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

|  |
| --- |
|  |

Sommaire

[Introduction 5](#_Toc452053996)

[Répartition des tâches 5](#_Toc452053997)

[Fonctionnement du CMS 6](#_Toc452053998)

[Introduction 6](#_Toc452053999)

[Particularités nécessaires au fonctionnement 6](#_Toc452054000)

[Architecture du projet 6](#_Toc452054001)

[Le dossier App 6](#_Toc452054002)

[Le dossier Errors 6](#_Toc452054003)

[Le dossier Lib 6](#_Toc452054004)

[Le dossier Web 6](#_Toc452054005)

[Point d’entrée de l’application 6](#_Toc452054006)

[Le fichier Web/bootstrap.php 6](#_Toc452054007)

[Le fichier Web/.htaccess 7](#_Toc452054008)

[La classe Application 7](#_Toc452054009)

[La classe ApplicationComponent 7](#_Toc452054010)

[Les composants de base du CMS 7](#_Toc452054011)

[La classe Auth 7](#_Toc452054012)

[La classe BaseController 7](#_Toc452054013)

[La classe Config 8](#_Toc452054014)

[La classe HTTPRequest 8](#_Toc452054015)

[La classe HTTPResponse 8](#_Toc452054016)

[La classe Page 8](#_Toc452054017)

[La classe Roles 8](#_Toc452054018)

[Les composants complémentaires du CMS 8](#_Toc452054019)

[La gestion des routes 8](#_Toc452054020)

[La classe Route 9](#_Toc452054021)

[La classe Router 9](#_Toc452054022)

[La gestion de la base de données 9](#_Toc452054023)

[La classe Entity 9](#_Toc452054024)

[La classe Manager 9](#_Toc452054025)

[La classe Managers 9](#_Toc452054026)

[La classe PDOFactory 10](#_Toc452054027)

[La gestion des sessions 10](#_Toc452054028)

[La classe User 10](#_Toc452054029)

[La gestion des formulaires 10](#_Toc452054030)

[La classe Validator 10](#_Toc452054031)

[La classe Field 10](#_Toc452054032)

[La classe Form 10](#_Toc452054033)

[La classe FormBuilder 11](#_Toc452054034)

[Fonctionnement de l’application 12](#_Toc452054035)

[La base de données 12](#_Toc452054036)

[Sa structure 12](#_Toc452054037)

[Sa gestion dans l’application web 12](#_Toc452054038)

[Les entités 13](#_Toc452054039)

[Les formulaires 13](#_Toc452054040)

[Fonctionnalités de l’application 14](#_Toc452054041)

[L’administration 14](#_Toc452054042)

[L’ajout de personne 14](#_Toc452054043)

[L’ajout de licence 14](#_Toc452054044)

[La gestion des rôles 14](#_Toc452054045)

[La gestion des catégories 14](#_Toc452054046)

[La gestion des certificats médicaux 14](#_Toc452054047)

[Valider les inscriptions 15](#_Toc452054048)

[La connexion et l’inscription 15](#_Toc452054049)

[L’inscription 15](#_Toc452054050)

[Connexion 15](#_Toc452054051)

[Les compétitions 15](#_Toc452054052)

[Ajouter une compétition 15](#_Toc452054053)

[La liste des compétitions 16](#_Toc452054054)

[Vue d’une compétition 16](#_Toc452054055)

[Les équipages 16](#_Toc452054056)

[Ajouter un équipage 16](#_Toc452054057)

[Voir un équipage 16](#_Toc452054058)

[La gestion des invités d’un équipage 17](#_Toc452054059)

[La gestion des participants d’un équipage 17](#_Toc452054060)

[Le profil 17](#_Toc452054061)

[La vue du profil 17](#_Toc452054062)

[La modification du profil 17](#_Toc452054063)

[La liste des compétitions 17](#_Toc452054064)

[Les invitations 18](#_Toc452054065)

[Les bénévoles 18](#_Toc452054066)

[Inscription d’un licencié comme bénévole 18](#_Toc452054067)

[Inscription d’un non-licencié comme bénévole 18](#_Toc452054068)

[La liste des bénévoles 18](#_Toc452054069)

[Conclusion 18](#_Toc452054070)

[Annexes 19](#_Toc452054071)

[Structure du dossier App 19](#_Toc452054072)

[Le fichier bootstrap.php 19](#_Toc452054073)

[Le fichier .htaccess 20](#_Toc452054074)

[Diagramme de classe : Application 20](#_Toc452054075)

[Diagramme de classe : ApplicationComponent 20](#_Toc452054076)

[Diagramme de classe : Auth 21](#_Toc452054077)

[Diagramme de classe : BaseController 21](#_Toc452054078)

[Diagramme de classe : Config 21](#_Toc452054079)

[Diagramme de classe : HTTPRequest 22](#_Toc452054080)

[Diagramme de classe : HTTPResponse 22](#_Toc452054081)

[Diagramme de classe : Page 22](#_Toc452054082)

[Diagramme de classe : Roles 23](#_Toc452054083)

[Diagramme de classe : Route 23](#_Toc452054084)

[Diagramme de classe : Router 24](#_Toc452054085)

[Diagramme de classe : Entity 24](#_Toc452054086)

[Diagramme de classe : Manager 24](#_Toc452054087)

[Diagramme de classe : Managers 25](#_Toc452054088)

[Diagramme de classe : PDOFactory 25](#_Toc452054089)

[Diagramme de classe : User 25](#_Toc452054090)

[Diagramme de classe : Validator 25](#_Toc452054091)

[Diagramme de classe : Field 26](#_Toc452054092)

[Diagramme de classe : Form 26](#_Toc452054093)

[Diagramme de classe : FormBuilder 26](#_Toc452054094)

[Exemple de classe dérivant FormBuilder 27](#_Toc452054095)

[Schéma de la base de données 28](#_Toc452054096)

[Exemple de classe mère abstraite d’un manager 28](#_Toc452054097)

[La classe Entity 29](#_Toc452054098)

[Le trait Hydrator 29](#_Toc452054099)

[Use case 30](#_Toc452054100)

[L’ajout de personne 30](#_Toc452054101)

[L’ajout d’adhérent 31](#_Toc452054102)

[La gestion des rôles 31](#_Toc452054103)

[La gestion des catégories 31](#_Toc452054104)

[La gestion des certificats médicaux 31](#_Toc452054105)

[Valider les inscriptions 32](#_Toc452054106)

[Les étapes de l’inscription 32](#_Toc452054107)

[La connexion 32](#_Toc452054108)

[Ajouter une compétition 33](#_Toc452054109)

[Liste des compétitions 33](#_Toc452054110)

[Vue d’une compétition 34](#_Toc452054111)

[Ajouter un équipage 34](#_Toc452054112)

[Voir un équipage 35](#_Toc452054113)

[Gestion des invités d’un équipage 35](#_Toc452054114)

[Gestion des participants d’un équipage 36](#_Toc452054115)

[La vue du profil 36](#_Toc452054116)

[La modification du profil 37](#_Toc452054117)

[La liste des compétitions d’un compétiteur 37](#_Toc452054118)

[Les invitations d’un compétiteur 37](#_Toc452054119)

[Inscription d’un licencié comme bénévole 38](#_Toc452054120)

[Inscription d’un non-licencié comme bénévole 38](#_Toc452054121)

[Liste des bénévoles 38](#_Toc452054122)

# Introduction

Nous avions pour projet de réaliser un site web pour une association de Canoë-Kayak. Pour cela nous avons développé un CMS (Content Management System) sous la forme d’un Framework PHP, et le site web en question sous la forme d’une application utilisant ce Framework.

Tout d’abord nous allons vous expliquer le fonctionnement du CMS, puis le fonctionnement de l’application et les fonctionnalités de cette dernière.

En complément, le code complet de l’application peut être trouvé à l’adresse <https://github.com/adraax/ProjetTechnoWeb>.

# Répartition des tâches

Pour développer cette application, nous nous sommes répartis les tâches de manière à ce que chacun soit responsable d’un domaine. Olivier s’est donc occupé du CMS, Angéline de la base de données ainsi que de sa liaison avec l’application (c'est-à-dire les managers), Antoine des formulaires et des entités, et Annabelle des fonctionnalités.

Fonctionnement du CMS

# Introduction

Le CMS a été développé en suivant le modèle Modèle-Vue-Contrôleur. Il utilise également le principe d’URL-Rewritting et utilise la notation PSR-0 pour charger automatiquement les classes PHP.

# Particularités nécessaires au fonctionnement

Pour le bon fonctionnement du site et de son routing, le projet doit être défini dans un virtual host (pointant sur le dossier Web/ du projet), et le module d’URL-rewritting d’Apache doit être activé.

Si ces conditions ne sont pas respectées, le site renverra en permanence une erreur 404.

# Architecture du projet

Le projet est constitué de 4 dossiers principaux : App, Errors, Lib et Web.

## Le dossier App

Ce dossier contient les applications gérées par notre CMS. À l’intérieur de ce dossier se trouvent les différents modules et leur contrôleur et vues correspondants, ainsi que les fichiers de configuration et le template de l’application (voir [Structure du dossier App](#_Structure_du_dossier)).

## Le dossier Errors

Ce dossier contient uniquement les vues des pages d’erreur, comme la page d’erreur 404 par exemple.

## Le dossier Lib

Ce dossier est composé de 2 sous-dossiers : le dossier GJLMFramework contenant le code du CMS, et le dossier Vendors contenant les modèles, les entités et les formulaires des applications.

## Le dossier Web

Ce dossier contient tous les fichiers publics du site : les images, le css, le javascript … Il contient également les fichiers .htaccess et bootstrap.php.

# Point d’entrée de l’application

Deux fichiers sont utilisés conjointement pour initialiser l’application : le fichier Web/.htaccess et le fichier Web/bootstrap.php.

## Le fichier Web/bootstrap.php

Ce fichier permet d’enregistrer les fonctions de chargement automatique des classes et de lancer l’application correspondant à la route demandée par l’utilisateur (voir [Le fichier bootstrap.php](#_Le_fichier_bootstrap.php)).

Ce fichier utilise la classe SplClassLoader (<https://gist.github.com/jwage/221634> ), afin de charger automatiquement les classes PHP. Cette classe permet d’associer un namespace avec un dossier, à condition que la notation PSR-0 soit utilisée pour le nommage des classes et pour l’organisation physique des dossiers du projet.

C’est le cas de notre CMS.

## Le fichier Web/.htaccess

Ce fichier de configuration Apache permet d’utiliser le module d’Url-rewriting pour la requête et de répartir les requêtes sur les bonnes applications. En effet plusieurs applications peuvent être gérées simultanément en précisant un préfixe précédant les routes de chaque application.

Il permet de réécrire l’URL en rajoutant bootstrap.php dans cette URL, tout en conservant l’URI de base.

(Voir [Le fichier .htaccess](#_Le_fichier_.htaccess)).

# La classe Application

(Voir [Diagramme de classe : Application](#_Diagramme_de_classe)).

La classe Application représente l’application dans son ensemble. Chaque Application a un nom ce qui permet d’en gérer plusieurs avec le même framework.

L’application est composée d’une requête et d’une réponse HTTP, d’une configuration et d’un utilisateur (correspondant à l’internaute, caractérisé par une session et pas forcément à un membre de l’association connecté à l’application).

Cette classe permet de récupérer le contrôleur correspondant à la requête et définit le fonctionnement de l’application (grâce à la méthode *run()* ).

# La classe ApplicationComponent

(Voir [Diagramme de classe : ApplicationComponent](#_Diagramme_de_classe_1)).

La classe ApplicationComponent représente les composants de l’application. Ainsi tous les composants héritent de cette classe et ont une référence vers l’application à laquelle ils appartiennent.

# Les composants de base du CMS

Les composants du CMS sont toutes les classes dérivant de la classe ApplicationComponent et ont une fonction plus ou moins importante dans le fonctionnement de l’application.

## La classe Auth

(Voir [Diagramme de classe : Auth](#_Diagramme_de_classe_2)).

Cette classe permet de filtrer les accès aux différentes pages du site. En effet cette classe renvoie un booléen permettant de savoir si une authentification est nécessaire pour accéder à la page

## La classe BaseController

(Voir [Diagramme de classe : BaseController](#_Diagramme_de_classe_3)).

Tous les contrôleurs des différentes applications héritent de cette classe. Les contrôleurs choisissent la fonction à exécuter en fonction de la requête et ils effectuent le lien entre le modèle et la vue pour renvoyer la page à l’internaute.

Ils possèdent un module et une action, qui leur permet de trouver le fichier contenant la vue correspondant à la requête. Ils possèdent également la page à renvoyer à l’utilisateur ainsi que des managers permettant d’accéder à la base de données.

## La classe Config

(Voir [Diagramme de classe : Config](#_Diagramme_de_classe_4)).

Cette classe permet d’obtenir tous les paramètres de l’application. La configuration est stockée dans le dossier Config de chaque application.

Le type de paramètres est varié : ils peuvent correspondre à la taille maximale d’un nom d’utilisateur comme aux éléments d’identification à la base de données.

## La classe HTTPRequest

(Voir [Diagramme de classe : HTTPRequest](#_Diagramme_de_classe_5)).

Cette classe représente la requête reçue par l’application. Elle permet de tester l’existence et de récupérer les valeurs des cookies, des variables $\_GET et des variables $\_POST. Elle permet également de récupérer la méthode de la requête (GET ou POST) ainsi que son URI.

## La classe HTTPResponse

(Voir [Diagramme de classe : HTTPResponse](#_Diagramme_de_classe_6)).

Cette classe représente la réponse http envoyée à l’utilisateur à la suite de sa requête. Elle permet de créer des cookies, d’ajouter des en-têtes, de faire des redirections et d’envoyer la page à l’utilisateur.

