DOCUMENTATION TECHNIQUE D’ARCHITECTURE

AP3

Mairie d’Ausonne

**I – La base de données**

L’application repose sur une base de données MySQL. Le type des tables est InnoDB. Le script de la base de données ce nomme script Aussonne. On le trouve dans le répertoire Outil de l’application.

***Actuellement le modèle relationnel est le suivant*** :

**ENTRAINEUR (idEntaineur, nomEntraineur, prenomEntraineur,login,pwd)**

**Clé primaire : idEntraineur**

Contient tous les entraineurs actuels des différents clubs.

**VACATAIRE (idEntaineur, telephone)**

**Clé primaire : idEntraineur**

**Clé étrangère : idEntraineur en provenance d’entraineur(idEntraineur)**

Ce sont les entraineurs qui ont un contrat CDD pour l’année scolaire (de septembre à septembre). Dans leur cas on conserve le numéro de téléphone en plus des informations normales sur les entraineurs. Nous avons un héritage sur entraineur

**TITULAIRE (idEntaineur, dateEmbauche)**

**Clé primaire : idEntraineur**

**Clé étrangère : idEntraineur en provenance d’entraineur(idEntraineur)**

Ce sont les titulaires, c’est-à-dire les entraineurs qui ont un contrat de type CDI. On conserve en plus de leur coordonnées, la date à laquelle ils ont été embauchés. Nous avons un héritage sur entraineur.

**EQUIPE(idEquipe, nomEquipe, nbrPlaceEquipe, ageMinEquipe, ageMaxEquipe, sexeEquipe, idEntraineur)**

**Clé primaire : idEquipe**

**Clé étrangère : idEntraineur en provenance d’entraineur(idEntraineur)**

Contient toutes les équipes qui sont gérées par la commune. En règle générale nous avons comme nomEquipe le type de sport pratiqué. Par exemple : « foot », cependant, si plusieurs équipes de foot venaient à se mettre en place, on pourrait imaginer d’avoir comme nomEquipe « foot junior » et « foot senior » par exemple. En effet, pour l’instant nous ne gérons pas la notion de club.

Une équipe ne possède d’un seul entraineur. S’il vient à changer, nous ne conservons pas le nom de l’ancien entraineur, d’où la clé étrangère.

**ADHERENT(idAdherent, nomAdherent, prenomAdherent, ageAdherent, sexeAdherent,login,pwd idEquipe)**

**Clé primaire : idAdherent**

**Clé étrangère : idEquipe en provenance de Equipe(idEquipe)**

Contient tous les sportifs. Actuellement un sportif ne peut adhérer qu’à une seule équipe, d’où la clé étrangère.

