

Evaluation en Orienté Objet en PHP et Laravel 5



**3W Academy - Promotion Lyon L6**

Évaluation de connaissances Laravel & POO

Lundi 28 septembre 2015 / 14h00-18h00

NOM : LESNE	Matière : Framework Laravel & POO
Prénom : Matthieu	Cours de : BOYER Julien
	Durée de l'épreuve : 4 heures

Appréciations	Note sur 20
De très bonnes et solides connaissances du framework dans son ensemble ! Prêt à affronter Symfony 2:)	16



Lisez attentivement le sujet et respectez soigneusement les modalités à suivre. Bien entendu, tout étudiant suspecté d'avoir copié ou de s'être inspiré de son voisin sera sanctionné.

Modalités à suivre

- ☐ **Accès aux supports de cours (prises de notes et documents numériques) :** autorisé
- ☐ **Usage des PC :** autorisé
- ☐ **Accès au réseau :** autorisé durant toute l'épreuve
- ☐ **Documents à récupérer sur l'intranet :** aucun
- ☐ **Rendu :** 1 fichier .pdf , .odt ou .doc
- ☐ **Nomenclature du fichier à déposer :** NOM_Prenom_3WA.pdf
- ☐ **Contrôle :** auprès du surveillant pour le sujet

Contexte et But

Ce test de connaissance et maîtrise de Laravel 5 a pour but de valider les connaissances du candidat sur une épreuve d'Orienté Objet en PHP, des concept jusqu'au Design Pattern et implémentation en framework et de bonnes pratiques au célèbre framework Laravel 5.

Recommandations de l'intervenant

Les réponses doivent être claires, concises et précises accompagnées d'exemple, dournis avec des bout de codes, des screenshots (Pertinence et qualité de la réponse du candidat, Argumentation et démonstration des réponses, Apport d'exemples, de citations/comparaisons, de codes sources etc...)

Quelques sources :

- + <http://laravel.com/docs/master/>
- + <http://cheats.jesse-obrien.ca/>
- + <http://creative-punch.net/wp-content/uploads/2013/12/cheat-sheet-A4.pdf>
- + <http://www.cheatography.com/bernattorras/cheat-sheets/laravel/>
- + <https://openclassrooms.com/courses/decouvrez-le-framework-php-laravel>

Livrables attendus

- Un fichier .pdf ou .doc est à rendre à l'intervenant.

Partie 1 Questions Générales en Orienté Objet & Design Pattern (30 points)

1. Quelle est l'utilité du « Front-Contrôleur » dans un CMS ou dans un framework ?

Le Front controller sert à réceptionner toutes les requêtes entrantes pour ensuite les répartir entre les différents contrôleurs de l'application.

Le fichier index.php est un fichier « entonnoir » où la Requête est réceptionnée et où la Réponse est envoyée. (traitement du protocole HTTP)

Il est utilisé dans la technologie Built-In-Server (serveur virtuel lancé en CLI depuis la version de PHP) quand on lance « php artisan serve »

2. Qu'est-ce qu'un namespace en PHP ?

Un namespace (ou espace de nom) permet de définir l'emplacement d'un fichier PHP. Cela permet d'éviter les conflits voire d'utiliser plusieurs fois les mêmes noms de classes.

Il est à déclarer au tout début du fichier PHP avant une classe et doit respecter la PSR-0 avec noms des dossiers et sous-dossiers nom du namespace et sousnamespace. Il est également chargé par Composer dans autoload_namespaces.php

Exemple :

```
namespace App\Console\Commands;
```

3. Quels sont les avantages d'un Moteur de Template ? (Citez un exemple)

Les templates permettent de séparer la présentation (HTML) du code de l'application (PHP). Cependant, il est toujours nécessaire dans le HTML d'utiliser des fonctions dynamiques pour l'affichage : conditions, boucles...

Les moteurs de templates visent à rendre l'écriture de ces fonctions plus simples et lisibles qu'en utilisant du PHP pur. Des fonctionnalités supplémentaires y sont généralement ajoutées, comme la possibilité d'hériter des templates, ou de sécuriser les variables utilisées.

Il apporte la sécurité en échappant chaque variable affichée, la lisibilité quand à l'intégration avec un pseudo langage et la performance car les vues sous moteur de template sont compilées et mises en cache.

Exemples de moteurs de templates : Blade pour Laravel, Twig pour Symfony

4. À quoi sert le cache dans un Framework ?

La mise en cache permet d'améliorer les performances d'affichage des pages web, en réduisant la consommation de bande passante, et les sollicitations du serveur web.

Le cache peut être sous différentes couches :

+ HTTP Cache

+ Cache en Reverse Proxy (Varnish) avec serveur statique

+ Technologies de cache diverses : Memcache, Redis, Compilation PHP etc.

5. Comment personnalise t-on les exceptions en PHP ?

On peut personnaliser une exception en PHP en 3 étapes :

- on la lance (throw) selon certaines conditions
- on essaye d'effectuer nos instructions (try)
- on attrape les exceptions qui seront levées (catch)

On peut également créer ses propres classe d'exception héritant de Exception et capturant par l'objet ces types d'exception personnalisées

Exemple :

```
function compare($a, $b)
{
    if (!is_numeric($a) || !is_numeric($b))
    {
        // Lancement de l'exception :
        throw new Exception('Les paramètres doivent être des chiffres');
    }
    return $a + $b;
}
try // Tentative d'exécution des instructions :
{
    // ...
}
catch (Exception $e) // On attrape les éventuelles exceptions levées
{
    echo 'Exception levée. Message : ', $e->getMessage();
}
```