C’est cette classe qui permet d’arrêter le fonctionnement du CMS, afin d’envoyer la réponse.

## La classe Page

(Voir [Diagramme de classe : Page](#_Diagramme_de_classe_7) ).

Cette classe représente la page qui sera affichée à l’utilisateur. Elle permet d’inclure la vue correspondant à la requête dans le template de base du site.

Pour cela, nous utilisons des buffers grâce aux fonctions *ob\_start()* et *ob\_get\_clean()* pour extraire les variables passées à la vue et les afficher.

(Voir <http://php.net/manual/fr/function.ob-start.php> , <http://php.net/manual/fr/function.ob-get-clean.php> ).

## La classe Roles

(Voir [Diagramme de classe : Roles](#_Diagramme_de_classe_8) ).

Tout comme la classe Auth, la classe Roles permet de filtrer l’accès aux différentes pages du site. En effet, cette classe permet de savoir si l’utilisateur connecté a le rôle nécessaire pour accéder à la page grâce à la fonction *hasRoles()*.

# Les composants complémentaires du CMS

En dehors des composants de base du CMS, il existe également des composants complémentaires qui sont indépendants du CMS, mais indispensables à son bon fonctionnement.

## La gestion des routes

Pour gérer les routes, nous avons créé un système de routing se basant sur les expressions régulières.

### La classe Route

(Voir [Diagramme de classe : Route](#_Diagramme_de_classe_9) ).

La classe Route représente les différentes routes de l’application. Elle est la modélisation objet du fichier de configuration *routes.xml*. Elle permet de récupérer toutes ses informations comme le module auquel elle correspond, l’action du contrôleur à exécuter, ou son url.

Elle permet également de vérifier si l’url de la requête lui correspond. Si c’est le cas, elle permet de récupérer les paramètres de la route s’ils existent.

Le système de routes paramétrées n’est pas utilisé dans notre application, mais nous l’avons quand même implémenté dans l’optique d’un framework PHP complet et réutilisable.

### La classe Router

(Voir [Diagramme de classe : Router](#_Diagramme_de_classe_10) ).

La classe Router correspond au routeur en lui-même. Il contient toutes les routes qu’il doit gérer dans un tableau.

C’est cette classe qui permet de récupérer la route correspondant à la requête de l’utilisateur. Si aucune route correspondant à la requête n’est trouvée, cette classe génère une *RuntimeException*, ensuite capturée par la classe Application pour générer une erreur 404.

## La gestion de la base de données

Afin de gérer et de structurer les accès à la base de données, nous avons créé un système basé sur des entités et des managers.

### La classe Entity

(Voir [Diagramme de classe : Entity](#_Diagramme_de_classe_11) ).

La classe Entity est une classe abstraite qui modélise les entités présentes dans la base de données. Elle implémente une fonction permettant de savoir si l’entité est nouvelle ou déjà existante grâce à son id.

Cette classe utilise [Le trait Hydrator](#_Le_trait_Hydrator), afin de pouvoir remplir l’entité automatiquement à la suite d’une requête SQL grâce au fetch de PDO.

### La classe Manager

(Voir [Diagramme de classe : Manager](#_Diagramme_de_classe_12) ).

Cette classe est une classe abstraite qui servira de base à tous les managers des applications gérées par le CMS. Elle contient uniquement le DAO (Database Access Object), c’est-à-dire l’objet contenant la connexion à la base de données.

### La classe Managers

(Voir [Diagramme de classe : Managers](#_Diagramme_de_classe_13) ).

Cette classe sert à gérer les différents managers. Elle contient le DAO ainsi que l’API.

En effet nous avons conçu notre système de façon à pouvoir utiliser plusieurs API pour accéder à notre base de données. Ainsi chaque manager est défini pour chaque API utilisée, et on choisit quel manager nous devons utiliser grâce à un choix de l’API.

Dans notre projet nous avons uniquement utilisé l’API PDO par soucis de simplicité, mais nous aurions également pu utiliser l’API mysql ou mysqli.

### La classe PDOFactory

(Voir [La classe PDOFactory](#_Diagramme_de_classe_14) ).

Cette classe sert à créer la connexion à la base de données. Ici la classe correspond à une connexion effectuée avec l’API PDO. Si d’autres API étaient utilisées, d’autres classe de type Factory existeraient pour fournir les connexions aux bases de données.

## La gestion des sessions

La gestion des sessions se fait uniquement grâce à la classe User qui contient toutes les méthodes nécessaires pour une bonne gestion.

### La classe User

(Voir [Diagramme de classe : User](#_Diagramme_de_classe_15) ).

Cette classe gère la session. Elle permet de tester l’existence, d’ajouter et de supprimer des variables de la session.

Elle permet également de créer des messages flash, c’est-à-dire des messages qui s’afficheront sur la prochaine page affichée, puis qui sont supprimés.

## La gestion des formulaires

Afin d’automatiser la création des formulaires et de gérer facilement la validation des formulaires et la gestion des erreurs, nous avons créé plusieurs classes.

### La classe Validator

(Voir [Diagramme de classe : Validator](#_Diagramme_de_classe_16) ).

Cette classe abstraite permet de créer des règles de validations avec des messages d’erreurs personnalisés.

Nous avons créé plusieurs Validator :

* MaxLengthValidator : permet de vérifier que la taille d’une chaine de caractères ne dépasse pas une taille spécifiée
* NotNullValidator : permet de vérifier qu’un champ n’est pas vide

### La classe Field

(Voir [Diagramme de classe : Field](#_Diagramme_de_classe_17) ).

Cette classe abstraite représente un champ de formulaire. Elle contient le nom du champ, le label à afficher et sa valeur (qui permet de conserver les entrées utilisateurs en cas d’erreur).

Chaque type de champ à une classe dérivée de la classe Field : DatePickerField, PasswordField, …

### La classe Form

(Voir [Diagramme de classe : Form](#_Diagramme_de_classe_18) ).

Cette classe correspond à un formulaire dans son ensemble. Il est composé de champs (Field) et est lié à une Entity. Cela permet de conserver les entrées utilisateurs en cas d’erreur.

Elle possède également une fonction permettant de tester la validité de l’ensemble des champs ainsi qu’une fonction permettant de générer le code HTML correspondant au formulaire.

### La classe FormBuilder

(Voir [Diagramme de classe : FormBuilder](#_Diagramme_de_classe_19) ).

Cette classe abstraite est utilisée pour créer les formulaires. Elle est dérivée pour chaque formulaire des applications gérées par le CMS.

La méthode *build()* de cette classe permet de renseigner l’enchainement des champs nécessaires à la construction du formulaire.

Voir un [Exemple de classe dérivant FormBuilder](#_Exemple_de_classe) .

Fonctionnement de l’application

# 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 [Schéma de la base de données](#_Schéma_de_la)). 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 partie 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 [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 gérée 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, qui 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 [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 [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.

Ces entités sont très pratiques, grâce à elles nous pouvons représenter un objet dans son intégralité plutôt que d’avoir une multitude de variables, et tout autant d’erreurs éventuelles.

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

# Les formulaires

Comme expliqué précédemment, les formulaires peuvent être créés automatiquement grâce à la classe FormBuilder. Lorsque l’application a besoin d’utiliser un formulaire, une classe est donc créée avec les champs qui seront disponibles. Cette technique est utilisée plusieurs fois dans l’application, que ce soit pour l’ajout d’une personne, d’une licence, pour la connexion ou l’inscription… Ils utilisent également les entités, vérifiant que les informations entrées dans le formulaire permettent à l’entité correspondante d’être valide.

Un aperçu de l’aspect de ces formulaires est disponible dans plusieurs captures d’écran des annexes (par exemple : [L’ajout de personne](#_L’ajout_de_personne)).

Fonctionnalités de l’application

L’application a été faite pour respecter au maximum le cahier des charges fournis. On retrouve donc cinq rôles qui interagissent avec le site : administrateur, secrétaire, entraîneur, compétiteur et loisir, chacun ayant ses propres fonctions (voir [Use case](#_Use_case)).

# L’administration

## L’ajout de personne

Aspect de l’interface : [L’ajout de personne](#_L’ajout_de_personne).

Cette partie est accessible uniquement pour un administrateur. Elle lui permet d’ajouter une personne dans l’application. Une personne peut évoluer vers deux fonctions : soit elle devient utilisatrice du site en s’inscrivant (il faut alors lui associer sa licence avant qu’elle puisse le faire), soit elle est inscrite par le secrétaire en tant que bénévole non officiel d’une compétition.

## L’ajout de licence

Aspect de l’interface : [L’ajout d’adhérent](#_L’ajout_d’adhérent).

Une fois que la personne a été enregistrée, l’administrateur peut alors la sélectionner dans une liste déroulante, lui attribuer un type de licence (compétiteur, entraineur, secrétaire ou loisir) qui permettra de lui attribuer un rôle sur le site lors de son inscription, ainsi que son numéro de licence. Une fois que cet ajout est fait, la personne peut alors s’inscrire sur le site.

## La gestion des rôles

Aspect de l’interface : [La gestion des rôles](#_La_gestion_des).

Là aussi, seul l’administrateur a accès à cette fonctionnalité. Il peut modifier les rôles attribués à chaque utilisateur (sélectionnable dans une liste déroulant). A chaque fois qu’il change d’utilisateur sélectionné, les rôles qu’il possède sont remplis automatiquement. Ce changement s’effectue grâce à de l’ajax : en réponse au clic sur la liste déroulante, on récupère les rôles de l’utilisateur sélectionné et on les retourne sous un format XML. Ensuite, ce résultat est lu afin de cocher les cases correspondantes.

Si l’administrateur attribue un rôle de compétiteur à un utilisateur qui ne l’avait pas déjà, la table adherent correspondante est créée dans la base de données. Inversement, si le rôle compétiteur est ôté à un utilisateur, la table qui lui correspond est supprimée.

## La gestion des catégories

Aspect de l’interface : [La gestion des catégories](#_La_gestion_des_1).

Si besoin est, le secrétaire peut modifier la catégorie d’un compétiteur. Pour cela, la liste déroulante contient le nom de tous les compétiteurs, et, de la même manière que pour les rôles (en utilisant de l’ajax), en sélectionner un cochera automatiquement sa catégorie actuelle.

## La gestion des certificats médicaux

Aspect de l’interface : [La gestion des certificats médicaux](#_La_gestion_des_2).

Accessible par le secrétaire, cette fonctionnalité permet de valider le certificat médical d’un compétiteur, lui permettant ainsi de s’inscrire aux compétitions. Pour cela, le secrétaire a juste à cliquer sur le bouton « Valider le certificat ».

## Valider les inscriptions

Aspect de l’interface : [Valider les inscriptions](#_Valider_les_inscriptions).

Accessible par l’entraîneur, il peut voir la liste des mineurs qui se sont inscrits à une compétition, et valider cette inscription. À partir de cette page, il peut également voir l’équipage auquel le mineur veut participer, et valider son inscription depuis la liste des participants de cet équipage.

Un entraîneur valide l’inscription d’un mineur lorsqu’il reçoit la confirmation des parents, qu’il obtient lorsque les parents emmènent leur enfant à un entraînement ou par une autorisation signée. C’est la méthode la plus pratique pour procéder à cette vérification : l’entraîneur est sûr que l’autorisation vient des parents.

# La connexion et l’inscription

## L’inscription

Aspect de l’interface : [Les étapes de l’inscription](#_Les_étapes_de).

Pour pouvoir s’inscrire, l’utilisateur doit avoir sa licence préalablement enregistrée dans l’application par l’administrateur. Une fois cela fait, l’utilisateur peut créer son compte.

Tout d’abord, il devra entrer son numéro de licence ainsi que sa date de naissance. Si les deux correspondent à la même personne et que la licence n’a pas encore été activée, il passe à la deuxième étape, c'est-à-dire choisir un nom d’utilisateur ainsi qu’un mot de passe. Il y a également un champ pour entrer une seconde fois le mot de passe et ainsi pouvoir limiter le risque de faute de frappe. Si les deux mots de passe correspondent, un compte est créé pour cet utilisateur, avec le rôle que l’administrateur lui a attribué au moment de la création de la licence. Le mot de passe est crypté dans la base de données.

## Connexion

Aspect de l’interface : [La connexion](#_La_connexion).

C’est sur cette interface que l’utilisateur peut se connecter s’il possède déjà un compte. Il lui suffit d’entrer son identifiant et son mot de passe, et s’il y a une correspondance dans la base de données (le mot de passe est crypté avant d’être comparé à celui de la base de données), l’utilisateur sera connecté avec le compte correspondant et pourra accéder aux fonctionnalités de l’application disponibles en fonction de son (ses) rôle(s). Une fois connecté, l’utilisateur est redirigé vers la page des compétitions.

La connexion est également via une fenêtre modale qui apparaît en cliquant sur « Connexion » disponible dans la barre de menu.

# Les compétitions

## Ajouter une compétition

Aspect de l’interface : [Ajouter une compétition](#_Ajouter_une_compétition).

Pour pouvoir ajouter une compétition, l’utilisateur doit être administrateur, secrétaire ou entraîneur. Ensuite, il doit remplir le formulaire présent (les champs météo et type d’hébergement sont optionnels, la météo car elle peut ne pas être connue et l’hébergement car il peut ne pas exister). Si le formulaire est rempli correctement, un clic sur valider enregistrera la compétition.

## La liste des compétitions

Aspect de l’interface : [Liste des compétitions](#_Liste_des_compétitions).

Tous les utilisateurs peuvent accéder à la liste des compétitions, qu’ils peuvent trier par niveau. Les compétitions apparaissent dans l’ordre chronologique inverse. Les boutons de suppression ne sont disponibles que pour les administrateurs et secrétaires. Ils ont également accès au bouton d’ajout, de même que l’entraîneur. En cliquant sur le bouton « Voir cette compétition », les utilisateurs peuvent afficher une vue plus détaillée.

