

# PHP Orienté Objet - IPSSI Decembre 2017 - TP Noté

---

Le code doit être versionné sous Git et pushé sur Github, Bitbucket ou Gitlab, avec les droits de lecture pour respectivement `tdutrion`, `thomas_dutrion` et `tdutrion`.

## Scenario

Vous devez développer un système de gestion de communauté organisant des meetings.

- Un meeting commence à une date donnée (fixe et interchangeable)
- Un meeting fini à une date donnée (fixe et interchangeable)
- Un meeting a un titre et une description
- Un utilisateur peut s'inscrire à un ou plusieurs meetings
- Un utilisateur peut être organisateur d'un meeting (en quel cas il doit être dans la liste des organisateurs et ne peut être dans la liste des participants)
- Un utilisateur peut créer un meeting et ajouter des participants et des organisateurs
- Un organisateur d'un meeting peut supprimer le meeting
- Un utilisateur peut voir la liste des meetings à venir
- Un utilisateur peut voir la liste des meetings (passés et à venir)
- Un utilisateur peut consulter une fiche pour un meeting donné, qui contient les dates de début et de fin, ainsi que la liste des organisateurs et des participants enregistrés

## Skeleton et installation

Pour simplifier le démarrage du projet, le skeleton que nous avons créé en cours est fourni et récupérable à l'adresse suivante : <https://github.com/engineor/ipssi-oop-php-2017>

Pour nettoyer l'historique du projet git, supprimez le dossier `.git` ( `rm -r .git` ) puis réinitialisez votre repository à l'aide de la commande `git init`.

## Notation

Les notes dépendront du nombre de fonctionnalités réalisés, pondérées par difficultés (la réalisation de toutes les fonctions avec un style de code minimal rapportera moins de points que la réalisation de moins de fonctionnalités mais avec un meilleur aspect d'un point de vue du génie logiciel).

Sera regardé :

- le typage fort

- les namespaces et l'autoload (avec composer, en PSR-4)
- la longueur des classes et des méthodes
- la non utilisation de fonctions anonymes, remplacées par des classes invokables
- le découpage en classes à responsabilité limité, et la composition d'objets (injection de dépendance) en utilisant des interfaces lorsque c'est nécessaire

Mots clés importants : `dependency injection` , `factory` , `psr-2` , `php-cs-fixer` , `final` , `declare(strict_types=1)` , `scalar type hint` , `return type` , `composition` , `inheritance` , `__construct` , `__invoke` , `Entity` , `Repository` , `PDO` .

## Requis (pour max 15/20)

---

Le minimum requis (pour avoir la moyenne uniquement) est d'avoir un système permettant l'affichage des valeurs depuis la base de donnée. La base de donnée est rempli directement soit en ligne de commande soit en utilisant une interface comme [Adminer](#) ou tout autre logiciel jugé adapté.

Les classes utilisées doivent être fonctionnelles et le site doit s'afficher correctement.

Les validations dans les entités ainsi que les types des attributs sont correct, si un choix technique est fait et ne semble pas évident, une note peut être ajoutée dans le `README.md` pour expliquer le choix lors de la correction.

Les liens entre les objets sont précautionneusement choisis (composition, héritage, aggregation, etc.). L'utilisation de classes appropriées ( `Repository` , `Exception` , `Entity` , `Factory` ) est requis pour montrer la compréhension de la responsabilité de chacun.

Utilisation d'exceptions pour les cas non standard.

## Bonus (entre 15 et 20)

---

Un container d'injection de dépendance est utilisé :

- [Zend ServiceManager](#)
- [PHP-DI](#)
- [Pimple](#)
- votre propre container d'injection

Un système de routing est utilisé :

- [FastRoute](#)
- [AuraRouter](#)
- Le routeur minimaliste utilisé en cours à titre d'exemple

**Il est par contre interdit d'utiliser le kernel applicatif ou classe application d'un framework (car je**

**veux voir si les concepts permettant de joindre les différentes parties sont bien acquise).**

Extra points pour du code défensif et pour le formatage PSR-2 complet.

Points pour l'utilisation de vues (Twig et Plates autorisé, mais attention à ne pas perdre votre temps dessus).

Points pour l'utilisation de UUID v4 à la place des id auto incrémentaux (voir `ramsey/uuid` ).

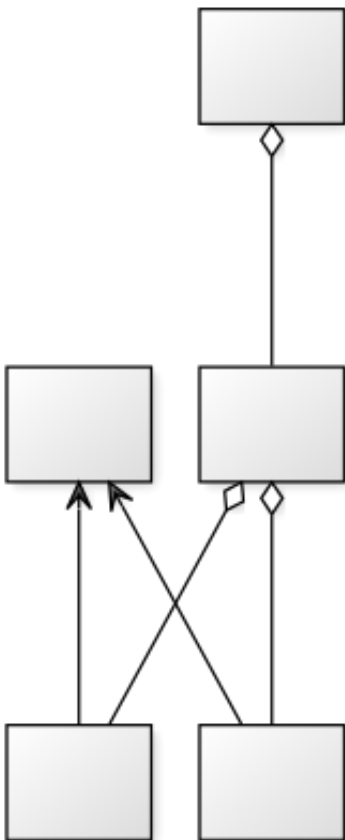
Utiliser un polymorphisme pour afficher une liste comportant tous les participants et tous les organisateurs d'un groupe en une seule fois.

# Aide

## Diagramme de classe

---

Voici un diagramme de classe possible qui ne modélise que les entités (sans montrer leur contenu). Votre projet n'est pas forcé de contenir exactement la même chose, mais devrait être relativement similaire quoi qu'il arrive.



## Traductions

---

- Communauté : Community
- Titre : Title
- Date de début : Start date
- Date de fin : End date
- Meeting : Meeting
- Inchangeable : Immutable
- Utilisateur : User
- Participant : Attendee
- Organisateur : Organiser (UK) / Organizer (US)
- Supprimer : Delete