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`.

Le projet est à rendre avant le dimanche 24/12/2017 à 23h59 *(histoire de quand même passer un bon Noël pour ceux qui le fêtent...*

Scenario

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

- Un meeting commence à une date donnée (fixe et inchangeable)
- Un meeting fini à une date donnée (fixe et inchangeable)
- Un meeting a un titre et une description
- Un utilisateur peu 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 nécéssaire

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 ajouté 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 comprehension 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 appliaction 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 formattage 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 organiseurs d'un groupe en une seule fois.

Aide

Si vous désirez modéliser votre code à l'aide de diagrammes UML (pour retrouver comment tout marche), vous pouvez utiliser des diagrammes de cas d'utilisation, diagramme de classe et diagramme de séquence.

Des exemples incomplets ou ne respectant pas entièrement la norme UML sont disponible ci-dessous.

Les logiciels que je vous conseille sont Visual Paradigm (version community) et YUML.me si les diagrammes sont assez simple.

Pour modéliser votre base de données, mysql workbench est de très bonne qualité (il produit des diagrammes EER).

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.

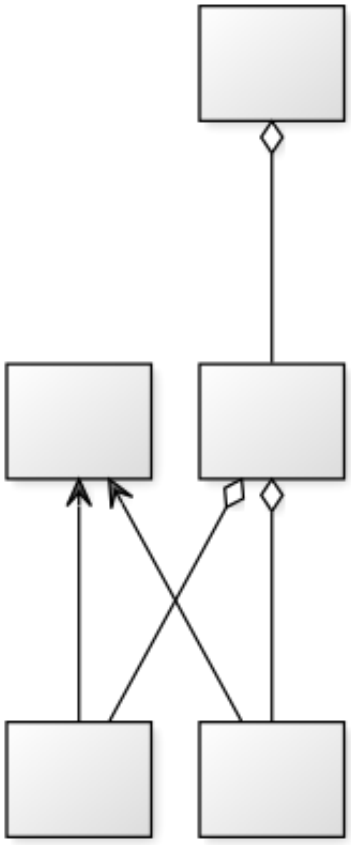
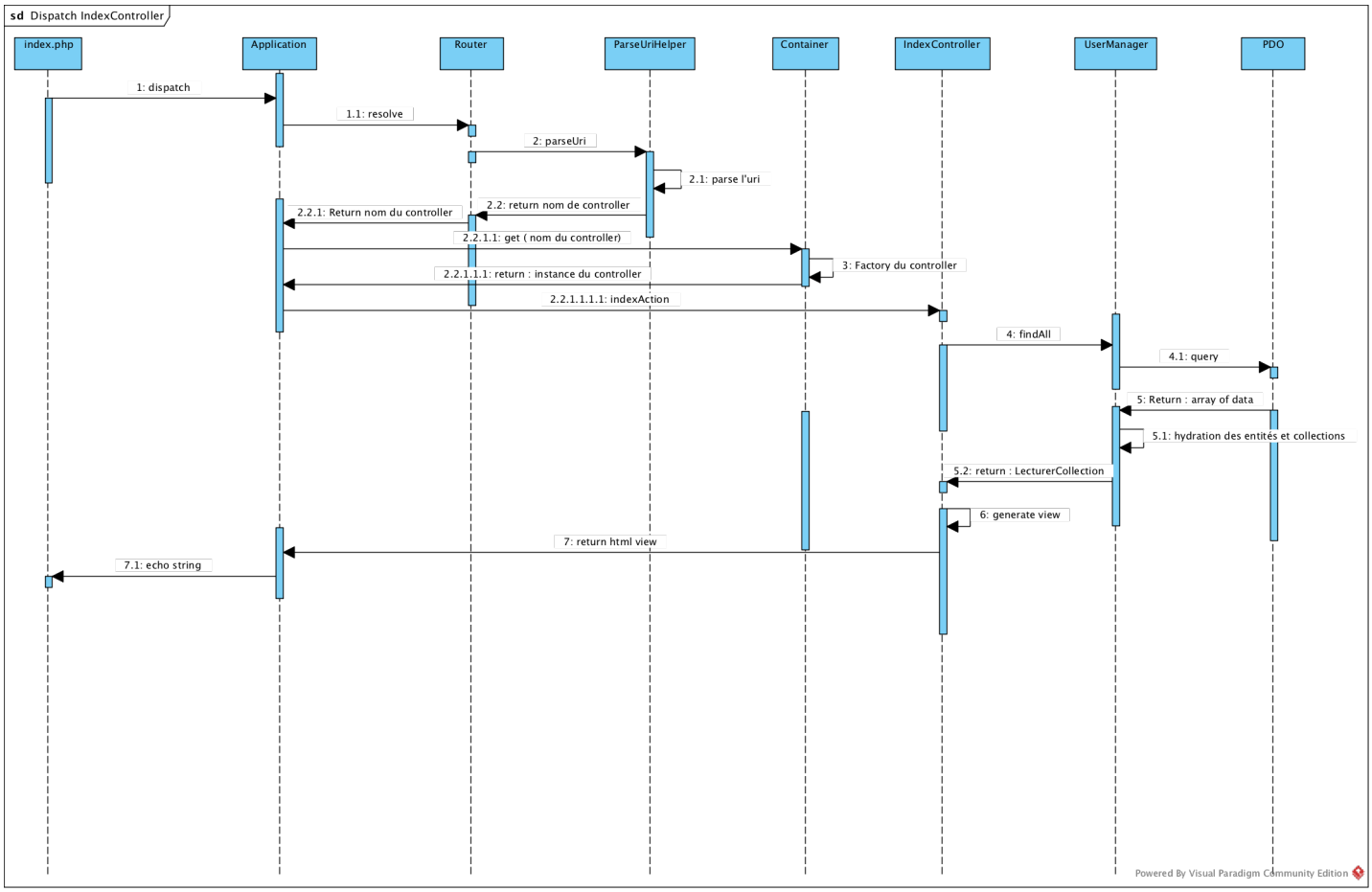