## Vue d’une compétition

Aspect de l’interface : [Vue d’une compétition](#_Vue_d’une_compétition).

Sur cette page, les fonctionnalités sont différentes en fonction du rôle de l’utilisateur. Tout le monde peut voir les informations de la compétition, les liens cartes et itinéraires redirigent respectivement vers une carte du lieu de la compétition et vers un itinéraire Dijon-lieu de la compétition. Tout le monde a également accès aux boutons « S’inscrire comme bénévole » et « Voir les bénévoles », qui redirigent l’utilisateur vers les pages correspondantes (voir la section « Les bénévoles »). Si l’utilisateur est administrateur ou secrétaire, il pourra également ajouter un bénévole.

Les compétiteurs ont d’autres possibilités sur cette page. S’ils ne sont pas inscrits à la compétition, ils peuvent créer un équipage si leur certificat médical est valide (ils ont un bouton « Inscrire un équipage », s’ils n’ont pas de certificat, cela leur est notifié). S’ils sont déjà inscrits, ce bouton devient « Voir l’équipage », qui redirige alors sur l’équipage du compétiteur pour cette compétition, et s’il reste des places disponibles pour le transport, ils peuvent s’y inscrire en cliquant sur le bouton « S’inscrire au transport ». Si le compétiteur s’inscrit au transport, le nombre de places disponibles est mis à jour, et le bouton devient « Annuler l’inscription au transport ». Il peut alors se désinscrire s’il le souhaite. S’il n’y a plus de place disponible, l’inscription au transport est impossible et cela est indiqué clairement.

# Les équipages

## Ajouter un équipage

Aspect de l’interface : [Ajouter un équipage](#_Ajouter_un_équipage).

Un compétiteur avec une licence valide peut ajouter un équipage pour une compétition s’il n’est pas déjà inscrit pour cette dernière. Le compétiteur voit alors un aperçu de la compétition, la spécialité et la catégorie de l’équipage étant présélectionnées en fonction de celles du compétiteur, et il ne lui reste plus qu’à choisir le nombre de places.

S’il veut modifier la catégorie, il peut seulement en choisir une plus élevée que la sienne (cela peut servir dans le cas d’un équipage à plusieurs places, où les participants auraient une faible différence d’âge mais une catégorie d’écart). En aucun cas un compétiteur ne pourra s’inscrire dans une compétition avec une catégorie inférieure à la sienne.

## Voir un équipage

Aspect de l’interface : [Voir un équipage](#_Voir_un_équipage).

Sur cette page, on retrouve un résumé de la compétition pour laquelle l’équipage est inscrit, la spécialité, la catégorie, le nombre de places et le nombre de participants valides (c’est-à-dire acceptés par l’entraîneur pour les mineurs) de l’équipage. Si l’équipage n’a qu’une seule place, on retrouve aussi le nom de participant, et la possibilité de désinscrire l’équipage pour ce participant.

Si l’équipage a plusieurs places, un bouton est disponible pour voir les participants. Si l’utilisateur est un participant, il aura également un bouton pour accéder à la page des invités. Si l’utilisateur est un invité, il pourra accepter ou refuser l’invitation depuis cette page.

Un message indique si l’équipage est valide (s’il a assez de participants validés) ou non.

## La gestion des invités d’un équipage

Aspect de l’interface : [Gestion des invités d’un équipage](#_Gestion_des_invités).

Seuls les compétiteurs faisant partie d’un équipage peuvent accéder à sa gestion des invités. Ils peuvent sélectionner un compétiteur qui ne participe pas déjà à la compétition et dont la catégorie est inférieure ou égale à celle de l’équipage.

Une liste est disponible, résumant les compétiteurs invités qui n’ont pas encore acceptés l’invitation. Un bouton pour annuler l’invitation est également disponible.

## La gestion des participants d’un équipage

Aspect de l’interface : [Gestion des participants d’un équipage](#_Gestion_des_participants).

L’utilisateur n’a pas besoin de faire partie de l’équipage pour voir cette page. Un invité peut la consulter pour voir les autres participants, ou l’entraineur pour valider une participation. Une liste regroupant les membres de l’équipage est disponible, en indiquant son statut (valide ou non).

Si un membre de l’équipage visite cette page, il a un bouton « Se désinscrire de l’équipage » en face de son nom afin d’annuler sa participation. Si l’équipage n’a qu’un membre et qu’il se désinscrit, l’équipage est supprimé.

# Le profil

## La vue du profil

Aspect de l’interface : [La vue du profil](#_La_vue_du).

Sur cette page, l’utilisateur peut voir ses informations personnelles, comme son nom, prénom, sa date de naissance… S’il est également compétiteur, sa catégorie, sa spécialité, son objectif de saison et la validité de son certificat médical sont affichés. Il peut cliquer sur « Modifier » afin de changer ces informations.

## La modification du profil

Aspect de l’interface : [La modification du profil](#_La_modification_du).

L’utilisateur peut modifier ses informations personnelles sur cette page. Le formulaire est déjà pré rempli avec les anciennes valeurs. De la même manière que pour la vue du profil, si l’utilisateur est aussi compétiteur il pourra modifier sa spécialité et son objectif de saison également.

## La liste des compétitions

Aspect de l’interface : [La liste des compétitions d’un compétiteur](#_La_liste_des).

Seuls les compétiteurs ont accès à cette partie de leur profil. Ils peuvent voir les compétitions pour lesquelles ils sont inscrits, et peuvent accéder à la fiche de la compétition directement depuis cette page.

## Les invitations

Aspect de l’interface : [Les invitations d’un compétiteur](#_Les_invitations_d’un).

Ici aussi, seuls les compétiteurs peuvent y accéder. S’ils ont des invitations, cela est indiqué à côté de l’onglet « Mes invitations » par le nombre d’invitations. Ils peuvent alors voir la liste de leurs invitations, voir l’équipage correspondant et accepter ou refuser cette invitation depuis cette liste.

# Les bénévoles

## Inscription d’un licencié comme bénévole

Aspect de l’interface : [Inscription d’un licencié comme bénévole](#_Inscription_d’un_licencié).

Tout utilisateur, quel que soit son rôle, peut s’inscrire comme bénévole depuis la fiche d’une compétition. Il a alors juste à entrer son rôle et à valider sa participation pour être ajouté aux bénévoles pour la compétition.

## Inscription d’un non-licencié comme bénévole

Aspect de l’interface : [Inscription d’un non-licencié comme bénévole](#_Inscription_d’un_non-licencié).

L’administrateur et le secrétaire peuvent ajouter des personnes non licenciées comme bénévoles pour une compétition. Pour cela, l’administrateur doit au préalable avoir ajouté la personne, et cette dernière ne doit pas avoir de licence. Ils peuvent alors sélectionner dans une liste déroulante une personne correspondant à ces critères, choisir le rôle et l’ajouter comme bénévole.

## La liste des bénévoles

Aspect de l’interface : [Liste des bénévoles](#_Liste_des_bénévoles).

La liste des bénévoles est divisée en deux parties : la liste des bénévoles officiels et la liste des bénévoles non licenciés. Ces deux listes résument les personnes et leur rôle en tant que bénévole pour la compétition correspondante.

Si l’utilisateur est un bénévole officiel, il peut se désinscrire depuis cette page. Pour désinscrire un bénévole non licencié, il faut que l’utilisateur soit administrateur ou secrétaire.

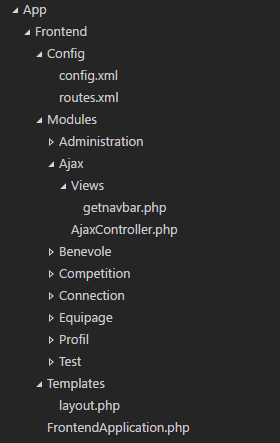
Conclusion

Tous les éléments présentés composent tous ensemble notre application. Les différentes parties se complètent de manière à former un ensemble cohérent, et répondent ainsi au cahier des charges. Pour mener à bien ce projet, plusieurs technologies ont été utilisées. A savoir : du PHP en grande majorité, mais aussi du XML, de l’ajax et du jquery.

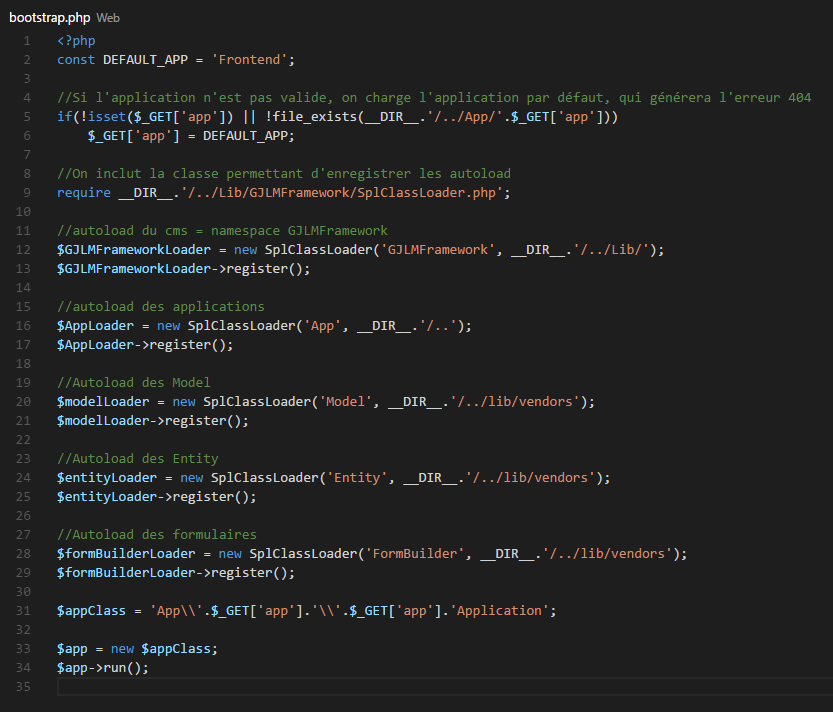
Il est à noter que les explications sont les plus succinctes possible afin de respecter la limite de pages. Pour un aperçu plus complet, le code le l’application est disponible dans le lien donné en introduction.

# Annexes

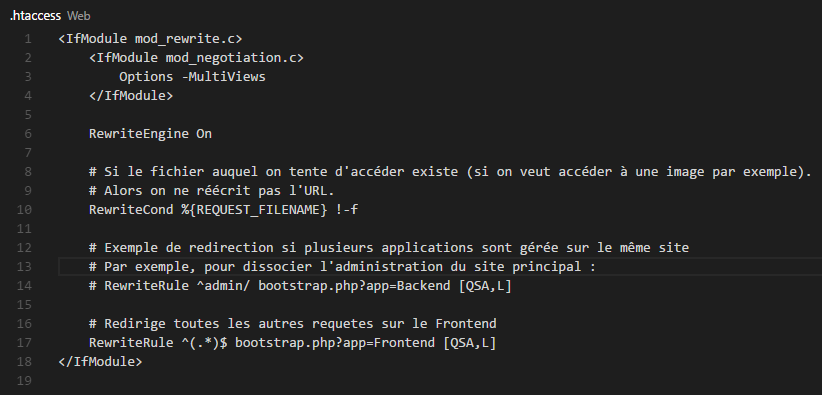
## [Structure du dossier App](#_Le_dossier_App)



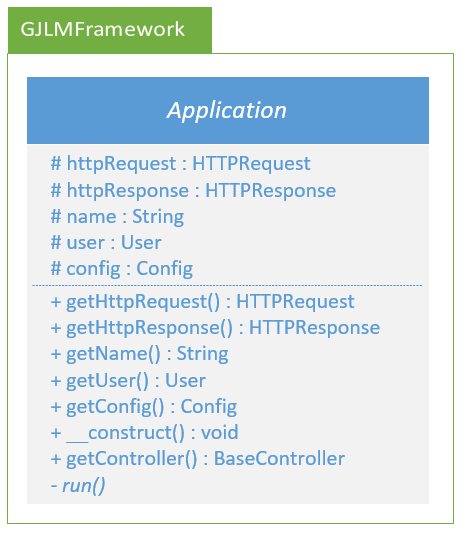
## [Le fichier bootstrap.php](#_Le_fichier_Web/bootstrap.php)



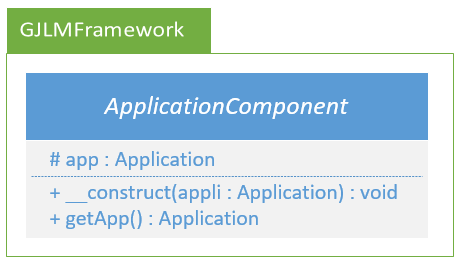
## [Le fichier .htaccess](#_Le_fichier_Web/.htaccess)



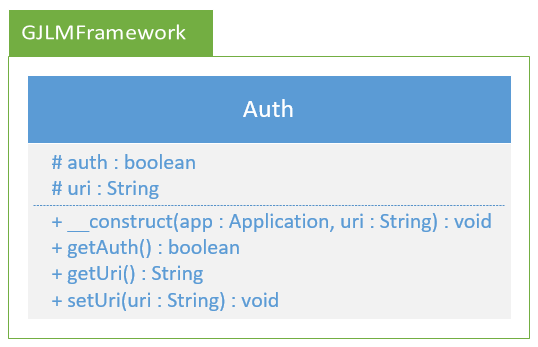
## [Diagramme de classe : Application](#_La_classe_Application)



