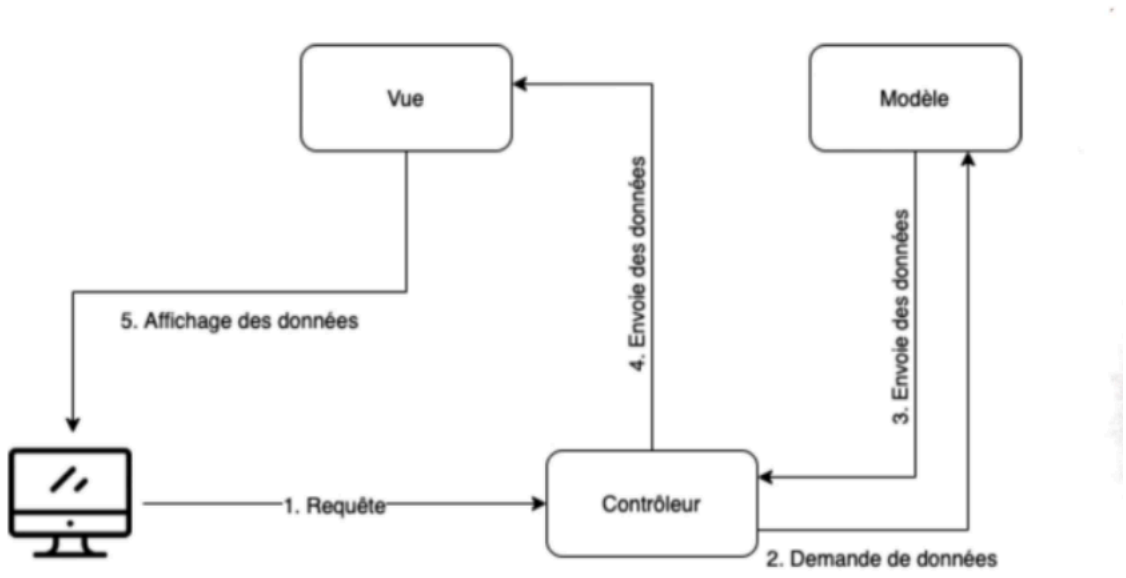
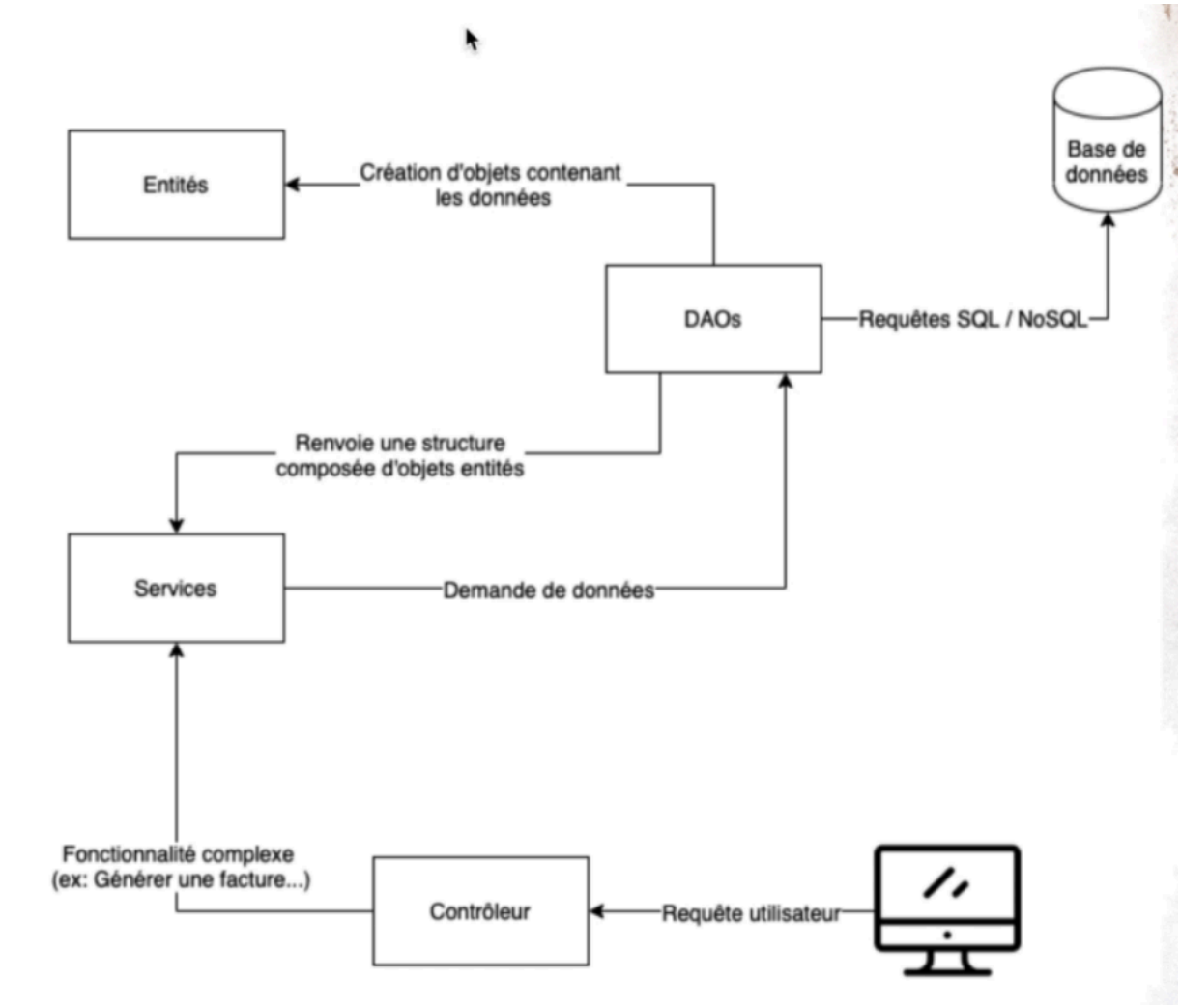
MVC PHP OBJET