Diagramme de séquence

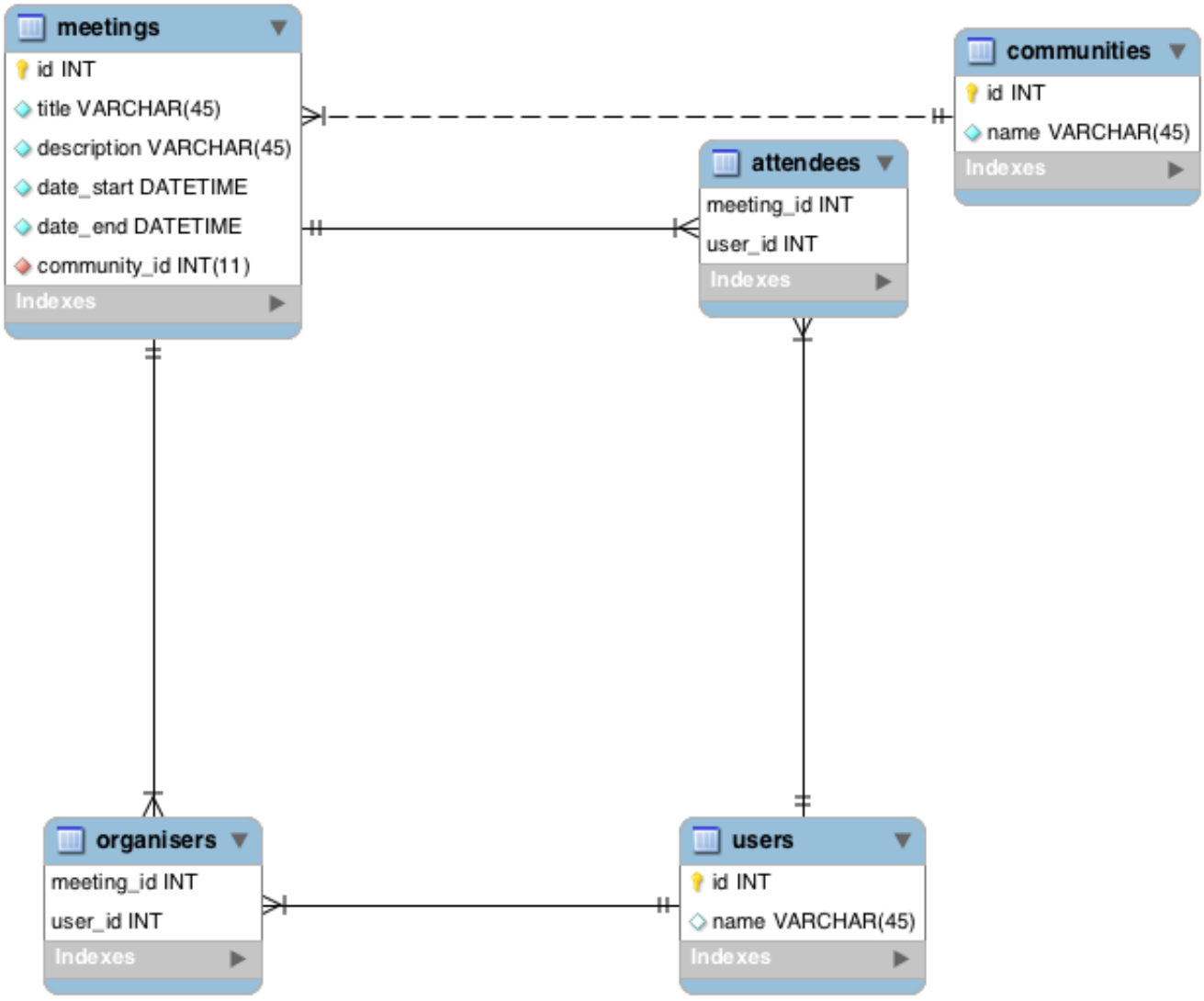
Bien que mal conçu (le nommage n'est pas cohérent et certains objets ne devraient pas être représentés, comme `index.php`), ce diagramme de séquence montre le parcours de la requête et les responsabilités de chaque composant du système.