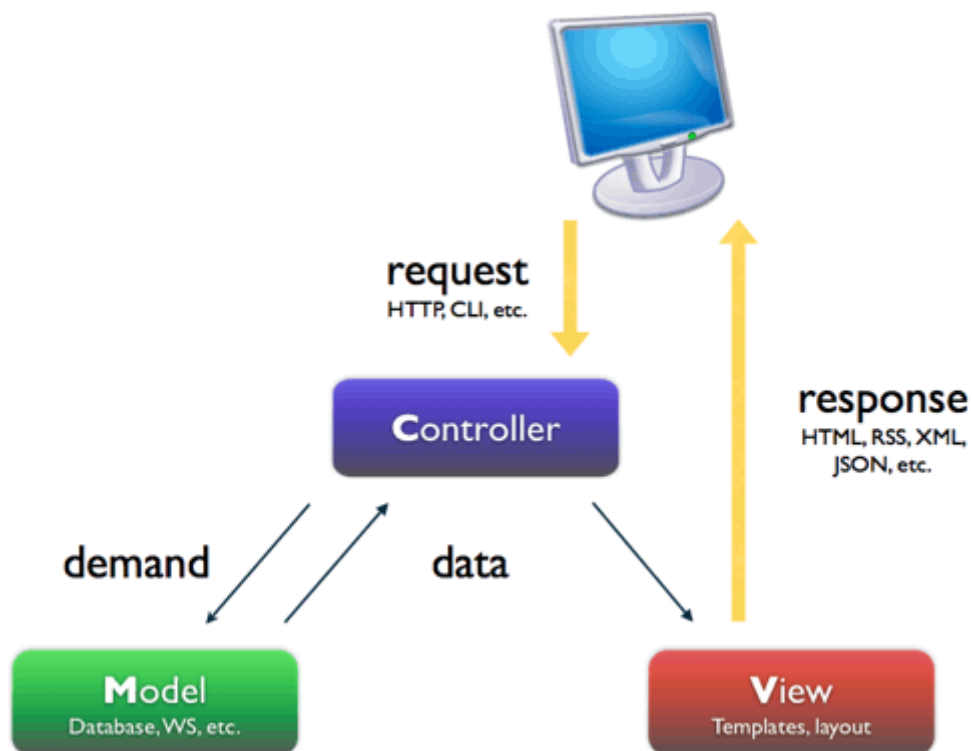
6. Qu'est ce que le MVC ?

C'est un Design Pattern de Structure, couramment utilisée dans certains CMS et framework modernes.

Le MVC (Modèle Vue Contrôleur) est un modèle d'organisation de ses applications en 3 grandes entités :

- les modèles : coeur de l'application, gérant essentiellement les relations avec les bases de données et le traitement des données
- les vues : servent à l'affichage des données et aux interactions avec les utilisateurs.
- les contrôleurs : ils reçoivent les événements des vues et enclenchent les actions à effectuer

L'objectif est de séparer clairement les données, les traitements et la présentation.



7. Quels sont les principaux IDE du marché actuellement ?

Quelques IDE spécialisés PHP parmi les plus connus et utilisés :

- PHPStorm (Sublime Text3)
- Eclipse
- Netbeans
- Geany

8. Que signifie que le PHP est coté serveur (côté « Backend ») ?

Cela signifie que son code s'exécute sur les serveurs (**cote serveur**) qui hébergent les sites web, et non dans le navigateur des clients. Cela implique que chaque action de l'utilisateur nécessite un rafraichissement de la page pour afficher des résultats dans le navigateur. (sauf pour la technologie **AJAX** ou **Websocket**)

9. Pouvez-vous définir en quelques mots l'injection des dépendances ?

Selon Fabien Potencier, créateur et lead developer (+ 1 ponts) du framework PHP Symfony :

Dependency Injection is where components are given their dependencies through their constructors, methods, or directly into fields.

En d'autres termes, si une classe a besoin d'une instance d'une autre classe, on la passe directement en paramètres pour ne pas avoir besoin de l'instancier elle-même. Cela permet d'éviter que les classes soient dépendantes (indépendantes) les unes des autres, donc plus facilement réutilisables et maintenables. (looseless compled)

Si dans la classe A , l'injecte un objet B.

Si plus tard B change de ses paramètres lors de sa construction , A ne sera affecté...

Dans un conteneur d'injection des dépendances, une seule et même instance est injectée ce qui centralise l'injection des dépendances

Exemple :

```
public function __construct(Adresse $adresse)
{
    $this->adresse = $adresse;
}
```

9. Quels sont les avantages de l'« Orienté-Objet » dans un framework ?

- facilité d'organisation (code factorisé en ensembles logiques)
- facilité de réutilisation du code,
- plus intuitif (la notion d'objets se rapprochant du monde « réel »
- possibilité d'héritage,
- plus facile à maintenir, à débiter, à sécuriser que le procédural,
- possibilité de séparer plus facilement le code d'une application pour le partager entre plusieurs développeurs

Meilleure logique (design patterns d'objets et de construction) et meilleur architecture (design pattern de structures) avec une meilleure cohérence et interaction entre les couches du MVC. (séparation de la

logique métier et responsabilités des classes) : Code plus maintenable, plus évolutif, plus organisé, plus scalable etc...

10. Donner une définition de l'Autoload?

En programmation orientée objet, la bonne pratique consiste à créer un fichier par classe (PSR-0). Un inconvénient de cette méthode est de se retrouver avec une longue liste de classes à inclure, surtout dans une application complexe.

Via Composer, les classes se chargent automatiquement et intelligemment, pas besoin de les inclure dans l'ordre.

L'autoload ou « auto-chargement de classes » permet de s'affranchir de cette écriture en appelant automatiquement les classes.

11. Comment fait-on pour accéder aux attributs « protected » d'une classe ?

La visibilité protected permet de ne rendre utilisable un attribut ou une méthode que par la classe que les définit ou par une classe qui étend cette classe.

On peut accéder aux attributs protected en définissant une méthode spéciale nommée « getter » : exemple :

Création de méthodes publiques pour accéder (getter) ou modifier (setter) les attributs protected ou private.

```
class Voiture extends Vehicule
{
    // ...
    public function getMarque()
    {
        return $this->marque;
    }
}
```

12. A quoi sert un constructeur dans une classe ?

Méthode qui initialise le propriété d'un objet ou déclenche des méthodes lors de sa création

Lorsque l'on crée un objet, celui-ci est vide par défaut. Une méthode constructeur est appelée automatiquement à la création d'un objet pour « construire » l'objet, par exemple lui donner des valeurs d'attributs.

Exemple :

```
class User
{
    public function __construct()
    {
        $this->pays = "France";
    }
}
```

13. Qu'est ce qu'une classe abstraite et une classe finale ?

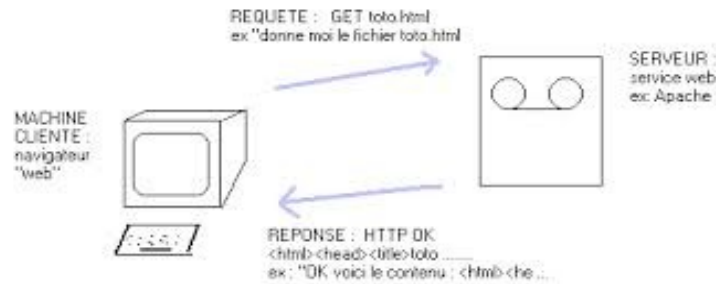
Une classe abstraite ne peut pas créer d'objet. Elle ne sert qu'à être héritée par d'autres classes. Elle peut déclarer des attributs et des méthodes (ces dernières pouvant également être abstraites pour obliger leur définition dans les classes filles)

Une classe finale ne peut pas être dérivée par une sous-classe, ses attributs et méthodes ne peuvent plus être redéfinis.