Patron de conception ou le modèle est plus détaillé

Vous connaissez le modele MVC suivant:



Le variante de ce modèle va consister à détailler la partie Modele en Services et DAOs
Plus de complexité que le MVC procédural (multiplication importante des fichiers mais l'avantage est la réutilisabilité des composants, les templates et accès DAOs réutilisables.



MODELES

SERVICES

logique métier de l'application (exemple demande de facture) fait appel au composant DAO (couche d'abstraction qui va s'occuper de manipuler la donnée en Base de données , les données sont récupérées en données brute , mais il nous faut des objets , la DAO fait le lien entre les données brutes et les objets dans les **entités** (à chaque table correspondra une entité ou classe)

La couche service est une couche de logique commerciale. C'est ici, et seulement ici, que doivent se trouver les informations relatives au flux des processus métier et à l'interaction entre les modèles métier. Il s'agit d'une couche abstraite et elle sera différente pour chaque application, mais le principe général est l'indépendance de votre source de données (la responsabilité d'un contrôleur) et du stockage des données (la responsabilité d'une couche inférieure).

1. permet la séparation des préoccupations

La couche de service fournit la modularité du code, la logique et les règles commerciales sont spécifiées dans la couche de service qui, à son tour, appelle la couche DAO, la couche DAO est alors seulement responsable de l'interaction avec la base de données.

2. fournit la sécurité

Si vous fournissez une couche de service qui n'a aucune relation avec la base de données, il est plus difficile d'accéder à la base de données depuis le client, sauf par le biais du service. Si le client ne peut pas accéder directement à la base de données (et s'il n'y a pas de module DAO trivial faisant office de service), un attaquant qui a pris le contrôle du client ne peut pas accéder directement à vos données.

3. Fournir un couplage lâche (loose coupling)

La couche service peut également être utilisée pour fournir un couplage lâche dans l'application. Supposons que votre contrôleur possède 50 méthodes et qu'il appelle 20 méthodes DAO, vous décidez plus tard de changer les méthodes DAO qui servent ces contrôleurs. Au lieu de cela, si vous avez 20 méthodes de service appelant ces 20 méthodes Dao, vous devez modifier seulement 20 méthodes de service pour pointer vers un nouveau Dao.

LE CONTROLEUR

. n'affiche rien
. n'accède pas au données directement
. Ne doit pas contenir de logique métier (réservé aux services)

CREATION D'UN PETIT FRAMEWORK MVC OBJET EN PHP

1. création d'un repo git distant exemple: "mvcobjet"
2. clonage de ce repo
3. création controllers / models/ views/ index.php
4. installation de composer

Composer est un logiciel gestionnaire de dépendances libre écrit en PHP. Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin.
c'est l'équivalent de npm en node.js.

