action)**

{

switch ($action)

{

case "ajouter" :

Aller vers l’écran d’ajout d’un guide ;;

case "visualiser" :

si pas de guide dans le conteneur alors

message d’erreur

sinon

remplir $\_SESSION['lesGuides'] ;

afficher la vue voirGuide

case "enregistrer" :

récupérer le nom du guide ;;

si il n’y a pas de nom rempli alors

message d’erreur

sinon

Ajouter un guide dans l’agence

Mettre à jur le controleur

case "detail" :

remplir $\_SESSION['details']

afficher la vue détailGuide

}

}

**private function vuePays($action)**

{

switch ($action)

{

case "ajouter" :

remplir $\_SESSION['lesLangues'] pour pouvoir afficher la liste déroulante

afficher la vue ajouterPays

case "visualiser" :

si il n’y a pas de pays alors

Message d’erreur

Sinon

Remplir $\_SESSION['lesPays']

Afficher la vue voirPays

Fsi

case "enregistrer" :

récupérer les informations sur le nouveau guide

Si une des tris informations n’est pas remplie alors

Message d’erreur

Sinon

Créer le nouveau pays (lobjet)

Afficher le message d’enregistrement

Mettre à jour mon controleur

}

}

## D - Le fichier index

C’est lui qui permet le lancement de l’application, son rôle est de permettre l’initialisation des conteneurs, de la connexion et de faire le lien entre le contrôleur et les vues.

<?php

session\_start();

include 'Controleur.php';

if (!isset ($\_SESSION['Controleur']))

{

$monControleur = new Controleur();

$\_SESSION['Controleur'] = serialize($monControleur);

}

else

{

$monControleur = unserialize($\_SESSION['Controleur']);

}

$monControleur->afficheEntete();

$monControleur->afficheMenu();

if ((isset($\_GET['vue']))&& (isset($\_GET['action'])))

$monControleur->affichePage($\_GET['action'],$\_GET['vue']);

$monControleur->affichePiedPage();

?>

## D – la classe accessBD

Cette classe à pour objectif d’être l’interface entre le SGBD MySQL et l’application. C’est la seule qui est autorisée à effectuer des actions sur la base de données.

L’objet est implémenté dans gestionVoyage.

<?php

class accesBD

{

private $hote;

private $login;

private $passwd;

private $base;

public function \_\_construct()

private function connexion()

public function chargement($uneTable)

public function insertGuide($unNomGuide)

public function insertLangue($unLibelleLangue)

public function insertActivite($unLibelleActivite)

public function insertPays($unNomPays, $unePhotoPays, $uneLangue)

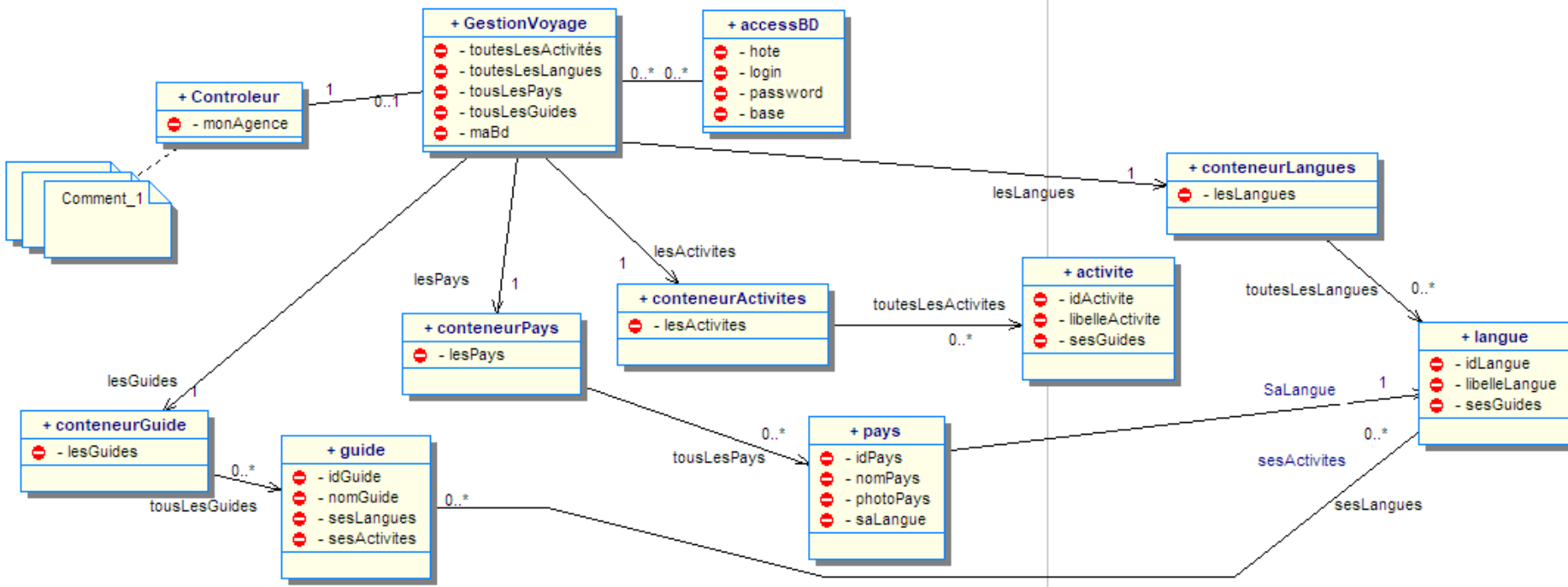
private function execRequete($uneRequete)

private function donneProchainIdentifiant($uneTable,$unIdentifiant)

}

?>

**Annexe 1 : Schéma d’architecture applicative (diagramme de classes fonctionnelles)**



Pour des raisons de visibilité l’ensemble des méthodes ne sont pas ici affichées.