Une classe abstraite contient au moins une méthode abstraite

14. Donnez une définition du protocole « Requête-Réponse »

Le protocole Requête-Réponse est à la base du protocole HTTP : un client (ex : un internaute) tente de communiquer avec un serveur connecté au réseau en effectuant des requêtes (clic sur des liens, soumission de formulaires...). Le serveur a pour rôle de donner une réponse à ses requêtes.



L'HyperText Transfer Protocol, plus connu sous l'abréviation HTTP — littéralement « protocole de transfert hypertexte » — est un protocole de communication client-serveur développé pour le World Wide Web. HTTPS est la variante du HTTP sécurisée par l'usage des protocoles SSL ou TLS



```
"require": {
    "symfony/validator": "2.1.*",
    "doctrine/dbal": "2.2.*",
    "monolog/monolog": "dev-master",
    "jtreminio/test-extensions": "dev-master"
},
"minimum-stability": "dev",
"autoload": {
    "psr-0": {
        "Brony": "src/",
        "Brony\\Tests": "tests/"
    }
}
```

Dependency Manager for PHP

15. A quoi sert la librairie «Composer» ?

Composer est un gestionnaire de dépendance, c'est-à-dire un programme qui permet d'installer rapidement des bibliothèques dans un projet.

Les frameworks PHP Laravel et Symfony l'utilise par exemple pour faciliter l'installation de leurs composants. (Bundles, Providers, Librairies, Components etc.)

Composer est un gestionnaire de dépendances open source écrit en PHP. Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin.

Composer est fortement inspiré de npm pour Node.js et de bundler pour Ruby

16. Quels sont les atouts de la couche « Routing » dans un framework ?

Le routing permet de relier les URLs (les URIs) aux actions des contrôleurs, dans l'objectif de :

- rendre les URL plus lisibles et « propres »,
- rendre la gestion des URL plus configurable et flexible

Elle gère également la sécurité en GET ou POST, les restrictions de paramètres, restrictions sur les nom de domaines spécifiques...Elle lie le traitement de l'Uri à la couche Contrôleurs.

17. Quelle sont les différences entre une classe abstraite et une interface ?

Une classe abstraite et une classe dont toutes les méthodes n'ont pas été implémentées, elle ne peut pas créer d'objet. (contient au moins 1 méthode abstraite et des méthodes publiques, sert uniquement à l'héritage..)

Une interface ressemble à une classe abstraite sauf qu'aucune méthode n'y est implémentée : elles sont seulement déclarées, afin que les classes qui implémentent cette interface les définisse.

(toutes les méthodes sont juste présente de par leurs signatures)

18. A quoi servent les Traits depuis PHP 5.4 ?

Un trait est semblable à une classe, mais il ne sert qu'à grouper des fonctionnalités, il n'est pas possible d'instancier un Trait. Cela permet à des classes d'utiliser ses méthodes sans besoin d'héritage.

C'est un ajout à l'héritage traditionnel, qui autorise la composition horizontale de comportements, c'est à dire l'utilisation de méthodes de classe sans besoin d'héritage. (Beaucoup utilisées dans Laravel)

19. Pourquoi crée t-on des interfaces dans une architecture Orienté Objet?

Oui ...ou pas (je te retire la mention de Fabpot, tu n'es pas digne ... :p :)

La création d'interface dans une architecture framework est avant tout la rigueur et l'homogénéité des classe et de classer les classe par famille et par comportement ou logique métier.

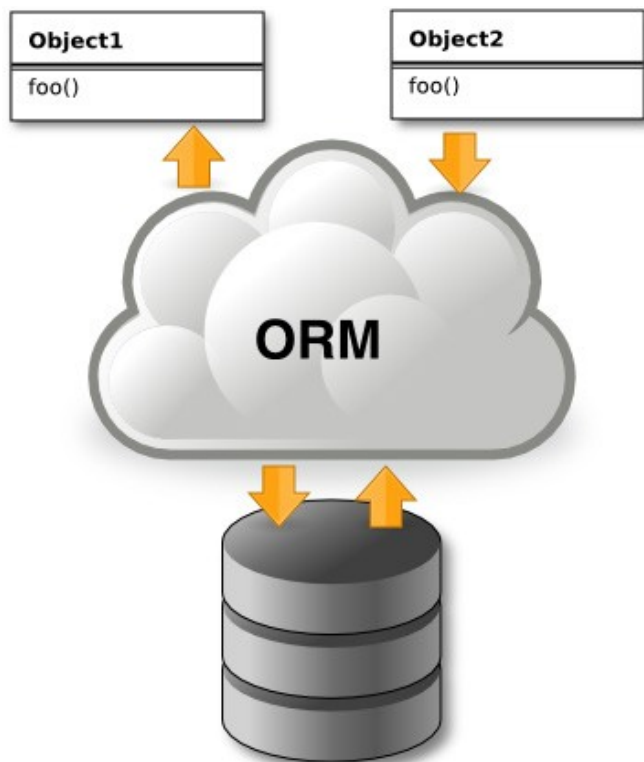
Elle sont omniprésente dans les Design Patterns qui sont implémentés dans le frameworks et laisse place à la souplesse et la scalabilité des classes.

Exemple : sous Laravel 5, cela se traduit directement par des Contrats.

20. Que signifie ORM et quels sont ses rôles ?

Un ORM (object relational mapping) permet d'utiliser une base de données relationnelles dans de la programmation orientée objet. (exemple : Doctrine, Eloquent, Hibernate...https://fr.wikipedia.org/wiki/Mapping_objet-relationnel)

L'ORM permet tout d'abord de simplifier grandement l'accès aux données. Chaque tuple devient une instance d'objet et les méthodes de modification sur les données deviennent uniformisées, on peut effectuer les mêmes traitements sur une donnée qu'elle provienne d'une base de données ou de n'importe quelle autre source.



Un autre avantage de l'ORM est de rendre l'accès aux données complètement indépendant du SGBD utilisé (MYSQL, Oracle, PostGreSQL...). Il devient donc très simple de changer de SGBD au cours du développement de l'application.

Il permet donc de manipuler des objets à partir de bases de données.

(exemples d'ORM : Doctrine pour Symfony, Eloquent pour Laravel)

21. Que est la ligne de commande qu'il faut pour mettre les droit d'écriture et de lecture sur le dossier storage ?

En admettant que l'utilisateur connecté en mode console possède les droits de modifications des permissions, et qu'il souhaite accorder les droits à tous :

```
chmod -R a+rw /storage
```

chmod => commande gestion des droits

-R => fonction récursive pour que les fichiers et dossiers inclus dans /storage soient également affectés par le changement de droit

a+rw: « a » correspond à tous les utilisateurs, « + » à l'ajout de droits, « rw » a « read » et « write »

/storage : dossier cible

CHMOD 777 -R /storage

Principles for maintainable object-oriented code



Single Responsibility Principle

Each class has a single purpose. All its methods should relate to function.