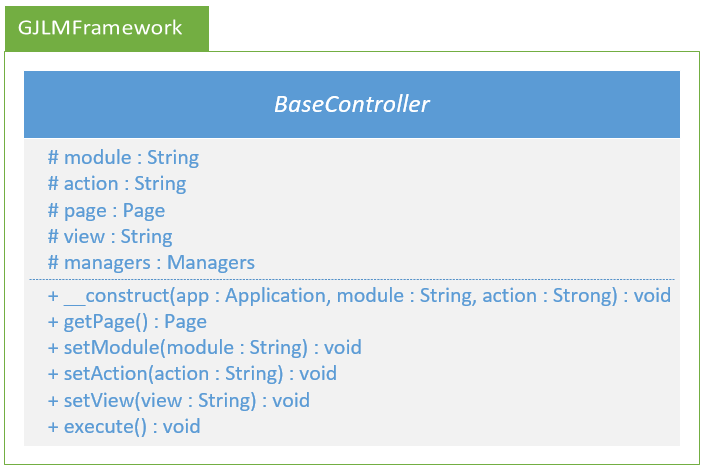
## [Diagramme de classe : ApplicationComponent](#_La_classe_ApplicationComponent)



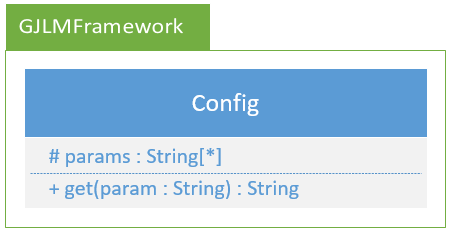
## [Diagramme de classe : Auth](#_La_classe_Auth)



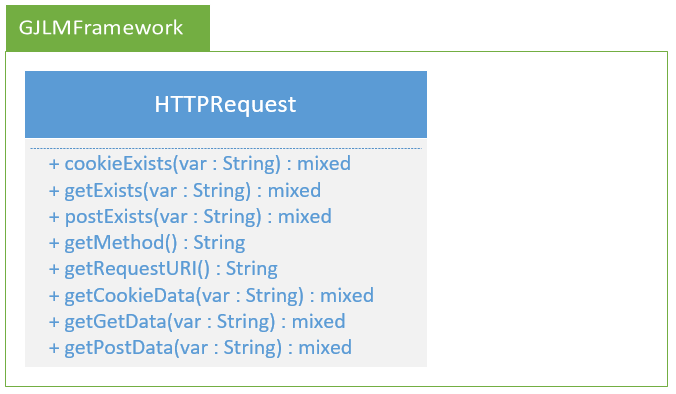
## [Diagramme de classe : BaseController](#_La_classe_BaseController)



## [Diagramme de classe : Config](#_La_classe_Config)



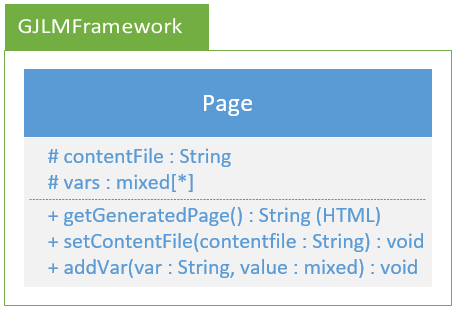
## [Diagramme de classe : HTTPRequest](#_La_classe_HTTPRequest)



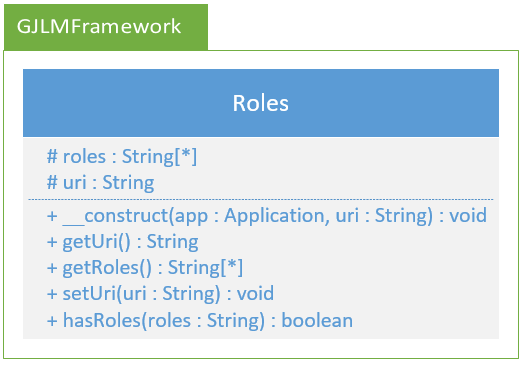
## [Diagramme de classe : HTTPResponse](#_La_classe_HTTPResponse)



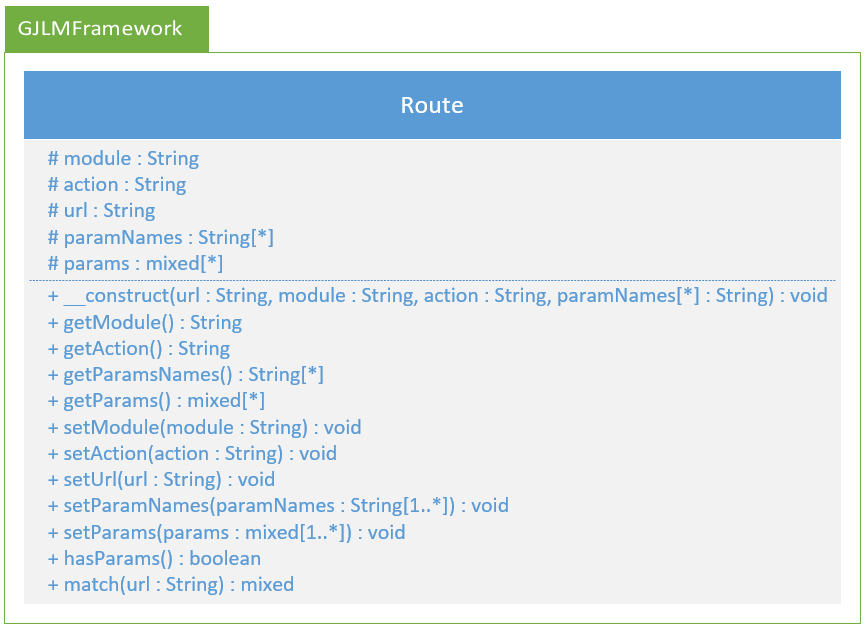
## [Diagramme de classe : Page](#_La_classe_Page)



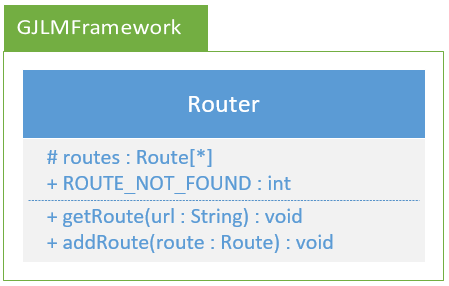
## [Diagramme de classe : Roles](#_La_classe_Roles)



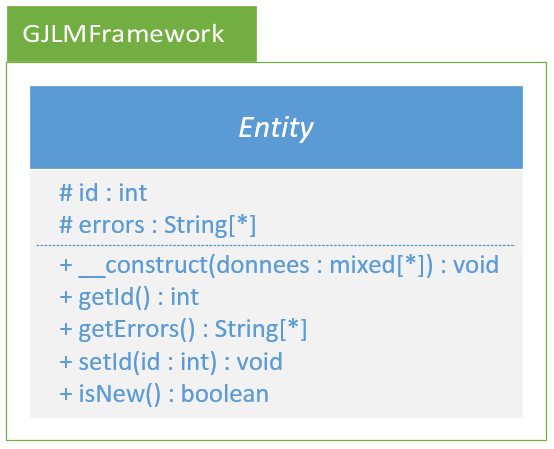
## [Diagramme de classe : Route](#_La_classe_Route)



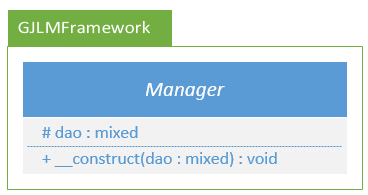
## [Diagramme de classe : Router](#_La_classe_Router)



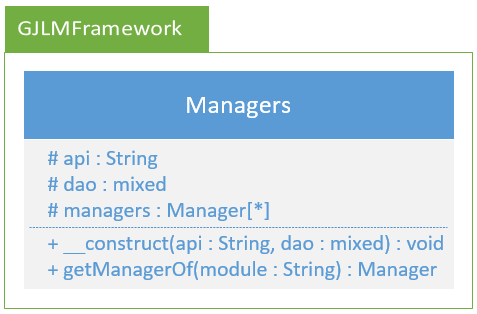
## [Diagramme de classe : Entity](#_La_classe_Entity)



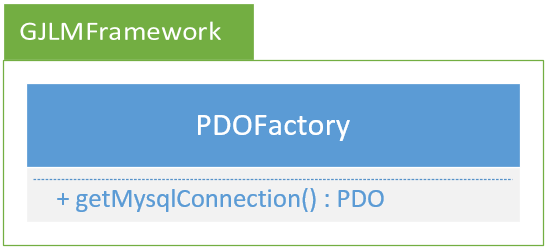
## [Diagramme de classe : Manager](#_La_classe_Manager)



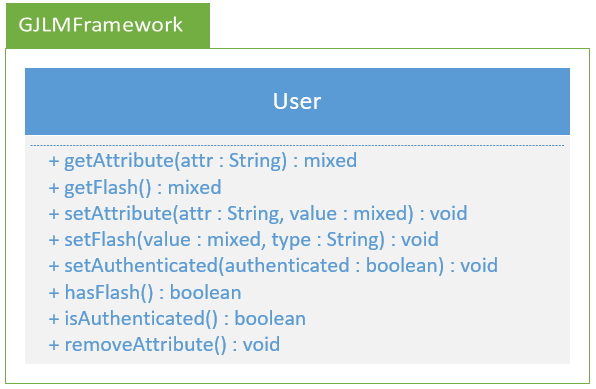
## [Diagramme de classe : Managers](#_La_classe_Managers)



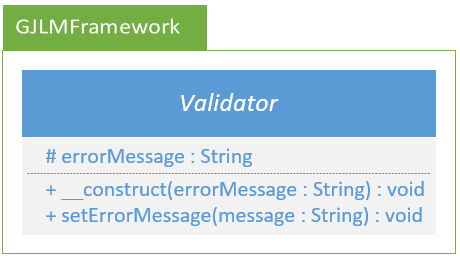
## [Diagramme de classe : PDOFactory](#_La_classe_PDOFactory)



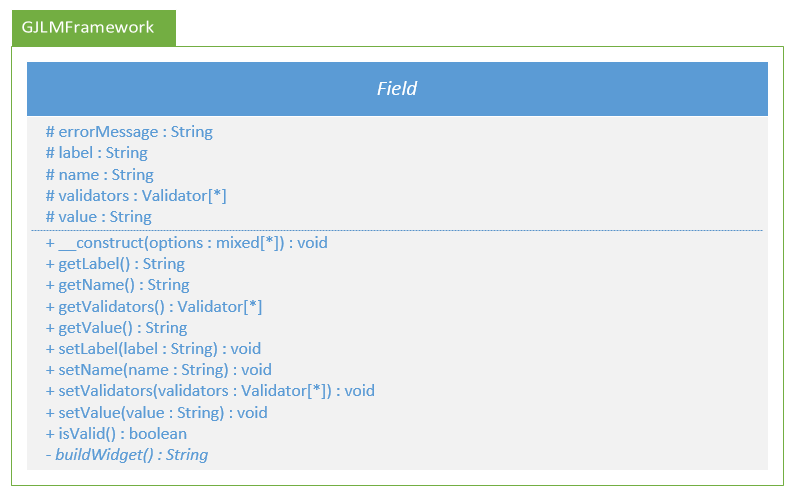
## [Diagramme de classe : User](#_La_classe_User)



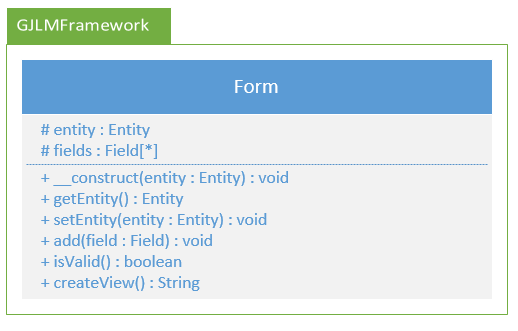
## [Diagramme de classe : Validator](#_La_classe_Validator)



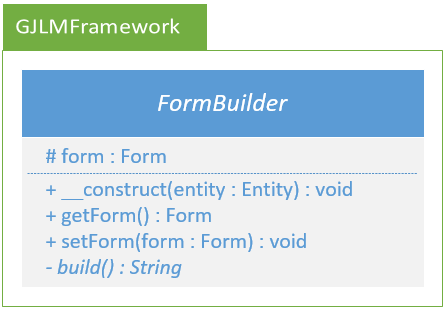
## [Diagramme de classe : Field](#_La_classe_Field)



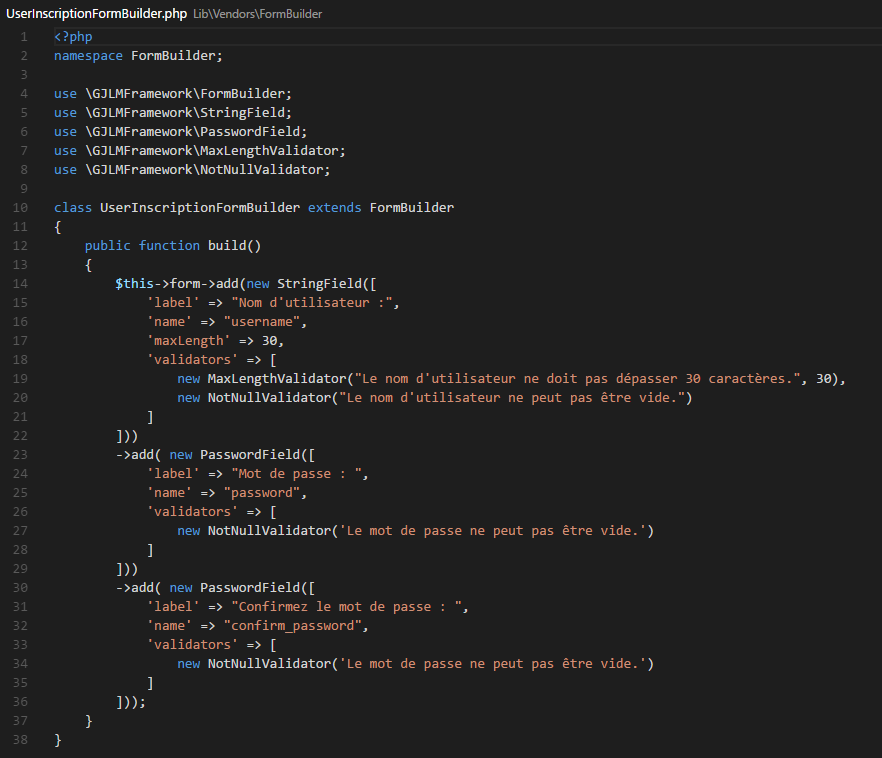
## [Diagramme de classe : Form](#_La_classe_Form)