Diagrammes EER de base de données possible

Ne vous limitez pas à ces schemas, mais voici deux schemas possibles de base de données permettant de réaliser le travail demandé.

Solution 1



Tous les meetings

```
SELECT title, description, date_start as dateStart, date_end as dateEnd FROM meeti
ngs;
```

Tous les meetings d'une communauté

```
SELECT title, description, date_start as dateStart, date_end as dateEnd FROM meeti
ngs WHERE community_id = :communityId;
```

Notez le `:communityId` qui est une variable de prepared statement (pour éviter les injections SQL), voir le top 10 owasp, expliqué par exemple par Gary Hockin à PHPUK (cherchez sur Youtube).

Les utilisateurs et les meetups qu'ils organisent

```
SELECT u.name as user_name, m.id as user_id, m.title as meeting_title, description
 as meeting_description, m.date_start as meeting_date_start, m.date_end as meeting
_date_end, c.name as community_id
FROM users u
inner join organisers o on u.id = o.user_id
inner join meetings m on o.meeting_id = m.id
inner join communities c on c.id = m.community_id;
```

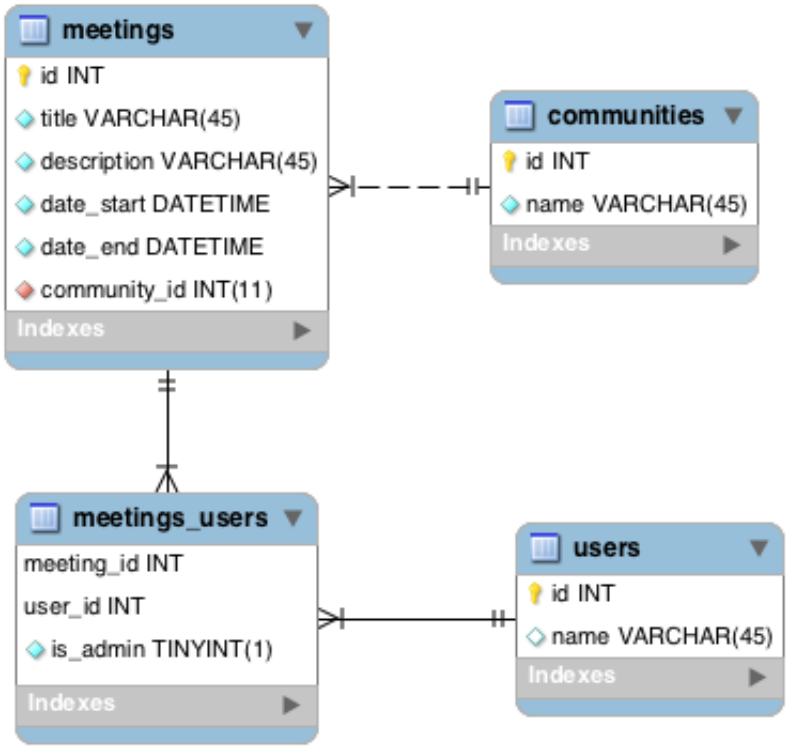
Un utilisateur et ses meetups

```
SELECT u.name as user_name, m.id as user_id, m.title as meeting_title, description
 as meeting_description, m.date_start as meeting_date_start, m.date_end as meeting
_date_end, c.name as community_id, 1 as is_organiser
FROM users u
inner join organisers o on u.id = o.user_id
inner join meetings m on o.meeting_id = m.id
inner join communities c on c.id = m.community_id
WHERE u.id = 1

UNION

SELECT u.name as user_name, m.id as user_id, m.title as meeting_title, description
 as meeting_description, m.date_start as meeting_date_start, m.date_end as meeting
_date_end, c.name as community_id, 0 as is_organiser
FROM users u
inner join attendees a on u.id = a.user_id
inner join meetings m on a.meeting_id = m.id
inner join communities c on c.id = m.community_id
WHERE u.id = 1
```

Solution 2



Inspirez-vous des requêtes ci-dessus (surtout le `0 as is_organiser` dans `Un utilisateur et ses meetups`.

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