**ADMINISTRATEUR ( idAdmin, nomAdmin, prenomAdmin, loginAdmin, pwdAdmin)**

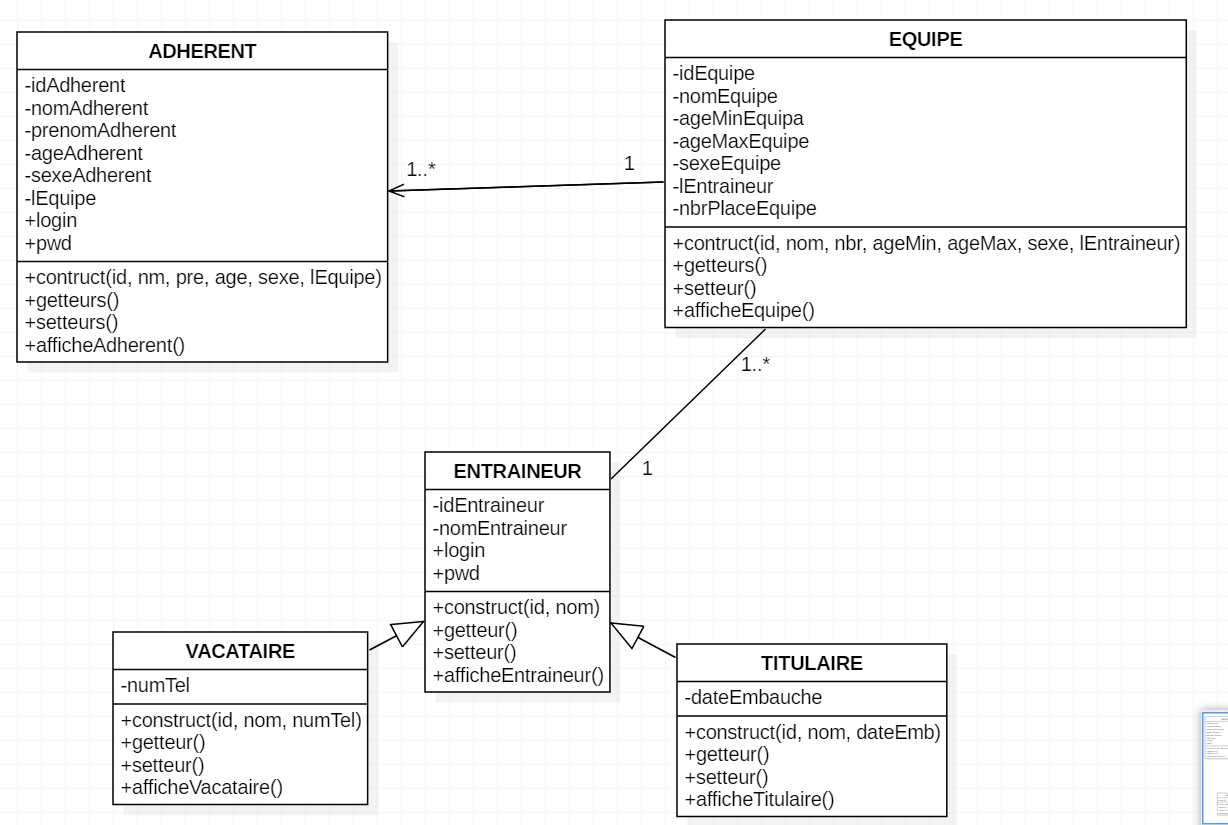
**Clé primaire : idAdmin**

Contient tous les administrateurs de la base de données et par la même des différentes applications que nous allons gérer. Par exemple le maire de la commune, la secrétaire charger de la gestion des sports.

Il comporte un jeu d’essai minimaliste.