1. installation de composer (sur windows) https://getcomposer.org/ https://packagist.org/
2. composer init (permet de créer un fichier composer.json de manière interactive
3. project / no dependence's / no dependence's
4. composer.json est crée avec les librairies.
5. ici composer est intéressant pour gérer les classes à l'intérieur de notre application (autoloader)
6. nommage particulier
7. créer la clé autoload dans composer
8. autoloader fait appel à une norme php qui s'appelle PSR-4
9. spécification pour la standardisation des concepts de programmation php (exigences obligatoires).
10. psr-n correspond à un standard particulier (ici no4)
11. exigences obligatoires à respecter
12. dans cet espace psr-4 je fournis un **namespace**
13. toutes les classes seront "namespacées" avec mvcobjet
14. je met toutes mes classes dans src et je mets mes repertoires controllers/models/views dans src
15. Installation d'un router simple (klein)
16. composer require klein/klein
17. fichier lock (blocage des dépendances) et regénération de autoloader (pour importer klein)

Entités objets de notre base de données conversion des données brutes et entités = role des DAOs

à chaque table correspond une classe
entités : accessors/setters (accéder / modifier)
reflet objet de notre base de données

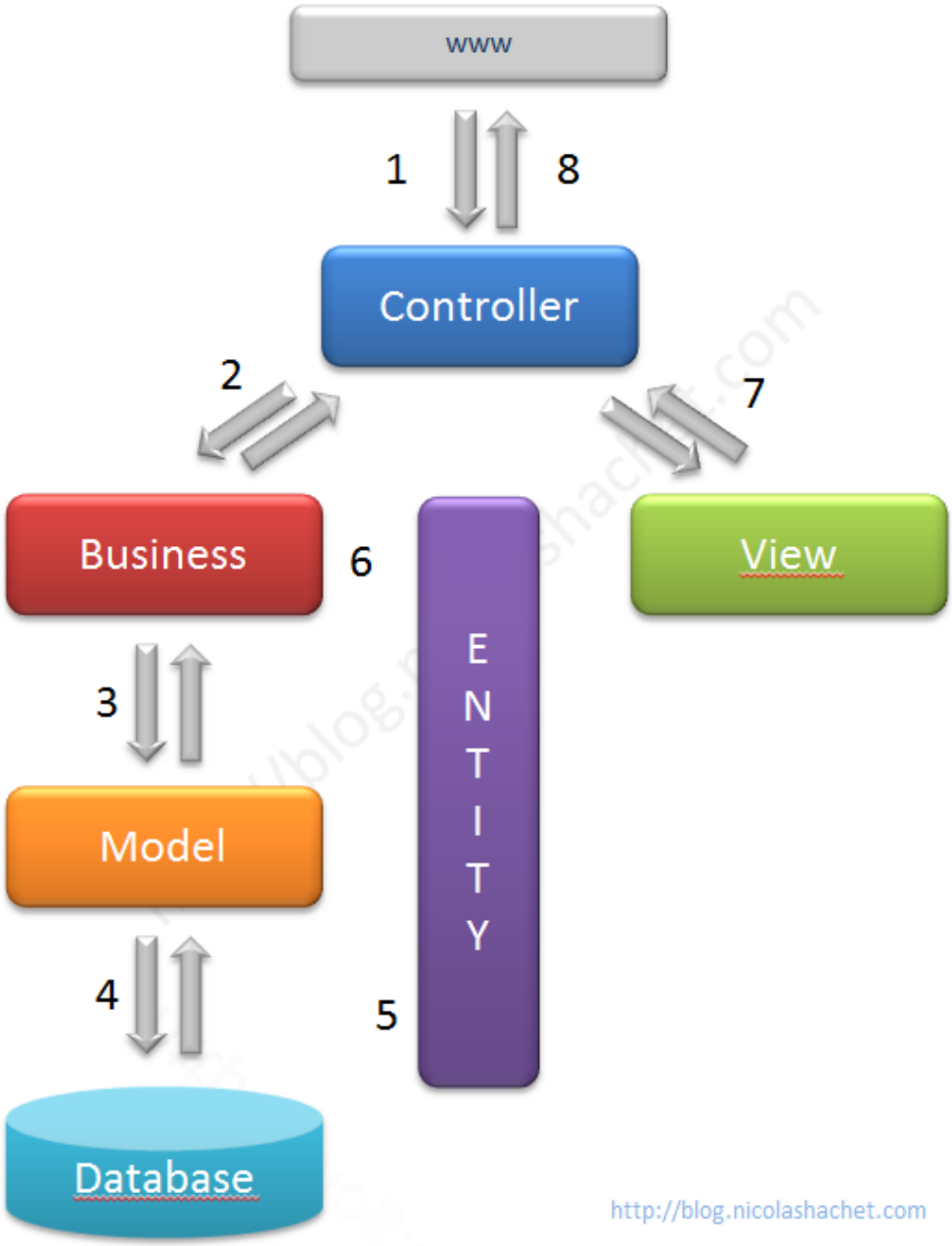
entités : interface claire pour les services et les vues.
à chaque classe correspond une table
dans les entités accesser/setters
dans models-> rajouter entités