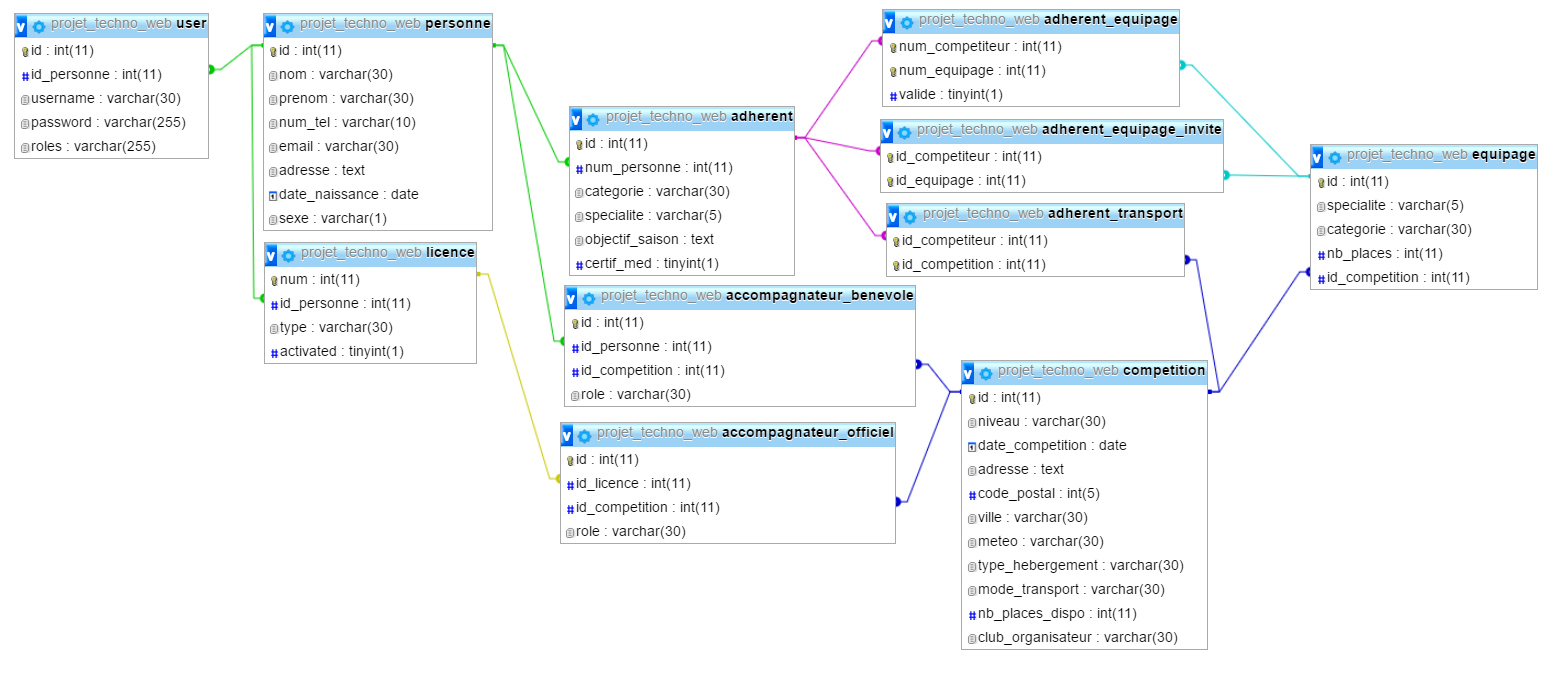
## [Diagramme de classe : FormBuilder](#_La_classe_FormBuilder)



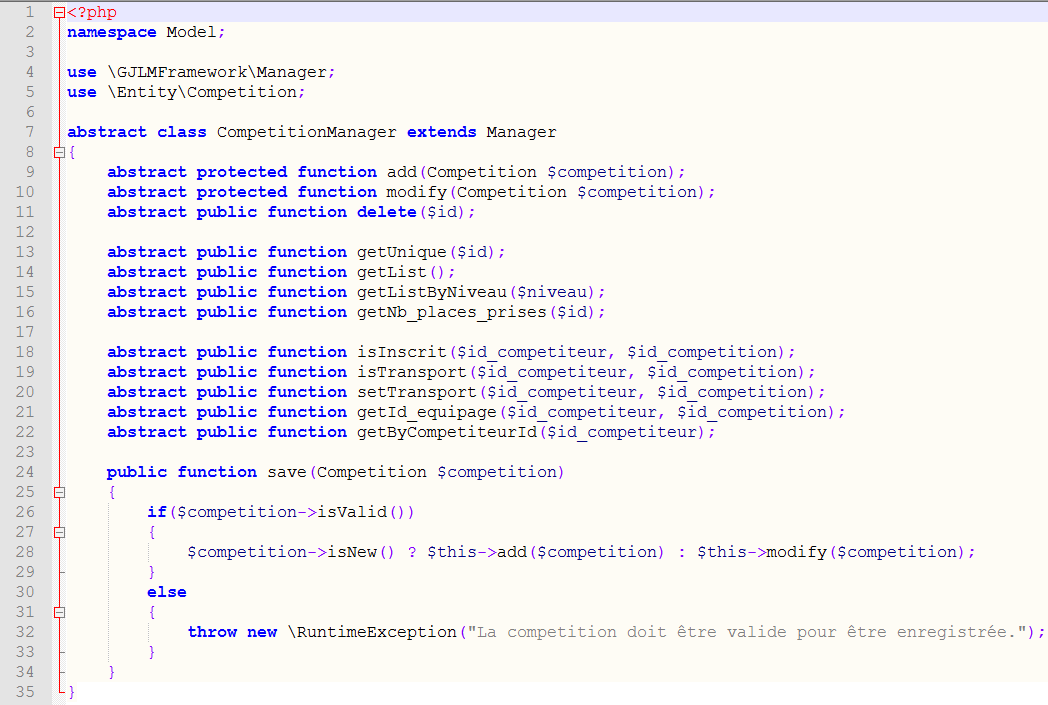
## [Exemple de classe dérivant FormBuilder](#_La_classe_FormBuilder)



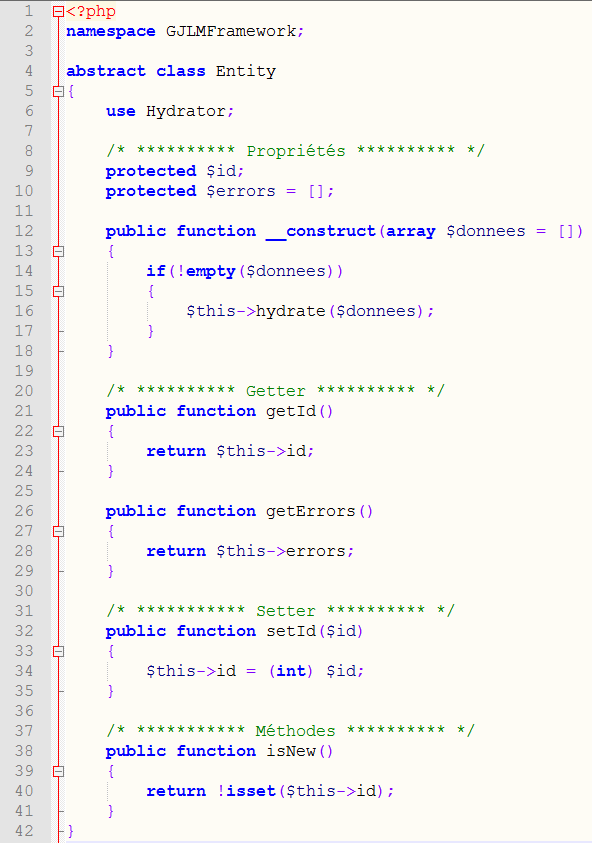
## Schéma de la base de données



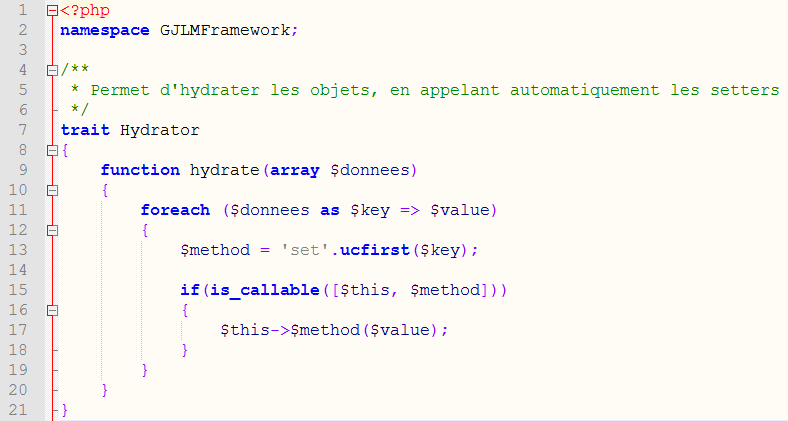
## Exemple de classe mère abstraite d’un manager