**II – L’application**

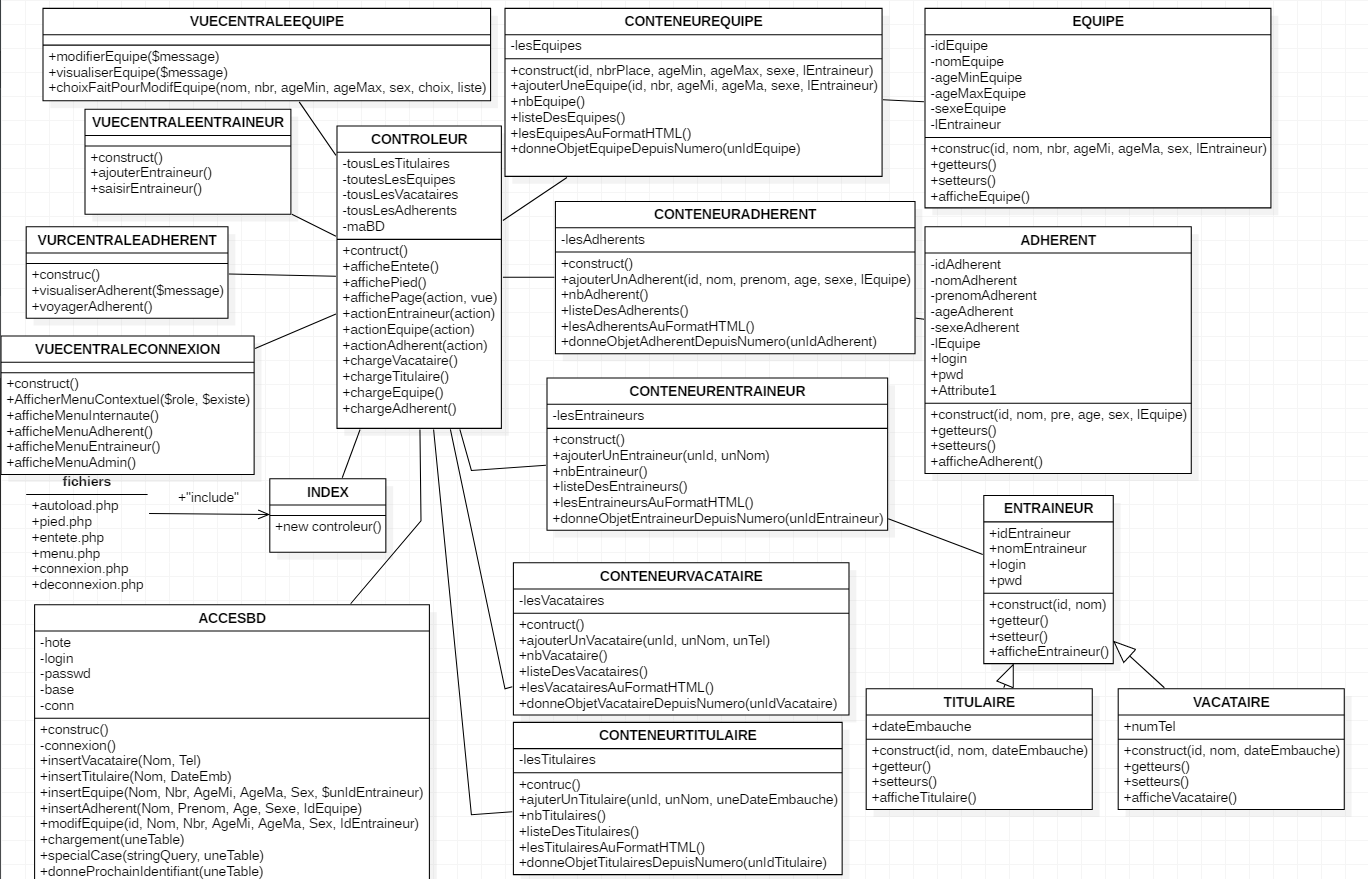
**A – Le diagramme des classes métiers**



***Attention :***

La table ADMINSTRATEUR n’ayant de lien avec aucun élément, elle n’a aucun besoin d’être représentée ici.

**B - L’architecture applicative**



Tous les attributs des classes sont en private : on ne peut donc pas faire appel à eu en dehors de leur classe sauf à partir des méthodes publiques.

**III – Descriptions des différentes classes**

Le principe de base est la séparation des différentes couches applicatives. Chacun des éléments à son objectif :

* **Les classes métiers** : permettre une cohérence métier de l’application, c’est-à-dire représenter les différents éléments nécessaires au fonctionnement du coeur de métier de l’application.
* **Les classes conteneurs** : elles permettent le stockage et l’utilisation des différents objets métiers. Elles permettent de ne pas faire appel à la base de données dès qu’on souhaite faire une manipulation sur les éléments. Il existe deux courants de pensée concernant le chargement des conteneurs :
  + Soit on les charges en début d’application, puis toutes les modifications seront effectuées sur les conteneurs et la base de données sera mise à jour à la fermeture de l’application. Cette méthode permet un travail en mode déconnecté, cependant il est dangereux en termes de sauvegarde et de mise à jour des données ;
  + Soit on les charge en début d’application, on les utilise pour toutes les manipulations, cependant à chaque fois qu’on fait une action (modif, ajout ou suppression) on répercute la mise à jour tout de suite dans la base de données. Cette méthode permet une meilleure réactivité concernant les mises à jour tout en optimisant les accès à la base de données (pas besoin dans le cas des select, ce qui, dans le type de site que nous avons à faire représente la majorité des accès.

Nous allons choisir ici la méthode 2 car elle permet une optimisation des accès au site central (uniquement lors des update, insert et delete) tout en optimisant les actions locales.