DAO -> converti données de la base en entités
on peut changer de stockage, DAO permet la maintenabilité

ça rajoute de la complexité mais rentable à long terme
DAO pont entre BDD et Entités
à chaque entité correspond une DAO

Pour le test nous allons utiliser les DAO directement dans le frontcontroller.

use mvcphpobjet\Models\Daos\xxxxDAO
use mvcphpobjet\Models\Entites\Actors



En **PHP**, la **couche métier (business)** est une **évolution du modèle MVC qui dissocie la logique technique et la logique fonctionnelle** (ie, les classes métier). Elle est parfois appelée, à tord à mon sens, **couche service.** Pour moi, les services peuvent être *techniques* ou *fonctionnels* et je préfère donc l'appellation de **couche métier** ou **service métier**. L'objectif de cette couche supplémentaire est de stocker le code métier de l'application en donnant au contrôleur un rôle de **coordination**.

Représentation d'une couche métier

Le placement de la couche métier (ici en rouge, nommée **Business**) se situe entre la couche modèle et la couche contrôleur du modèle MVC. Elle se compose des **classes métiers** qui contiennent la logique applicative.

Fonctionnement théorique

1. Une requête arrive sur l'action du contrôleur.
2. Le contrôleur appelle le **service métier** qui réalise les traitements associés à l'action. Ce service métier pourra appelé N autre service selon ses besoins.
3. Un service métier peut appeler (ou non) la **couche modèle** si des interactions doivent être effectuées avec la base de données (récupération de données, mise à jour). Il peut également appelé des services techniques (ex : service d'envoi de mails).
4. La couche modèle interagit avec la **base de données** par l'intermédiaire d'une couche d'abstraction plus ou moins grande (ex : PDO pour être près de la base ; Doctrine ORM pour être plus abstrait).
5. L'application gère les données issues de la base de données via les notions **d'objets** (appelée entités en Symfony / Zend) ou de tableaux (par défaut en CakePHP).
6. Une fois remonté jusqu'au contrôleur, celui-ci appelle une **vue** pour l'affichage. La vue peut utiliser les données issues de la base, toujours par l'intermédiaire des entités (ou des tableaux en CakePHP). La vue peut être au format HTML, JSON, XML, etc.
7. La réponse est envoyée au client.

Un peu de concret

Bon la théorie c'est bien mais avec un exemple on comprend mieux. Je vais utiliser le classique du produit ajouté au panier… Oui, c'est dans les vieux pots qu'on fait les meilleures crèmes !

1. Un client demande à ajouter un produit à un panier (exemple de requête : POST / paniers/produit/123
2. Le contrôleur appelle le service métier « panier > ajouter un produit par référence ». Le service métier panier fait appel au service métier « produit » pour savoir si le produit est en stock.
3. Le service métier « produit » demande au modèle « produit » de vérifier si le produit est dispo.
4. Le modèle « produit » fait sa requête en base de données et retourne le nombre de produits dispos.
5. L'application renvoi le nombre de produit dispo.
6. Une fois la vérification effectuée, le service panier fait appel au modèle « panier » pour ajouter le produit en base.
7. Si tout s'est bien passé (pas d'exception), le contrôleur récupère une vue JSON qui permettra de confirmer l'ajout du produit au client : « {'msg' : 'Ok votre produit est bien ajouté au panier'} » .
8. La réponse est envoyée au client « HTTP 200 OK ».

Notez bien que les services métiers ne correspondent pas forcément à des modèles physiques en base.

Quelques règles d'écriture

L'écriture d'une couche métier suit plusieurs règles :

- garder à l'esprit que cette couche est **purement métier**, il est donc déconseillé d'interagir directement avec le *framework* utilisé. Dans l'absolu, il devrait être possible de **changer de framework sans changer une ligne de code de la couche métier.** L'*injection de dépendances* est une réponse à cette problématique **;**
- la couche métier est organisée en **service métier** qui peuvent s'appeler les uns les autres, il est également possible d'appeler des **services techniques** (ex : envoi de mails, logs, etc.) ;
- les exceptions PHP apportent de la souplesse à la gestion des erreurs.