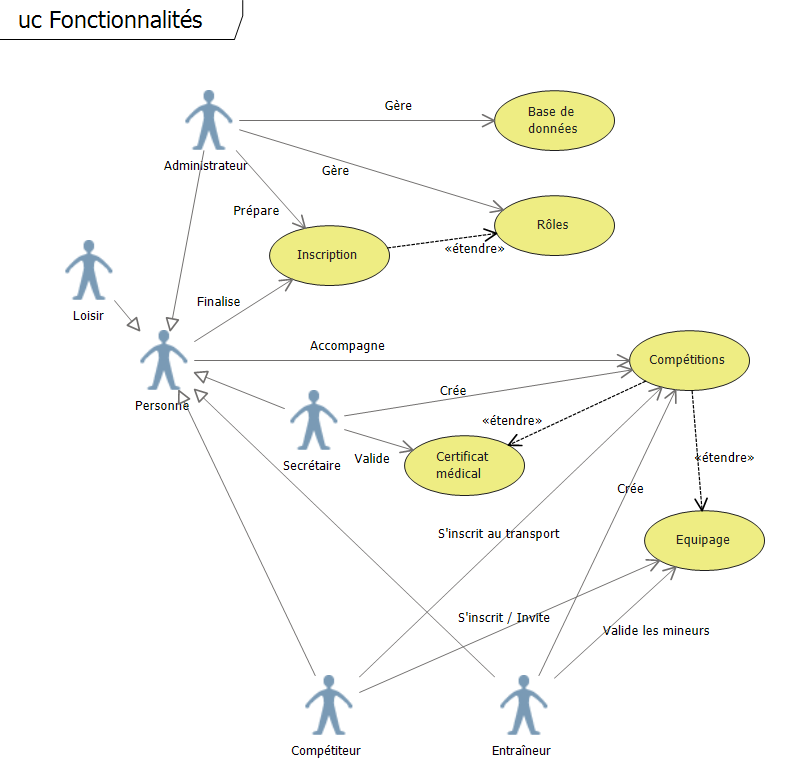
## La classe Entity



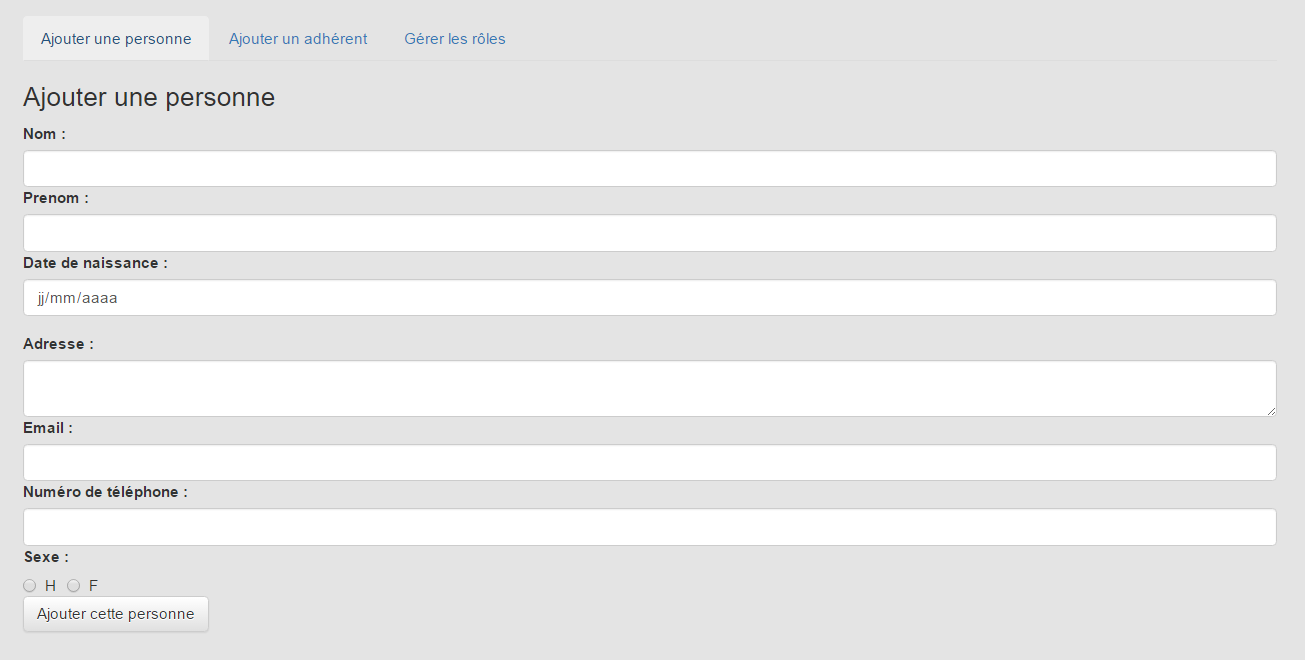
## Le trait Hydrator



## Use case



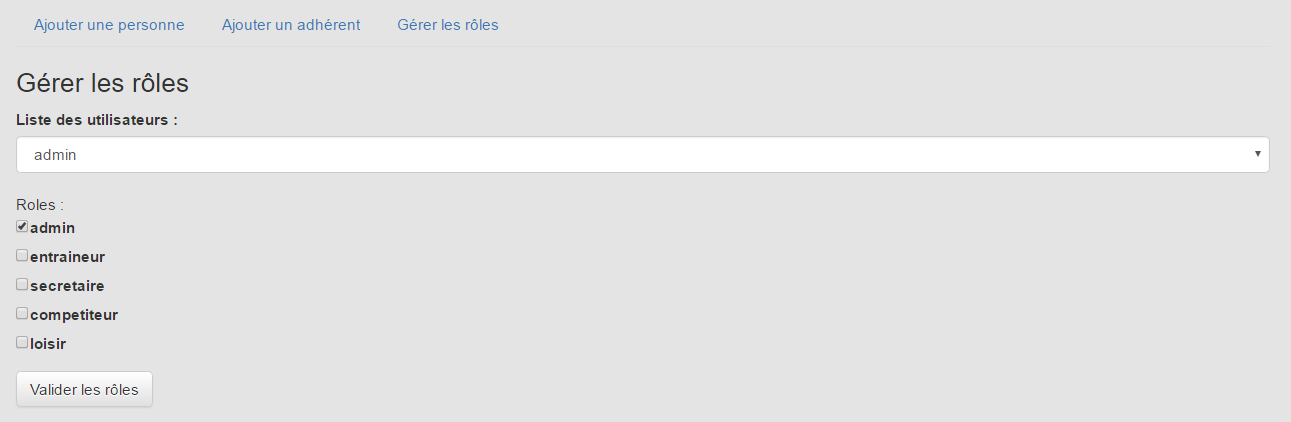
## L’ajout de personne



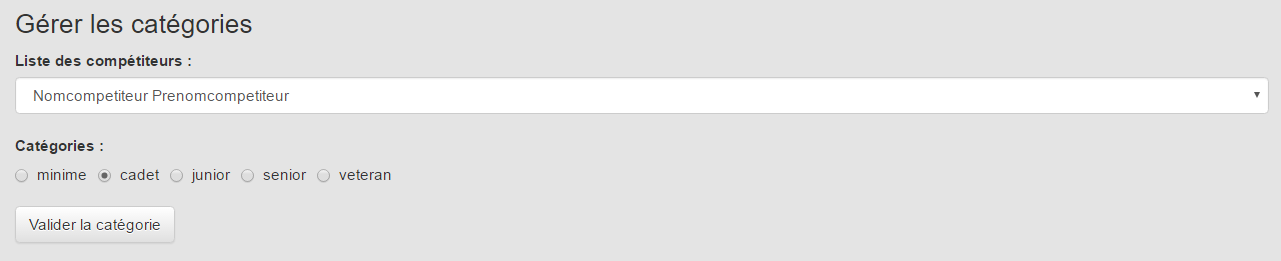
## L’ajout d’adhérent



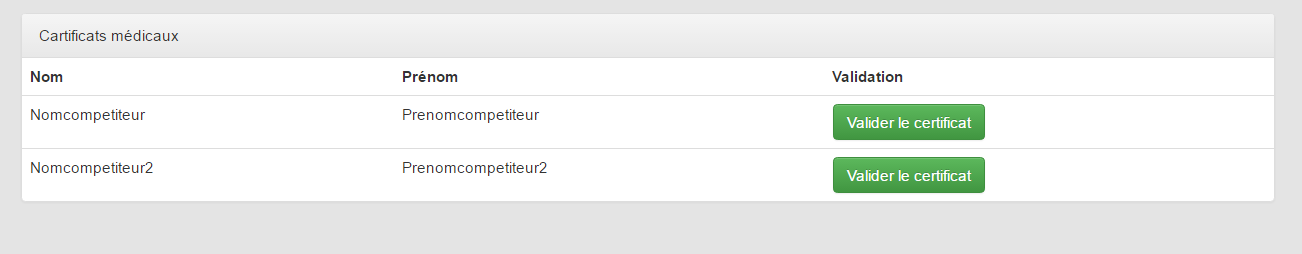
## La gestion des rôles



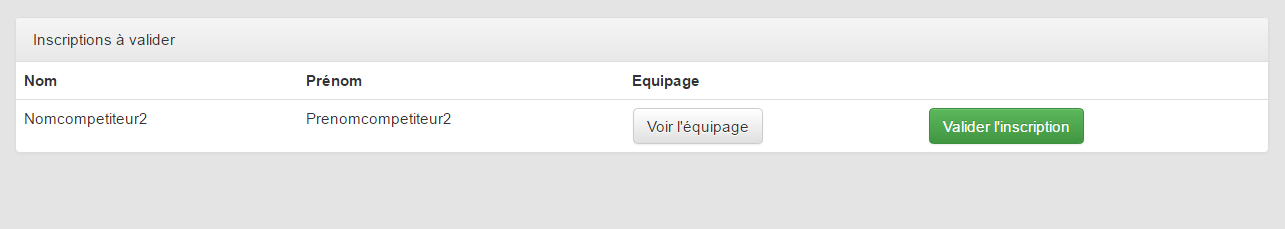
## La gestion des catégories



## La gestion des certificats médicaux



## Valider les inscriptions



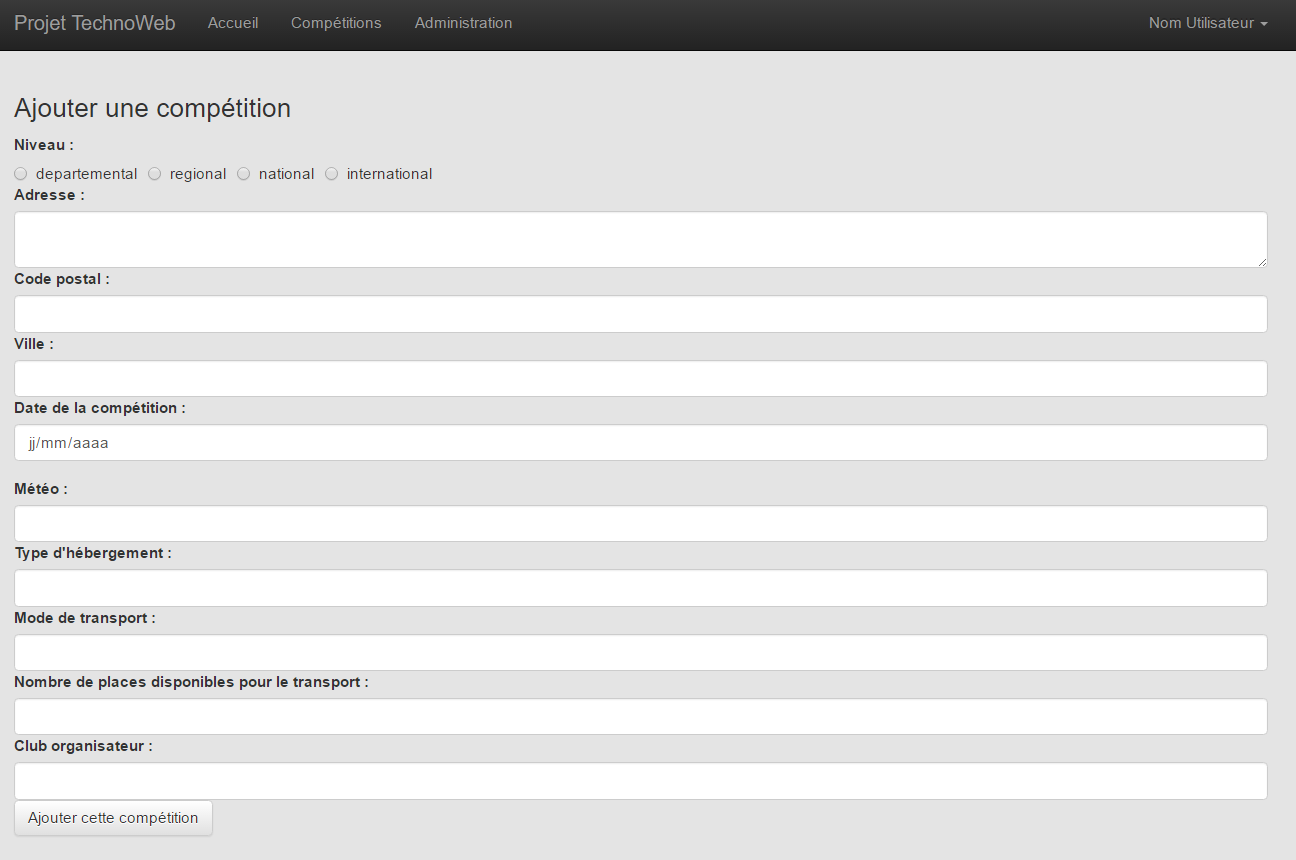
