|  |
| --- |
|  |
| Rapport de technologies du web |
|  |
|  |
| **Lesur Antoine – Merlin Olivier – Jacquinot Angéline – Gillet Annabelle** |
|  |

|  |
| --- |
|  |

Contents

[Introduction 2](#_Toc451016444)

[La base de données 2](#_Toc451016445)

[Sa structure 2](#_Toc451016446)

[Sa gestion dans l’application web 2](#_Toc451016447)

[Les entités 3](#_Toc451016448)

[Annexes 4](#_Toc451016449)

[Annexe 1 : schema de la base de données 4](#_Toc451016450)

[Annexe 2 : exemple de classe mère abstraite d’un manager 4](#_Toc451016451)

[Annexe 3 : la classe Entity 5](#_Toc451016452)

[Annexe 4 : le trait Hydrator 5](#_Toc451016453)

# Introduction

# La base de données

## Sa structure

La base de données a été pensée pour s’adapter au mieux aux besoins de l’application. Lors de son développement, elle a constamment évolué pour arriver à sa forme actuelle (voir [Annexe 1 : schema de la base de données](#_Annexe_1_:)). Elle possède plusieurs tables permettant de définir les différentes entités :

* La table personne : permet de modéliser une personne, et de retrouver toutes les informations la concernant. Son champ id auto-incrémenté permet son identification.
* La table licence : est liée à la table personne (son champ id\_personne permet de faire ce lien). Elle est identifiée par son numéro (champ num) unique, et son champ activated permet de savoir si la personne correspondant s’est déjà inscrite sur le site ou non.
* La table user : symbolise un utilisateur du site. On retrouve son nom d’utilisateur (username), son mot de passe crypté depuis l’application (password) ainsi que son (ses) rôle(s) sur le site (administrateur, secrétaire, entraîneur et/ou compétiteur).
* La table adherent : représente les compétiteurs du club. On retrouve donc les informations comme leur catégorie ou leur spécialité, ainsi que la validité de leur certificat médical. Comme c’est une personne, on peut lier les deux tables grâce au champ num\_personne.
* La table competition : contient tous les renseignements sur une compétition. Elle est liée à plusieurs tables, permettant ainsi de gérer les inscriptions à une compétition et au transport, ou la participation des bénévoles.
* La table equipage : modélise un équipage inscrit à une compétition (son champ id\_competition permet d’identifier cette compétition). Cette table est également liée à plusieurs autres tables, afin de gérer les compétiteurs prenant part à cet équipage.
* La table adherent\_transport : sert de lien entre une compétition et un adhérent, et l’identifie comme prenant part au transport pour cette compétition.
* La table adherent\_equipage : place un adhérent dans un équipage. Le champ valide permet de savoir si un entraîneur doit valider l’inscription ou non.
* La table adherent\_equipage\_invite : lie un compétiteur à un équipage auquel il a été invité à participer.
* La table accompagnateur\_benevole : permet d’enregistrer un non licencié comme bénévole d’une compétition et de connaître son rôle. Il est lié à une personne grâce au champ id\_personne.
* La table accompagnateur\_officiel : fonctionne de la même manière que la table accompagnateur\_benevole, sauf qu’elle est liée à une licence puisque l’accompagnateur doit faire parti du club.

## Sa gestion dans l’application web

Plusieurs éléments permettent de gérer la base de données dans l’application. Tout d’abord, le fichier PDOFactory.php permet de définir les paramètres de connections à la base de données.

Ensuite, les fichiers Managers.php et Manager.php permettent de gérer les différents managers utilisés pour récupérer les informations souhaitées. Les managers utilisés sont constitués de deux fichiers chacun : un représentant la classe mère abstraite (voir [Annexe 2 : exemple de classe mère abstraite d’un manager](#_Annexe_2_:)), et l’autre la classe fille contenant les fonctions utilisant PDO. De cette manière, si la base de données a besoin d’être gérer à l’aide d’un autre outil, créer une autre classe fille suffit et permet de conserver la compatibilité. Ils font le lien entre la base de données et les entités.

On retrouve toujours les fonctions add, modify (gérées par save, sui détermine si l’entité est nouvelle ou non) et delete, qui permettent respectivement d’ajouter, de modifier et de supprimer une entrée dans la base de données. Bien souvent, nous avons une fonction getUnique, permettant de récupérer l’entité en fonction de son id unique. Le reste des fonctions dépend des besoins de l’application web envers chaque table.