Reasoning: Each responsibility could be a reason to change a class in the future. Fewer responsibilities → fewer opportunities to introduce bugs during changes.

Example: Split formatting & calculating of a report into different classes.



Open / Closed Principle

Classes (or methods) should be open for extension and closed for modification. Once written they should only be touched to fix errors. New functionality should go into new classes that are derived. This is popularly interpreted to advocate inheriting from an abstract base class.

Reasoning: Again you lower the odds of breaking existing code.



Liskov Substitution Principle

You should be able to replace an object with any of its derived classes.

Your code should never have to check which sub-type it's dealing with.

Reasoning: Prevents awkward type checking and weird side-effects.



Interface Segregation Principle

Define subsets of functionality as interfaces.

Reasoning: Small, specific interfaces lead to a more decoupled system than a big general-purpose one.

Example: A PersistenceManager implements DBReader & DBWriter.



Dependency Inversion Principle

High level modules should not depend on low-level modules. Instead, both should depend on abstractions. Abstractions should not depend on details. Details should depend upon abstractions.

Reasoning: High-level modules become more reusable if they are ignorant of low-level module implementation details.

Examples: 1) Dependency Injection. 2) Putting high-level modules in different packages than the low-level modules it uses.

22. Que signifie S.O.L.I.D. en programmation Orienté Objet ?

Expliquez la philosophie de cet acronyme ou chaque lettre suivis d'exemple si possible

SOLID est un acronyme représentant 5 principes de programmation orientée objet, pour créer des applications plus fiables.

Single Responsibility : une classe doit avoir une seule responsabilité. Ceci afin d'éviter de lier les responsabilités entre elles et donc de rendre le code plus rigide

Open/Closed : une classe doit être ouverte à l'extension mais fermée à la modifications

Liskov Substitution : chaque sous classe doit pouvoir être substituée à leur classe de base, sans problème pour l'application

Interface segregation : préférer plusieurs interfaces spécifique plutôt qu'une seule grosse

Dependency Inversion : les modèles de haut niveau (le « coeur » des applications) ne doivent pas dépendre de modules de plus bas niveau, afin que des modifications dans ces derniers ne puissent pas avoir de répercussion sur les modules de haut niveau.

A LIRE ATTENTIVEMENT...

<http://code.tutsplus.com/fr/tutorials/solid-part-1-the-single-responsibility-principle--net-36074>

<http://code.tutsplus.com/tutorials/solid-part-3-liskov-substitution-interface-segregation-principles--net-36710>

<http://code.tutsplus.com/tutorials/solid-part-2-the-openclosed-principle--net-36600>

<http://code.tutsplus.com/tutorials/solid-part-4-the-dependency-inversion-principle--net-36872>

```
class Book {  
    function getTitle() {  
        return "A Great Book";  
    }  
  
    function getAuthor() {  
        return "John Doe";  
    }  
  
    function turnPage() {  
        // pointer to next page  
    }  
  
    function printCurrentPage() {  
        echo "current page content";  
    }  
}
```

Mélanger la logique et la représentation est mauvais cela va à l'encontre du principe d'unique responsabilité. Voyons l'exemple de code suivant:

```

class Book {
    function getTitle() {
        return "A Great Book";
    }

    function getAuthor() {
        return "John Doe";
    }

    function turnPage() {
        // pointer to next page
    }

    function getCurrentPage() {
        return "current page content";
    }
}

interface Printer {
    function printPage($page);
}

class PlainTextPrinter implements Printer {
    function printPage($page) {
        echo $page;
    }
}

class HtmlPrinter implements Printer {
    function printPage($page) {
        echo '<div style="single-page">' . $page . '</div>';
    }
}

```

Même cet exemple très basique montre que la séparation de la présentation et la logique, respectant le principe d'unique responsabilité, donnent de grand avantages dans la flexibilité de notre architecture.

De manière générale, la valeur principale d'un logiciel **est sa capacité à être modifié**.

La seconde réside **en ses fonctionnalités**.

Cependant, pour **avoir de bonnes fonctionnalités, la modification doit être facile**.

Pour cela, nous devons avoir une architecture qui nous permet de modifier, d'adapter ou de compléter les fonctionnalités.

C'est pour cela que le principe d'unique responsabilité doit être respecté.

Nous pouvons raisonner point par point :

1. Une haute capacité à changer conduit avec le temps à de bonnes fonctionnalités.
2. De bonnes fonctionnalités nécessite de comprendre les besoins des utilisateurs.
3. Les besoins des utilisateurs conduisent aux besoins des acteurs.
4. Les besoins des acteurs conduisent aux raisons de changer de ces acteurs.
5. Les raisons de changer définissent les responsabilités.

Donc, lorsque nous concevons un logiciel, nous devons :

1. Trouver et définir les acteurs.
2. Identifier leurs responsabilités.
3. Grouper nos fonctions et nos classes de façon que chacune n'ai qu'une seule responsabilité.

```
class Book {  
  
    function getTitle() {  
        return "A Great Book";  
    }  
  
    function getAuthor() {  
        return "John Doe";  
    }  
  
    function turnPage() {  
        // pointer to next page  
    }  
  
    function getCurrentPage() {  
        return "current page content";  
    }  
  
    function getLocation() {  
        // returns the position in the library  
        // ie. shelf number & room number  
    }  
}
```

Toutes les méthodes de la classe `Book` concerne la logique d'utilisation du livre. Donc nous devons nous placer dans le contexte d'une utilisation du livre. Si notre application était écrite pour être utilisée par de vrais libraires qui cherchent des livres et qui nous donnent un exemplaire physique du livre, alors le principe d'unique responsabilité ne serait pas respecté.

Nous pouvons nous dire que les acteurs "opérations" sont ceux qui utiliseront les méthodes `getTitle()`, `getAuthor()` et `getLocation()`. Peut-être que les clients de la librairie ont aussi accès à l'application qui permet de choisir un livre. Donc les acteurs "lecteurs" seront intéressés par toutes les méthodes sauf `getLocation()`. Un client ordinaire demandera un livre sans chercher à comprendre où il se trouve et le libraire lui remettra. Nous avons donc une violation du principe d'unique responsabilité.

Open / Closed : Ouvert à l'extension mais fermé à la modification, L'idée est qu'une fois qu'une classe a été approuvée via des revues de code, des tests unitaires et d'autres procédures de qualification, elle ne doit plus être modifiée mais seulement étendue.