## Les étapes de l’inscription



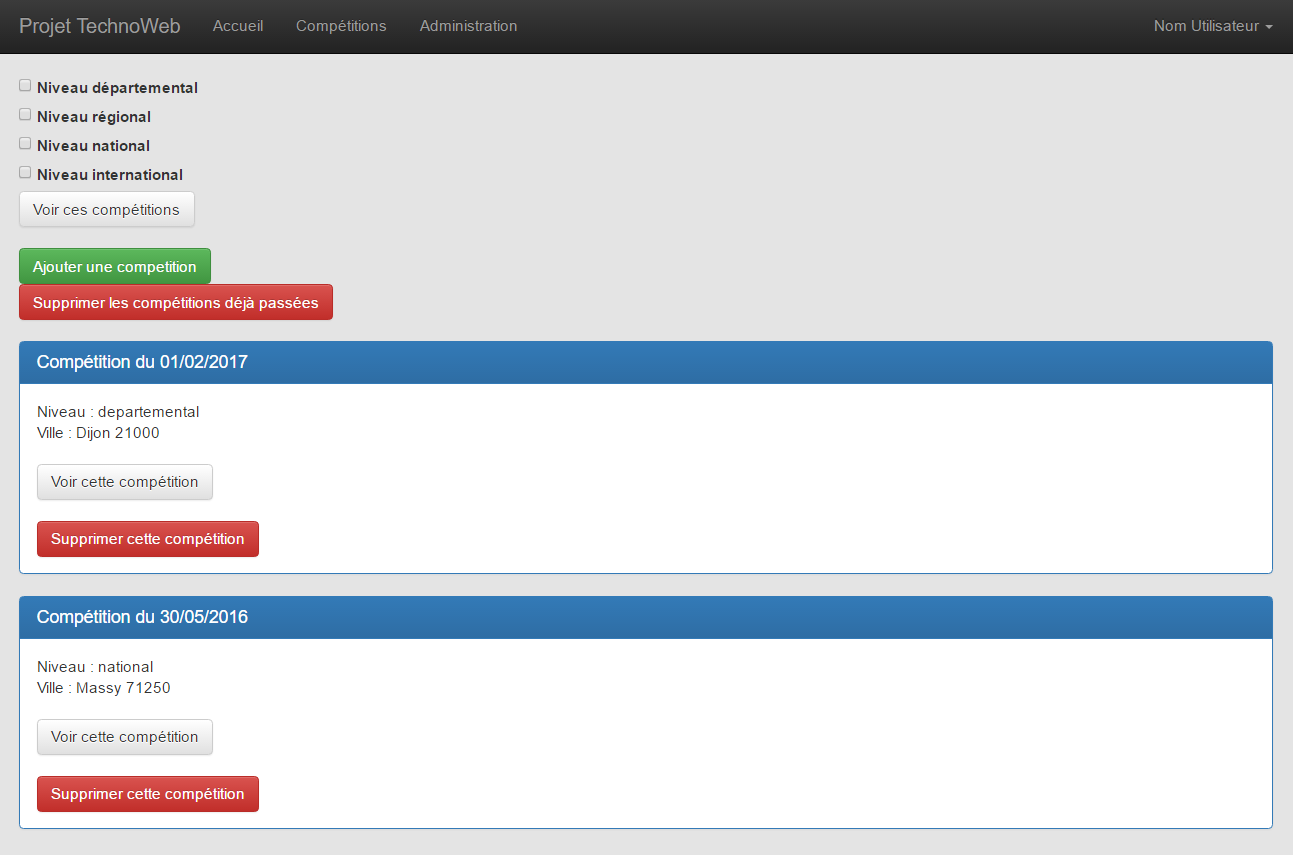
## La connexion



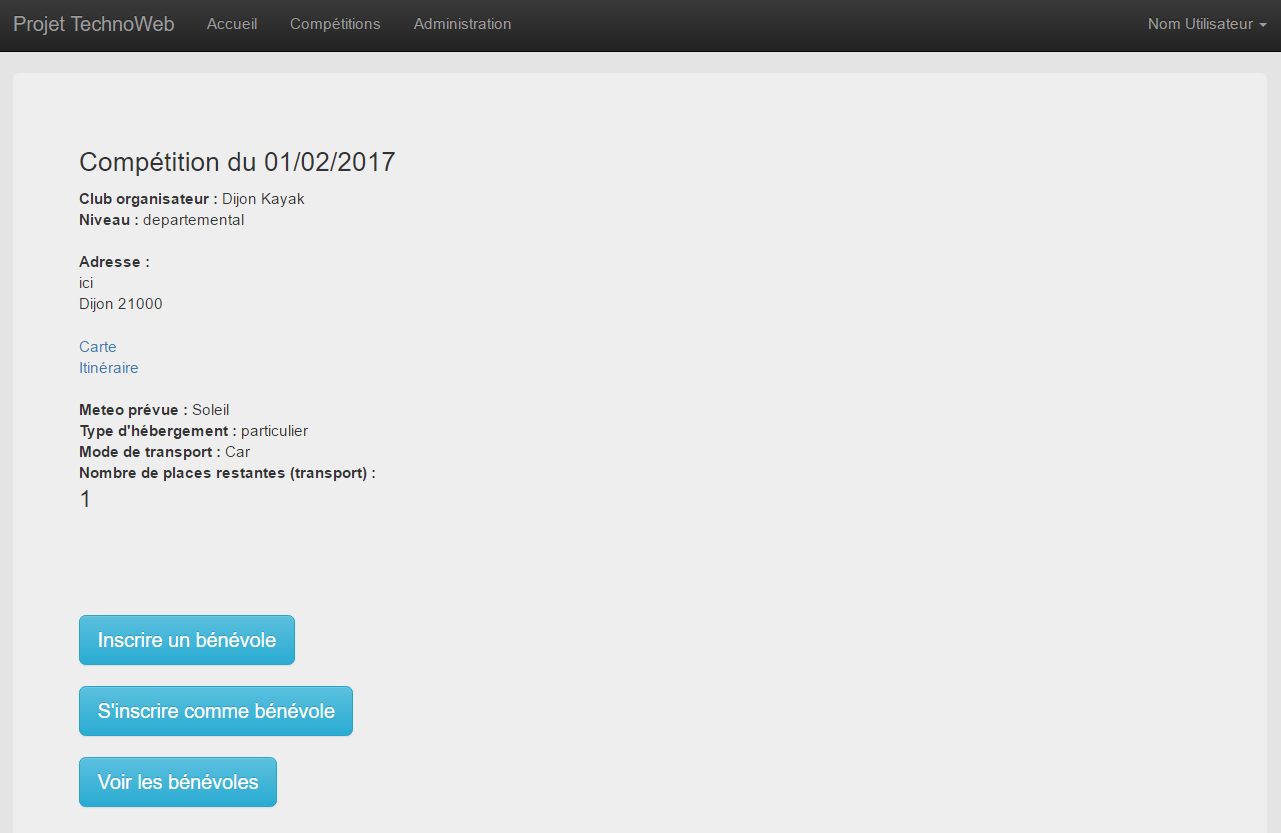
## Ajouter une compétition



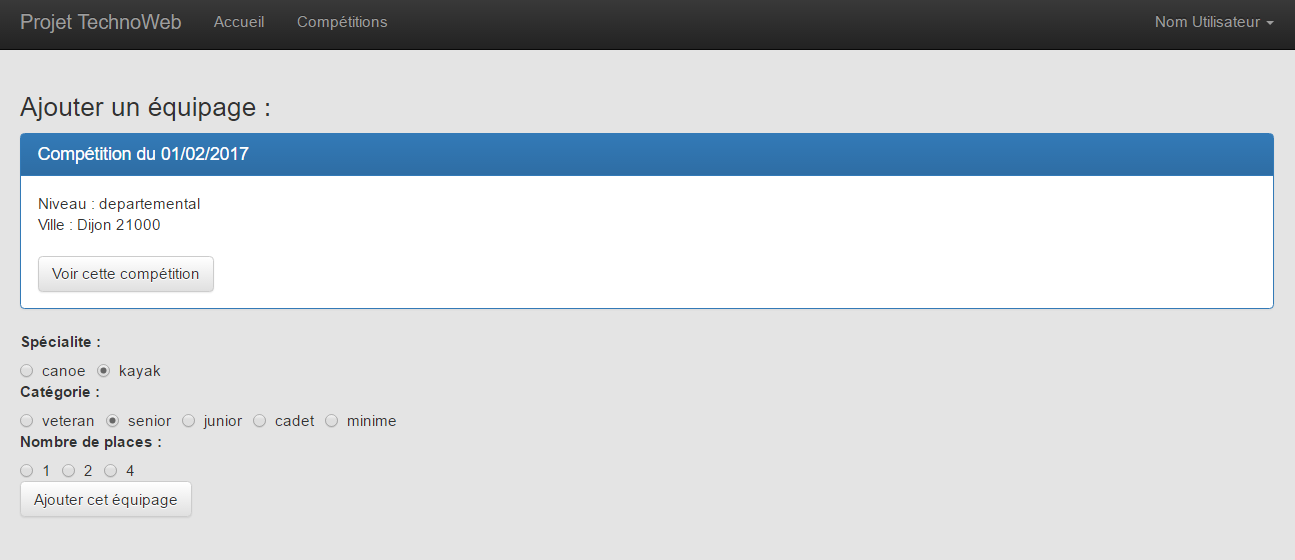
## Liste des compétitions



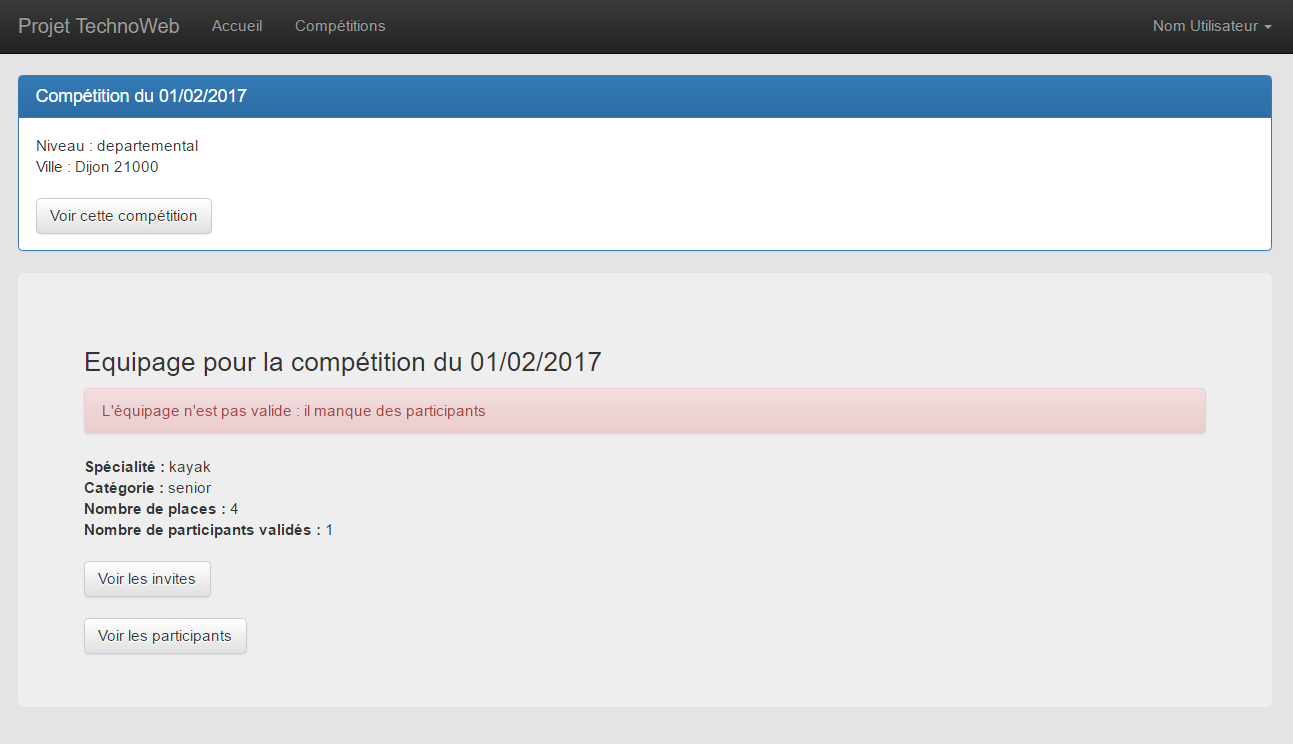
## Vue d’une compétition



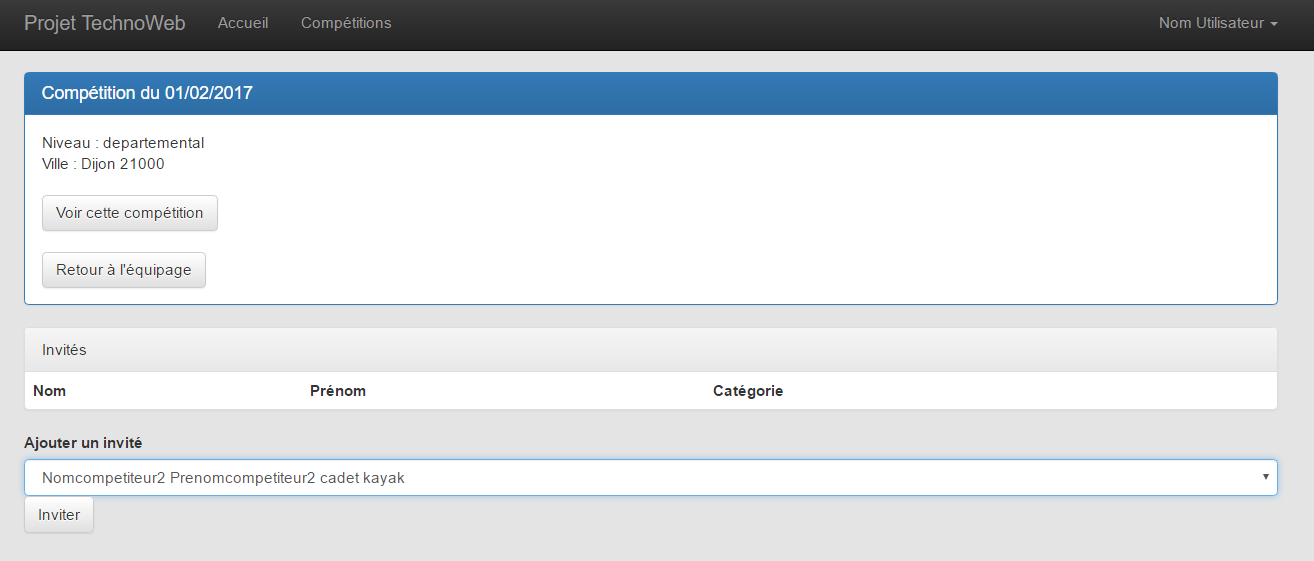
## Ajouter un équipage



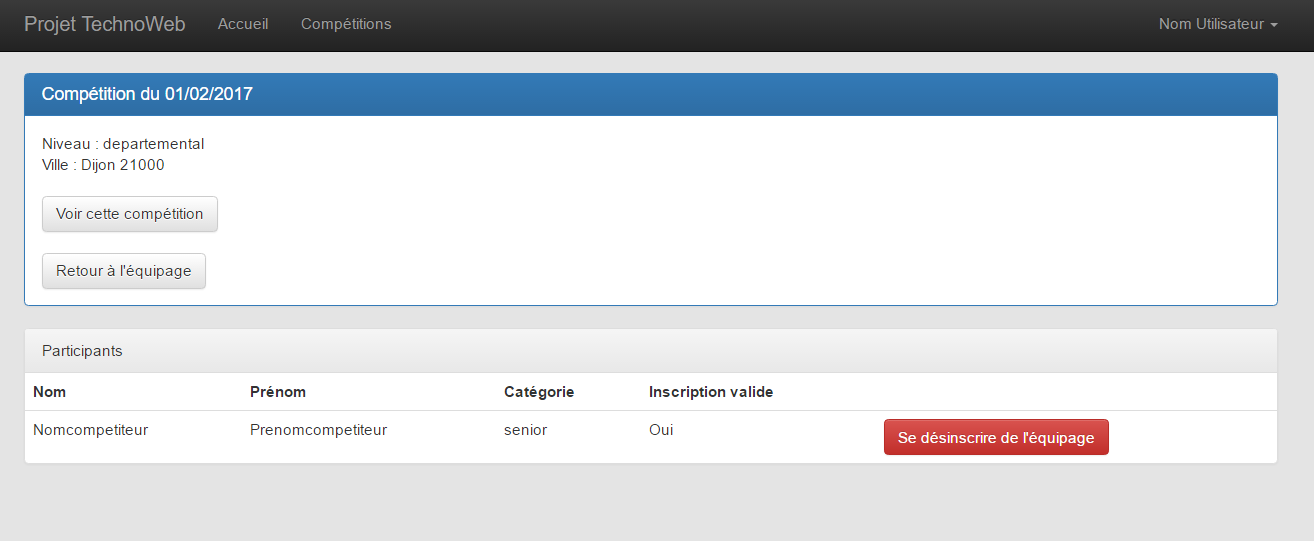
## Voir un équipage



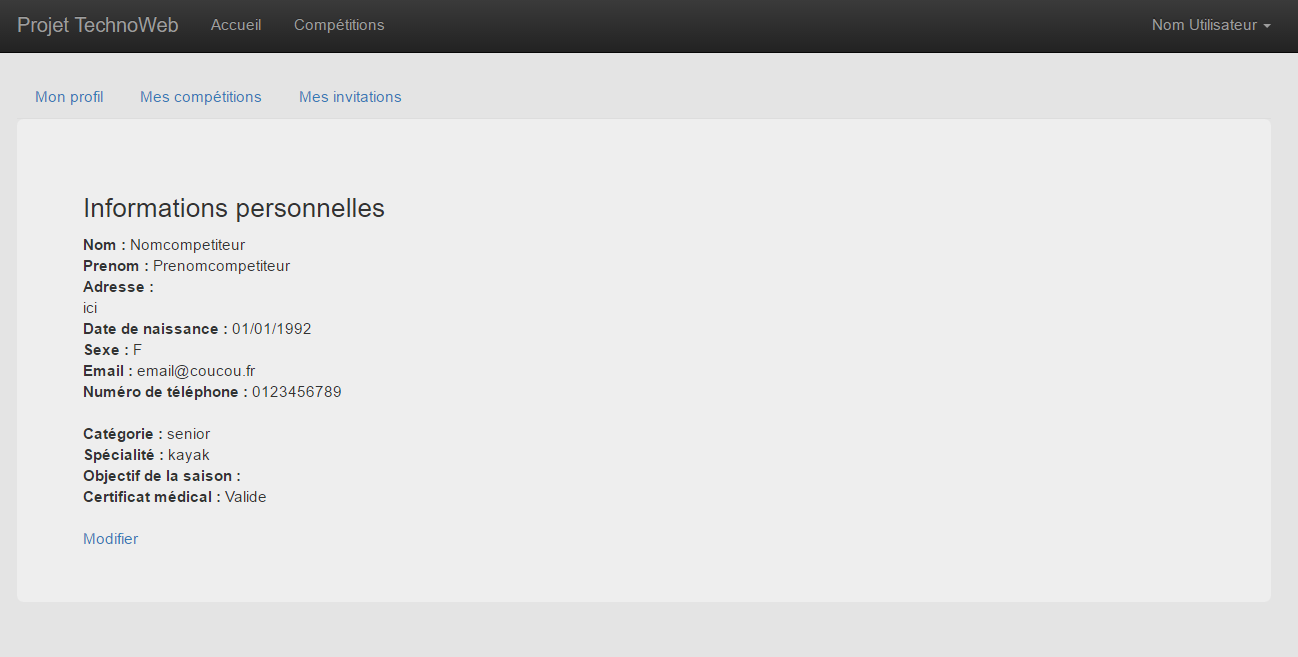