Nous pouvons utiliser des managers pour gérer les bénévoles officiels et non licenciés, les compétiteurs, les compétitions, les équipages, les licences, les personnes et les utilisateurs.

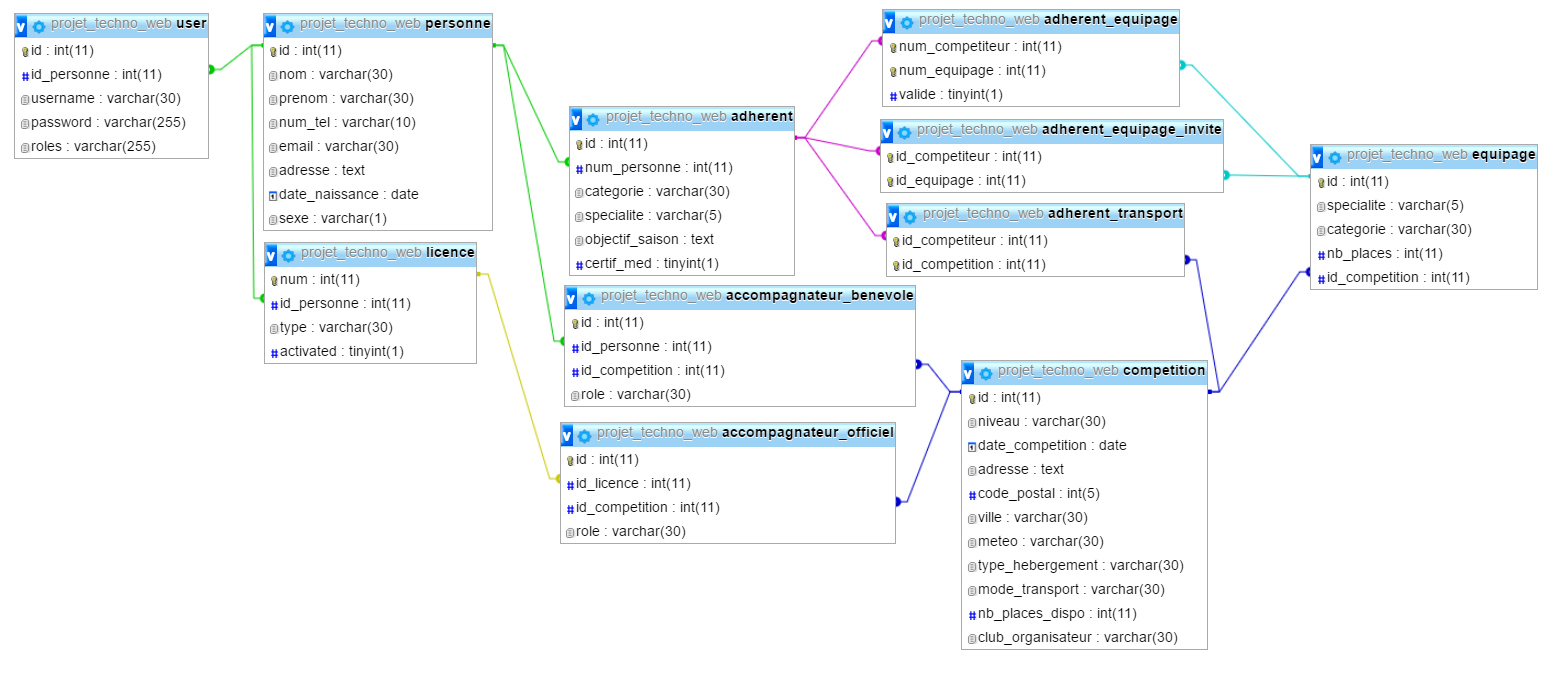
# Les entités

Les entités sont des classes représentant un objet utile à l’application. C’est dans ces objets que les managers manipulent les informations. Elles sont toutes des classes filles de la classe Entity (voir [Annexe 3 : la classe Entity](#_Annexe_3_:)), qui permet de définir l’attribut id (présent dans toutes les entités) et permet d’hydrater la classe grâce au trait Hydrator (voir [Annexe 4 : le trait Hydrator](#_Annexe_4_:)). De cette manière, le constructeur n’a pas besoin d’être redéfini à chaque fois, les setters suffisent à l’initialisation de la classe.