En pratique, le principe ouvert/fermé oblige à faire bon usage de l'abstraction et du polymorphisme.

Exemple : Le Design Pattern Stratégie :

Dans une application **orientée objet**, lorsqu'une nouvelle tâche est identifiée, la réaction du programmeur est habituellement de créer une classe qui représente cette tâche. Cependant, il se peut que la classe évolue, ce qui génère de nombreuses sous-tâches, chacune d'elles rendant la tâche d'origine plus complexe. Parfois, ce type de sous-tâches peut avoir de nombreux rôles différents. Un exemple typique est une classe logger (1) dont le rôle est habituellement d'écrire un message dans un fichier. Dans un premier temps, cette classe pourrait être enrichie, permettant l'écriture sur de nombreuses autres cibles : fichier et base de données. L'ajout d'une fonctionnalité pourrait permettre au message en question d'être formaté de différentes façons : chaîne de caractères, XML, données sérialisées voire HTML.

Désormais, notre **classe** comprend une tâche basique, l'action de logger. Elle inclut également deux sous-tâches distinctes : formater et écrire. Chaque sous-tâche possède de nombreuses fonctionnalités ayant chacune leur importance. Le pattern strategy suggère une façon d'implémenter une structure de classe flexible qui permet aux programmeurs de permettre une extension aisée de la classe, évitant ainsi quelques pièges habituels tout en conservant cette flexibilité.

En outre, le pattern strategy démontre que, bien souvent, la composition est plus puissante que le simple héritage.

Liskov Substitution :

Subtypes must be substitutable for their base types.

Child classes should never break the parent class' type definitions.

```
class Vehicle {  
  
  function startEngine() {  
    // Default engine start functionality  
  }  
  
  function accelerate() {  
    // Default acceleration functionality  
  }  
}
```

```
}  
}
```

```
class Car extends Vehicle {  
  
    function startEngine() {  
        $this->engageIgnition();  
        parent::startEngine();  
    }  
  
    private function engageIgnition() {  
        // Ignition procedure  
    }  
}  
  
class ElectricBus extends Vehicle {  
  
    function accelerate() {  
        $this->increaseVoltage();  
        $this->connectIndividualEngines();  
    }  
  
    private function increaseVoltage() {  
        // Electric logic  
    }  
  
    private function connectIndividualEngines() {  
        // Connection logic  
    }  
}
```

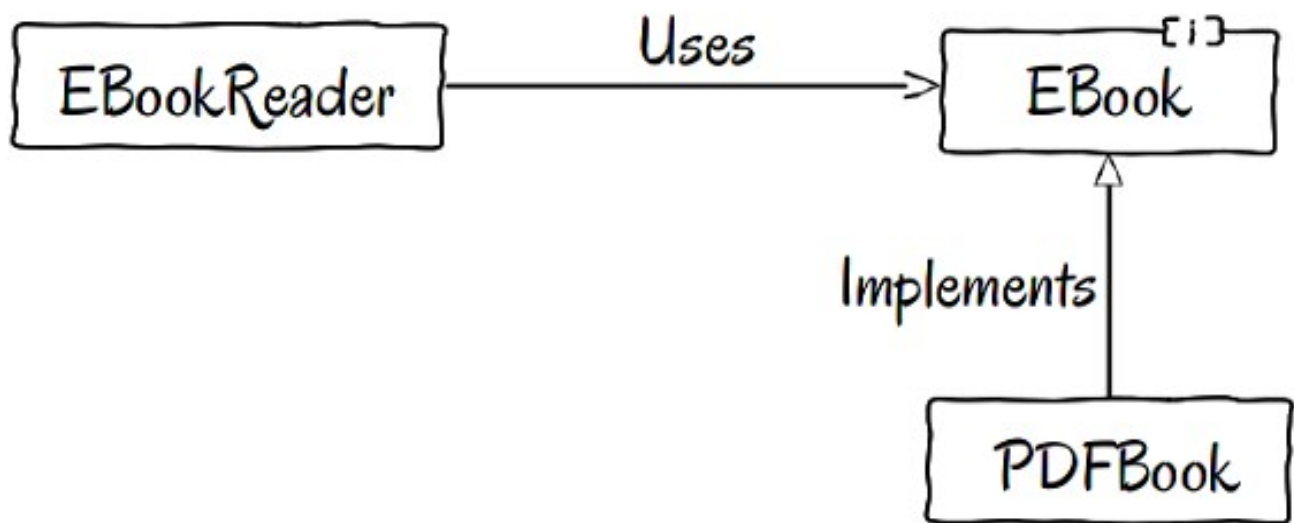
A client class should be able to use either of them, if it can use `Vehicle`.

```
class Driver {  
    function go(Vehicle $v) {  
        $v->startEngine();  
        $v->accelerate();  
    }  
}
```

The interface-segregation principle (ISP) states that no client should be forced to depend on methods it does not use.

A huge `Car` or `Bus` class implementing all the methods on the `Vehicle` interface. Only the sheer dimensions of such classes should tell us to avoid them at all costs.

- Or, many small classes like `LightsControl`, `SpeedControl`, or `RadioCD` which are all implementing the whole interface but actually providing something useful only for the parts they implement.