## Gestion des invités d’un équipage



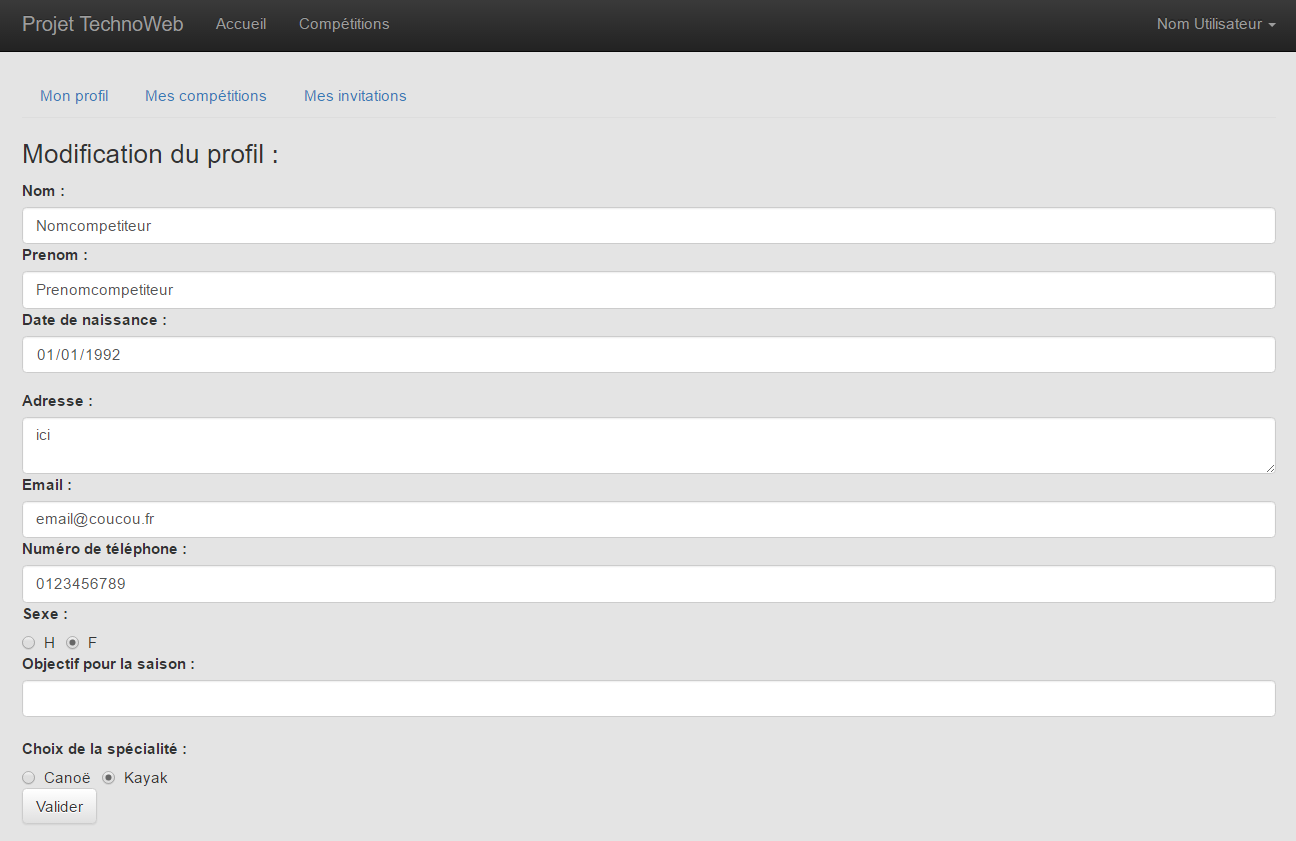
## Gestion des participants d’un équipage



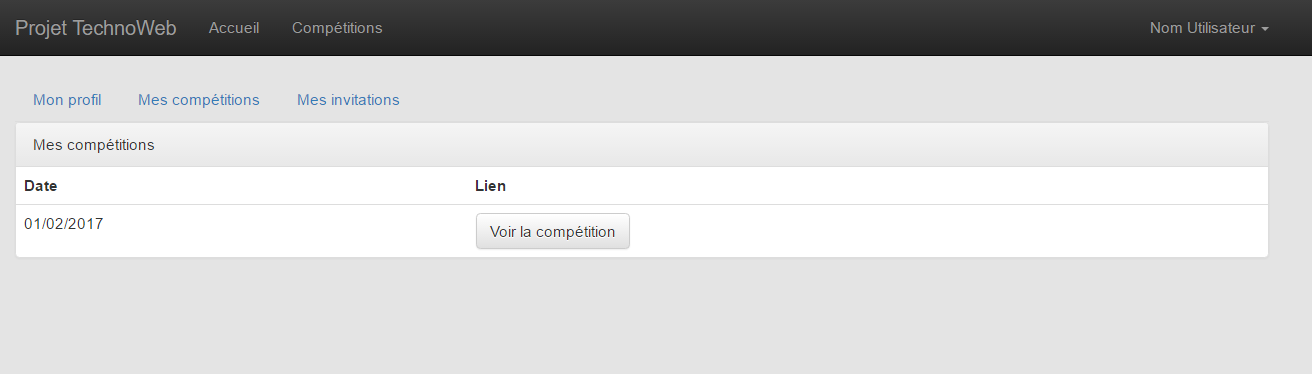
## La vue du profil



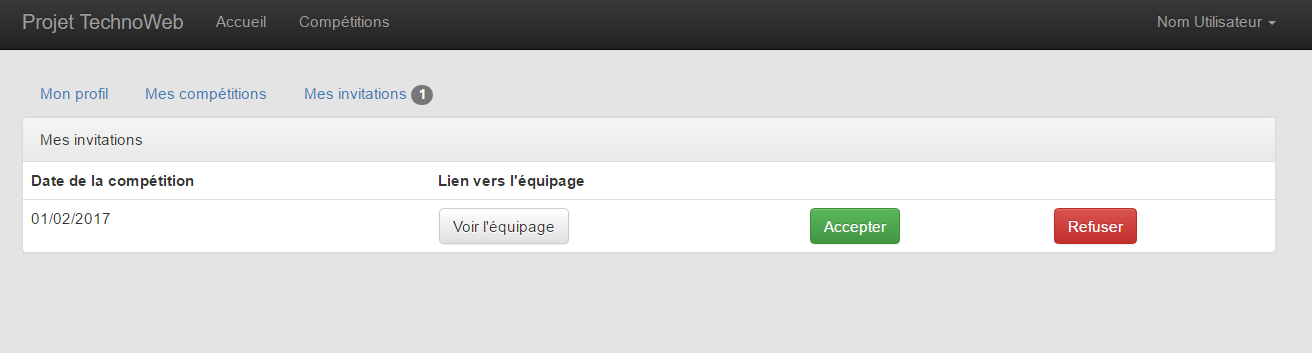
## La modification du profil



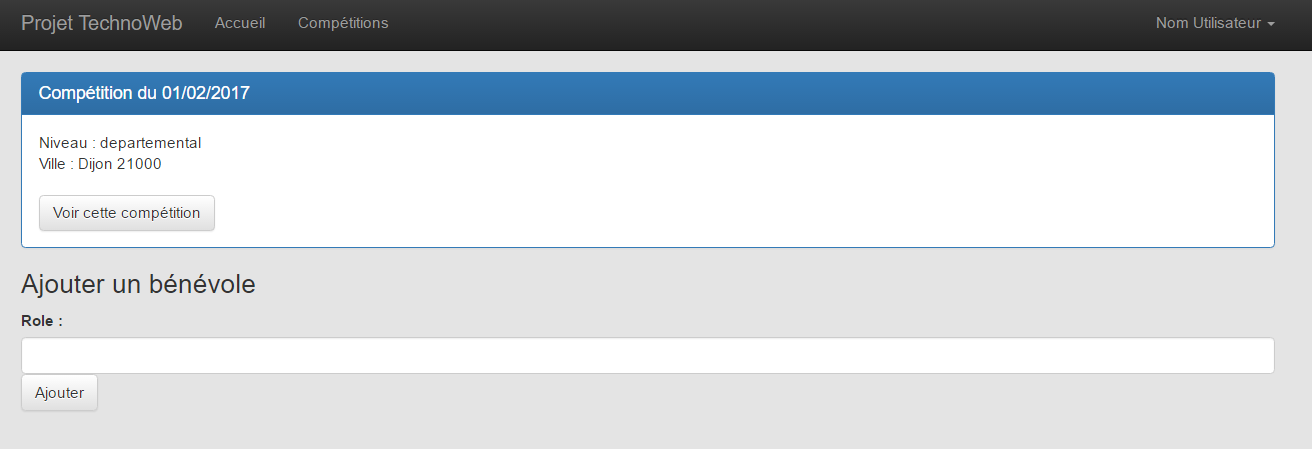
## La liste des compétitions d’un compétiteur



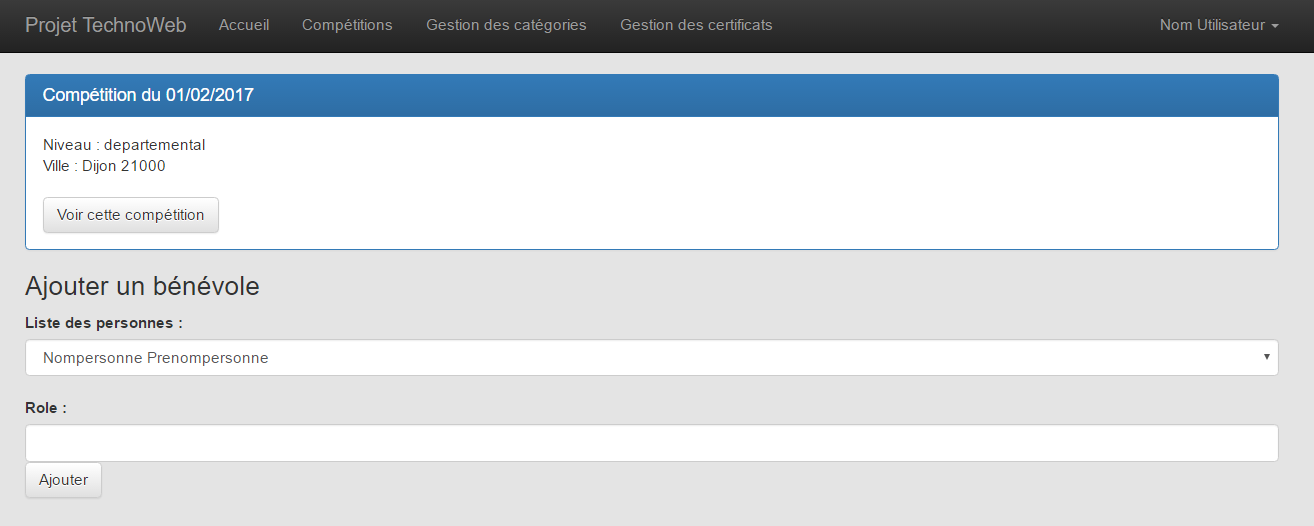
## Les invitations d’un compétiteur



## Inscription d’un licencié comme bénévole



## Inscription d’un non-licencié comme bénévole



## Liste des bénévoles