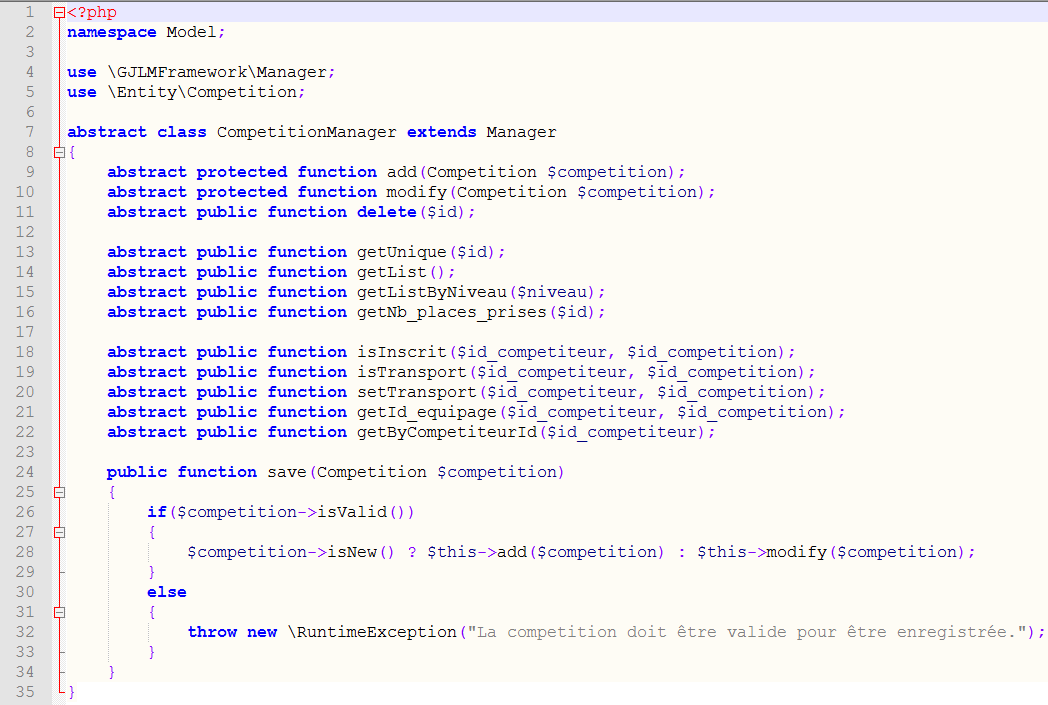
De la même manière que les managers, les entités disponibles sont Benevole, BenevoleOfficiel, Competiteur, Competition, Equipage, Licence, Personne et User.

# Annexes

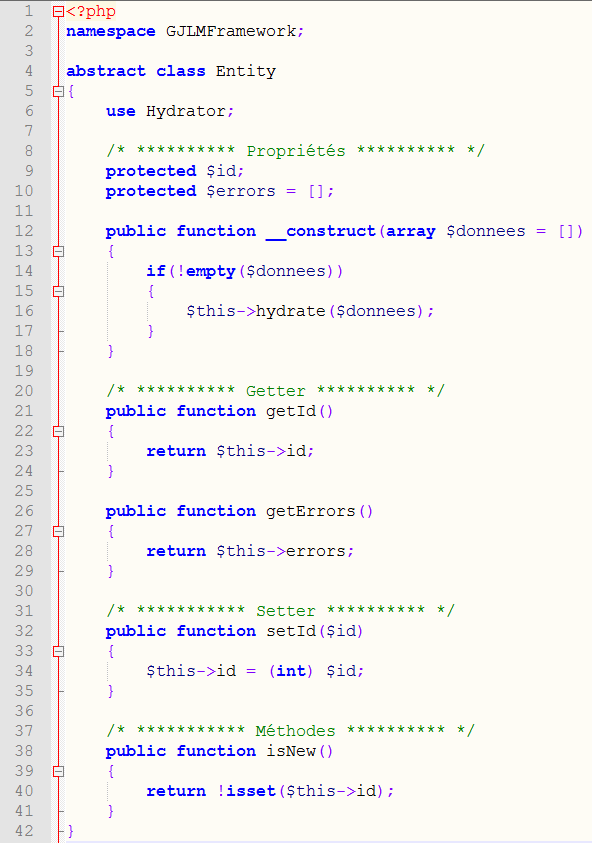
## Annexe 1 : schema de la base de données



## Annexe 2 : exemple de classe mère abstraite d’un manager



## Annexe 3 : la classe Entity



## Annexe 4 : le trait Hydrator