Dependency Injection

```
class Test extends PHPUnit_Framework_TestCase {

    function testItCanReadAPDFBook() {
        $b = new PDFBook();
        $r = new EBookReader($b);

        $this->assertRegExp('/pdf book/', $r->read());
    }
}

interface EBook {
    function read();
}

class EBookReader {

    private $book;

    function __construct(EBook $book) {
        $this->book = $book;
    }

    function read() {
        return $this->book->read();
    }
}
```

```

}

class PDFBook implements EBook{

    function read() {
        return "reading a pdf book.";
    }
}

class Test extends PHPUnit_Framework_TestCase {

    function testItCanReadAPDFBook() {
        $b = new PDFBook();
        $r = new EBookReader($b);

        $this->assertRegExp('/pdf book/', $r->read());
    }

    function testItCanReadAMobiBook() {
        $b = new MobiBook();
        $r = new EBookReader($b);

        $this->assertRegExp('/mobi book/', $r->read());
    }
}

interface EBook {
    function read();
}

class EBookReader {

    private $book;

    function __construct(EBook $book) {
        $this->book = $book;
    }

    function read() {
        return $this->book->read();
    }
}

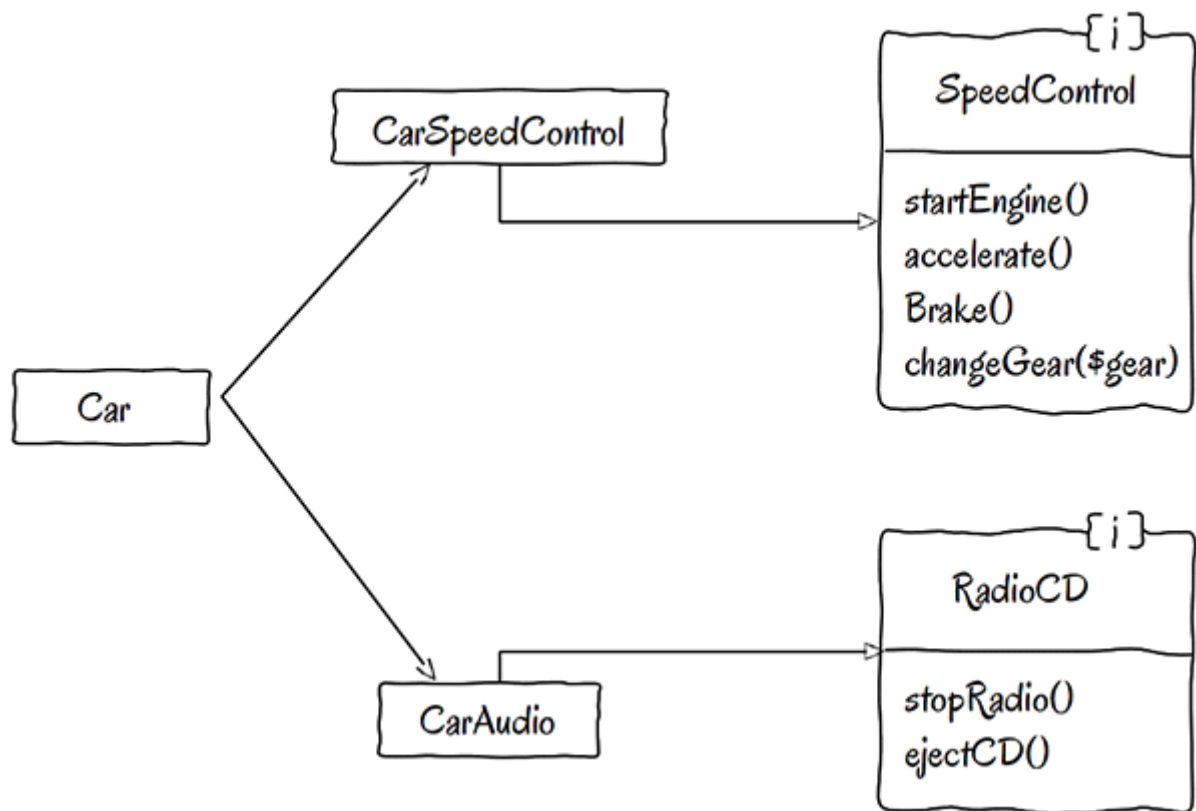
class PDFBook implements EBook {

    function read() {
        return "reading a pdf book.";
    }
}

class MobiBook implements EBook {

```

```
function read() {
    return "reading a mobi book.";
}
}
```



Qu'est ce que le type hinting dans les méthodes d'une classe ?

Cela marche dans l'injection de dépendances pour les Array ou Classe en PHP et même pour les Double ou Int en PHP 7

Cela consiste à préciser le type de données attendu dans les paramètres de la méthode :

exemple :

```
// Le premier paramètre de la méthode store doit être un array
public function store(array $array) {
    echo $array;
}
```

23. Pourquoi il vaut mieux injecter un objet en paramètres de méthode de classe ?

L'utilisation de l'injection de dépendance permet d'éviter que les classes soient dépendantes les unes des

autres, donc plus facilement réutilisables et maintenables.

Cela est plus souple si je change des paramètres dans mon objet injecté ou que je lui ajoute des méthodes/attributs.

<https://openclassrooms.com/courses/programmez-en-orienté-objet-en-php/les-design-patterns>

24. Quel est l'intérêt du Design Pattern Abstract Factory ? Décrivez-le...

Factory Method :

Ce design pattern Factory Method est appelé en français fabrique; c'est un design pattern dit « de création » puisque son but est de créer (jusque là, vous suivez...). Factory est **utilisé pour réduire le couplage entre les classes**; son but est qu'une classe cliente ne fasse plus des instantiations elle-même comme dans le code ci-dessous :

```
class Singe {}
class Girafe {}
class Lion {}

class Zoo {
    public function __construct() {
        $singe = new Singe;
        $girafe = new Girafe;
        $lion = new Lion;
    }
}
```

```
$zoo = new Zoo;
```

Ici Zoo est très **fortement couplée avec Singe, Girafe et Lion** puisque les instantiations sont faites directement par cette classe (dans le constructeur). Nous allons rendre beaucoup plus lâche ce couplage en faisant appel à notre Fabrique :

```
class Singe {}
class Lion {}

class Zoo {
    public function __construct() {
        $fabriqueSinges = new FabriqueDeSinges;
        $singe = $fabriqueSinges->fabriquerAnimal();
        // idem pour les lions et les girafes...
    }
}

abstract class FabriqueDAnimaux {
    abstract public function fabriquerAnimal();
}

class FabriqueDeSinges extends FabriqueDAnimaux {
    public function fabriquerAnimal() {
        $classeCible = 'Singe';

        if (class_exists($classeCible)) {
            return new $classeCible;
        }
    }
}
```

```

}

class FabriqueDeLions extends FabriqueDAnimaux {
    public function fabriquerAnimal() {
        $classeCible = 'Lion';

        if (class_exists($classeCible)) {
            return new $classeCible;
        }
    }
}

```

\$zoo = new Zoo;

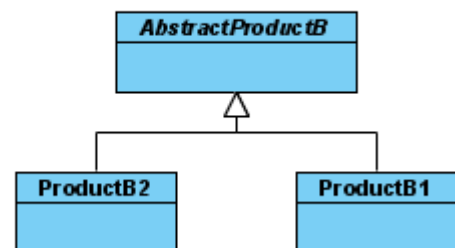
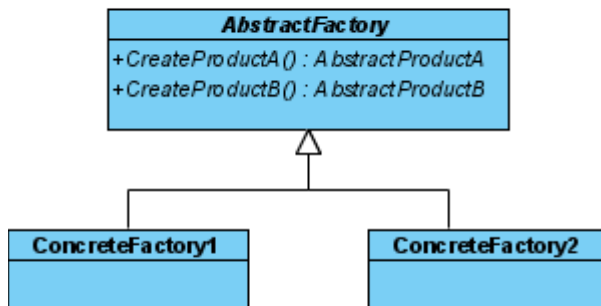
La classe cliente : Zoo

La fabrique abstraite : FabriqueDAnimaux

Les fabriques concrètes : FabriqueDeSinges, FabriqueDeLions, FabriqueDeGirafes...