TWIG

Twig est un langage de modélisation qui se compile en code PHP optimisé. Il est principalement utilisé pour produire du HTML, mais peut également être utilisé pour produire tout autre format de texte. Il s'agit d'un composant autonome qui peut être facilement intégré à tout projet PHP.

accede aux variables

```
<!DOCTYPE html>
<html>
  <body>
    <span>Hello {{ name }}</span>
  </body>
</html>
```

```
accède aux variables d’un tableau
```

```
<!DOCTYPE html>
<html>
  <body>
    <span>Hello {{ user['name'] }}</span>
  </body>
</html>
```

```
Accède aux propriété d’un objet
```

```
<!DOCTYPE html>
<html>
  <body>
    <span>Hello {{ user.name }}</span>
  </body>
</html>
```

```
boucles conditionnelles
```

```
{% if temperature < 10 %}
    It's cold
{% elseif temperature < 18 %}
    It's chilly
{% elseif temperature < 24 %}
    It's warm
{% elseif temperature < 32 %}
    It's hot
{% else %}
```

```
    It's very hot
{% endif %}
```

Filtres

```
{{ array['key'] | upper }} {{ object.text | upper }}


{% set array = "3,1,2"| split(',') %}

{{ array | sort | first }}


{% for current in array %}
    <h1>This is number {{ current }} in the array </h1>
{% endear %}
```

Récupération d'un objet movie.
L'objet est construit à partir des requêtes DAO dans chaque table.
Il n'y a pas un SQL qui renvoi toutes les information mais seulement les requêtes nécessaires.

Objet movie composé des ses attributs et de références sur d'autres objets.

```
object(mvcobjet\Models\Entities\Movie)#97 (9) {
   ["id":"mvcobjet\Models\Entities\Movie":private]=>string(1) "1"
   ["title":"mvcobjet\Models\Entities\Movie":private]=>string(11) "Forest Gump"
   ["description":"mvcobjet\Models\Entities\Movie":private]=>string(236) "The presidencies of
Kennedy and Johnson, the events of Vietnam, Watergate, and other historical events unfold through
the perspective of an Alabama man with an IQ of 75, whose only desire is to be reunited with his
childhood sweetheart."
   ["duration":"mvcobjet\Models\Entities\Movie":private]=>string(4) "2h22"
```

réference sur objet DateTime

```
   ["date":"mvcobjet\Models\Entities\Movie":private]=>

  object(DateTime)#98 (3) {
    ["date"]=>
    string(26) "1994-10-05 14:18:47.000000"
    ["timezone_type"]=>
    int(3)
    ["timezone"]=>
    string(13) "Europe/Berlin"
  }

  ["coverImage":"mvcobjet\Models\Entities\Movie":private]=>
  string(31) "https://i.imgur.com/6vy8xeA.png"
```

réference sur 1 objet Genre

```
  ["genre":"mvcobjet\Models\Entities\Movie":private]=>
  object(mvcobjet\Models\Entities\Genre)#101 (2) {
    ["id":"mvcobjet\Models\Entities\Genre":private]=>
    string(1) "1"
    ["name":"mvcobjet\Models\Entities\Genre":private]=>
    string(5) "Drame"
  }
```

réference sur 1 objet Réalisateur

```
  ["director":"mvcobjet\Models\Entities\Movie":private]=>
  object(mvcobjet\Models\Entities\Director)#102 (3) {
    ["id":"mvcobjet\Models\Entities\Director":private]=>
    string(1) "2"
    ["firstName":"mvcobjet\Models\Entities\Director":private]=>
    string(6) "Robert"
    ["lastName":"mvcobjet\Models\Entities\Director":private]=>
    string(8) "Zemeckis"
  }
```

réference sur 1 tableau d'acteurs

```
  ["actors":"mvcobjet\Models\Entities\Movie":private]=>
  array(2) {
    [0]=>
```

référence sur Objet acteur 1

```
    object(mvcobjet\Models\Entities\Actor)#99 (3) {
      ["id":"mvcobjet\Models\Entities\Actor":private]=>
      string(1) "1"
      ["firstName":"mvcobjet\Models\Entities\Actor":private]=>
```

```
      string(3) "Tom"
      ["lastName":"mvcobjet\Models\Entities\Actor":private]=>
      string(5) "Hanks"
    }

   [1]=>
```

référence sur Objet acteur 2

```
   object(mvcobjet\Models\Entities\Actor)#100 (3) {
      ["id":"mvcobjet\Models\Entities\Actor":private]=>
      string(1) "7"
      ["firstName":"mvcobjet\Models\Entities\Actor":private]=>
      string(3) "Tim"
      ["lastName":"mvcobjet\Models\Entities\Actor":private]=>
      string(7) "Robbins"
    }
  }
}
```