* **La classe contrôleur** : classe essentielle à l’aiguillage des différentes actions. Elle va permettre le choix des actions. De manière à avoir une meilleure lisibilité on va avoir 3 parties : entraineur, equipe et adherent.
  + Il peut arriver que le contrôleur soit tellement grand que nous ayons besoin de visibilité le concernant. Dans ce cas on le partage en fonction des fonctionnalités. Par exemple, si cela était nécessaire, nous aurions ici : un controleurEntraineur, uncontroleurAdherent et un controleurEquipe.

On effectue ce type de structure uniquement lorsque le nombre de ligne des différentes parties dépasse 800 environ. Attention ce chiffre n’est pas une règle d’or. En effet, on peut aussi décider de le faire si le nombre de méthodes outils des différentes parties est importants et rend le code illisible ou bien si on pense des le départ que le fichier va devenir ingérable…

Cependant, cela n’est pas notre cas, en effet notre projet ne génère actuellement que peu de ligne et un de méthodes outils donc pas besoin de faire cette séparation.

* **Les classes vueCentrale**… : elles contiennent le code html permettant d’interaction avec les utilisateurs. Elles ne manipulent pas d’objets juste des chaines. Il est impératif de ne pas les mettre en contact avec les objets car sinon la séparation des couches ne seraient plus effectives. Elles sont appelées directement par le contrôleur. Et ne communiquent leurs résultats qu’avec lui. Elles s’imbriquent dans le page index.php et représentent donc la partie centrale de la page.
  + **La classe vueCentraleAdherent :** elle gère l’ensemble des interactions avec l’utilisateur concernant les Adhérents.
  + **La classe vueCentraleEntraineur :** elle gère l’ensemble des interactions avec l’utilisateur concernant les Entraineurs.
  + **La classe vueCentraleEquipe :** elle gère l’ensemble des interactions avec l’utilisateur concernant les Equipes.
  + **La classe vueCentraleConnexion :** elle gère l’ensemble des différents menus contextuels. En effet les utilisateurs n’ont pas accès aux mêmes fonctionnalités en fonction de leur rôle dans l’application (admin, adherent, entraineur, internaute…). Nous aurions pu l’appeler vueCentraleMenu.
* **Les différents fichier.php indépendants** : ce que nous appelons ici des fichiers .php indépendants sont les fichiers qui ne contiennent pas de classe. Nous faisons appel à eux par l’intermédiaire qu’un require.
  + *fichier index* : C’est le cœur de notre application. Il fait appel aux fichiers autolaod.php, la création du contrôleur et l’appel au menu.php de départ d’application.

Le mode de fonctionnement est le suivant :

**Début**

Executer le fichier autoload.php qui contient les différents includes de notre application.

**SI** le controleur n’existe pas **ALORS**

Créer le controleur

Affcher l’entete.php qui contient la bannière et les différents liens css et js

**FSI**

**SI** les variables $vue et $actions sont remplies **ALORS**

// cela veut dire que l’utilisateur à cliquer sur un des menus de notre application

Partir dans le controleur executer la partie qui correspond aux variables $vue et $action.

**SINON**

// cela veut dire que nous sommes juste à l’ouverture de notre application

Afficher le fichier menu.php qui correspond au menu de base des internautes.

**FSI**

Afficher le pied.php qui contient le copyright de notre application

**Fin**

* + *fichier entete.php* : il contient les différents liens dont nous avons besoin (css, js…), ainsi que la bannière de notre site. Il est appelé dans index.php
  + *fichier menu.php* : il contient les items de menu auquel l’internaute de base peut accéder. A l’intérieur il y a les liens vers deux modales particulièrement importante (connexion.php et deconnexion.php. C’est deux fichiers vont nous permettre pour le premier de saisir les informations concernant les utilisateurs et pour le deuxième de nous déconnecter.
  + Connexion.php : contient le formulaire de saisi des informations concernant l’utilisateur qui veut se connecter (login, password et role) ce formulaire nous renvoie vers le controleur.
  + Deconnexion.php : contient l’ordre de déconnexion de notre utilisateur.

**IV – Chemin lors de l’ouverture de l’application**