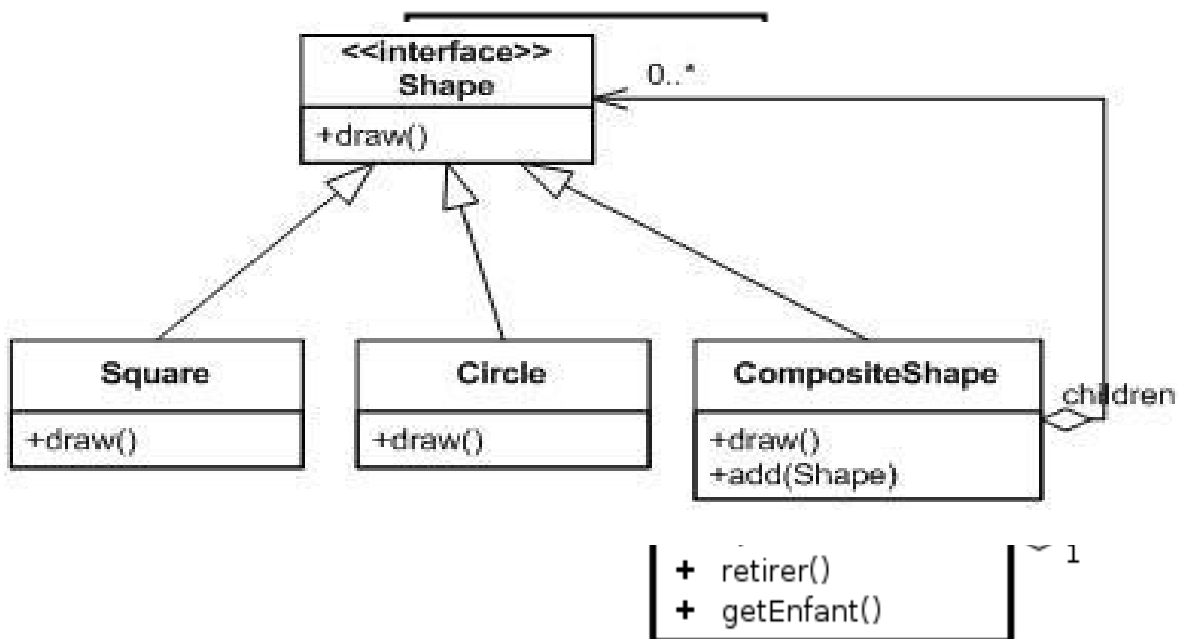
Les classes « produits » (ce qu'on recherche) : Singe, Lion, Girafe

Ce design pattern permet de créer des familles d'objets liés ou inter-dépendants sans avoir à préciser au moment de leur création la classe concrète à utiliser



25. **Quel est l'intérêt du Design Pattern Composite ?** Décrivez-le ...

Il permet de manipuler un groupe d'objets de la même façon que s'il s'agissait d'un seul objet.



La classe « composant » représente l'abstraction pour tous les composants.

La classe « feuille » implémente le comportement par défaut, elle représente un composant qui n'a pas de sous-élément

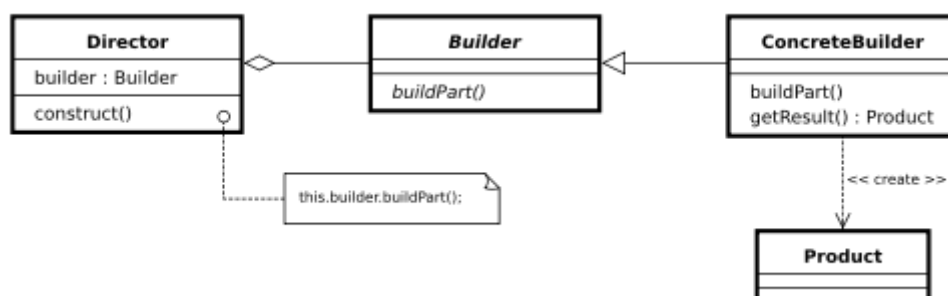
Enfin, la classe composite stocke des composants « enfants ».

Composer des objets et sous objets en liens parentals et passer par une classe abstraites et classe Parent pour déléguer la construction des sous objets dans les classes filles

26. **Quel est l'intérêt du Design Pattern Builder?** Décrivez-le ...

Il permet de monter des objets complexes à partir d'objets sources (ex : pour monter un objet « voiture », on va assembler des objets « roues », « portes », « moteur »...).

Classe pour assembler les différentes partis d'un objet à partir d'autre classes afin de préparer un objet concret dans une application.



La classe « Builder » est la classe abstraite

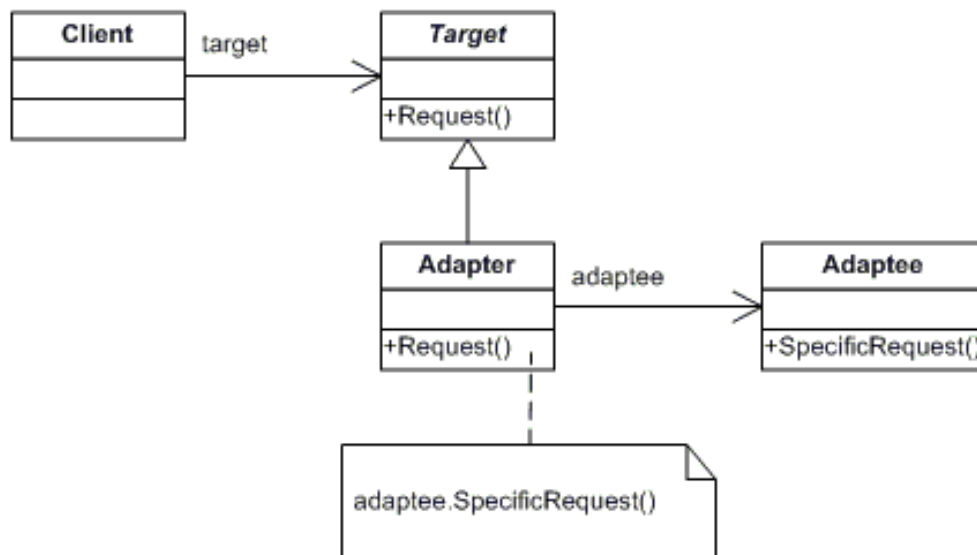
La classe ConcreteBuilder fournit une implémentation de « Builder », elle construit et assemble les différentes parties des « Product »

La classe Director construit un objet en utilisant le méthode de « Builder »

27. Quel est l'intérêt du Design Pattern Adapter? Décrivez-le ...

Une classe qui permet d'adapter les méthodes d'une vieille classe à partir des méthode d'une nouvelle classe

Il permet de convertir l'interface d'une classe en une autre interface. Il permet donc d'ajouter de nouvelles méthodes sans toucher aux anciennes (très utilisé par exemple pour connecter l'API de réseaux sociaux sur un site existant, pour assurer la compatibilité lors de la mise à jour de librairies...)



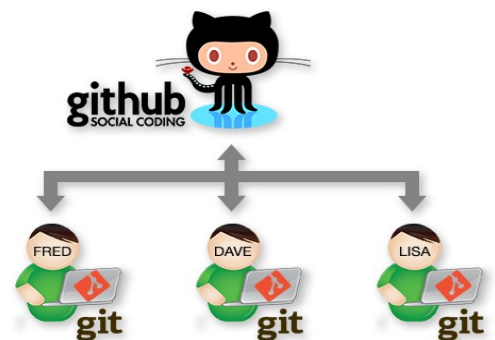
28. Qu'est ce que GIT et Github et comment l'utilise t-on dans un framework?

Git est un logiciel de gestion de versions décentralisé, qui permet de gérer les différentes versions des fichiers d'une application. (Versionning)

Github est un service web (Réseaux Social de Coding) d'hébergement et de gestion de développement de logiciels, utilisant Git.

Git indexe les fichiers d'après leur somme de contrôle calculée avec la fonction de hachage SHA-1. Quand un fichier n'est pas modifié, la somme de contrôle ne change pas et le fichier n'est stocké qu'une seule fois. En revanche, si le fichier est modifié, les deux versions sont stockées sur le disque..

Les serveurs Git utilisent par défaut le port 9418 pour le protocole spécifique à Git. Les protocoles HTTP, HTTPS et SSH (et leurs ports standard) peuvent aussi être utilisés.





Partie 2: Connaissances au framework Laravel 5 (50 points)

1. A quoi sert le dossier app/ dans Laravel ? Le dossier bootstrap ? Et le dossier config ?

- app/ contient les éléments essentiels de l'application : commandes en mode console, événements, exceptions, contrôleurs, routes, middleware, requêtes, providers, **models**...
- bootstrap / contient les scripts de Laravel pour le chargement automatique des classes, la détermination de l'environnement, le démarrage de l'application **avec Composer**
- config/ contient toutes les configurations du framework : application, authentification, cache, base de données, namespace, session...
Il utilise la fonction `env()` et le fichier de config local caché `.env` en utilisant la librairie DotEnv
<https://github.com/bkeepers/dotenv>

Shim to load environment variables from `.env` into ENV in development.

2. Comment affiche t-on toutes les lignes de commandes disponibles sous Laravel ?

En tapant la ligne de commande suivante dans le dossier de l'installation :

php artisan

```
php artisan list
```

3. A quoi sert le fichier database.php contenu dans config/ ? Et le fichier app.php contenu dans ce même dossier?

-database.php sert à configurer la connexion aux bases de données (username, password...), pour les différents systèmes de gestion de base de données supportés (MySQL, SQLite, PostgreSQL, MongoDB...)

database sert aux Migrations (constructions de classe représentant la structure de la base de données)

app.php sert à la configuration globale du projet(encodage, fuseaux horaires, langues, providers, aliases..)

4. Quelle est la ligne de commande pour installer Laravel 5 ? (donnez en 2 et expliquez-les)

Il ya deux méthodes d'installation de Laravel :

1/ En téléchargeant l'installateur de Laravel (via composer en ligne de console)

L'Installer permet d'installer plusieurs instances de Laravel pour plusieurs projets.

```
composer global require "laravel/installer=~1.1"
```

2/ directement via composer

```
composer create-project laravel/laravel --prefer-dist
```

=> va créer directement un projet Laravel. Méthode a priori la plus simple.

5. A quoi sert le fichier .env à la racine ?

Il définit les variables d'environnement globales qui sont les accès SMTP, FTP, BDD etc. accessibles depuis n'importe où dans le projet.

Il sert à définir l'environnement de configuration du framework : nom et informations d'accès à la base de données principale, drivers de cache et session, configuration mails, activation ou non du mode « debug »...

6. Dans quel dossier personnalise t-on toute la configuration d'un projet Laravel et quelle est la fonction de pont avec le fichier .env ?

Le dossier de configuration est le dossier /config situé à la racine.

La fonction `env(MAVARIABLE, VALEURPARDEFAUT)` permet d'aller chercher les variables placées dans le fichier **.env**

7. Comment utilise t-on GULP dans un projet Laravel 5 ? (développement/production)

Laravel propose une API pour définir des tâches Gulp facilement dans le framework, nommé Elixir. Elixir nécessite l'installation de Node.js, **et NPM pour les packages** puis de Gulp, et enfin d'Elixir.

Les tâches Gulp peuvent ensuite être lancées rapidement en console via la commande *gulp* (n.b. *gulp -production* permet en plus de **minifier - concatener - déboguer - compiler (SASS/ Coffeescript)** tous les CSS et Javascript)

La commande *gulp watch* permet de surveiller en temps réel les changements sur les fichiers concernés par Gulp.

Les tâches sont définies dans un fichier `gulpfile.js` à la racine du site. Exemple :

```
// Utilisation d'Elixir
var elixir = require('laravel-elixir');

// Définition des tâches
elixir(function(mix) {
    // Gestion des feuilles de styles
    mix.styles([
        'style.css'
    ]).stylesIn('public/css');
```



```
// Gestion du pré-processeur SASS
mix.sass('*.sass', 'public/css/app.css');

// Gestion des fichiers Javascripts
mix.scripts([
    '*.js'
], 'public/js');
});
```

8. Comment déclare t-on une route en GET qui pointe vers /blog?

Exemple d'une route en GET pointant vers blog, utilisant la méthode index du contrôleur BlogController, avec un alias :

```
Route::get('/blog', ['uses' => 'BlogController@index', 'as' => 'blog.index']);
```

9. A quoi sert de nommer les routes dans un framework ?

C'est utile pour pouvoir les utiliser de façon beaucoup plus simple dans les vues (ou dans une redirection de Controller par exemple.):
exemple :

```
<div data-url="{ route('comments.update', ['id' => $comment->id]) }" ></div>
```

=> la balise div comporte un attribut data-url qui indique une route via son alias « comments.update », avec un paramètre id passé dans un tableau.

10. Écrivez une route sécurisée qui pointe vers la page de recherche « /search/Lyon/69002/ » location avec pour arguments ville, code postal et type de vente ?

```
Route::post('/search/{ville}/{zipcode}/{type}', ['uses' => 'PagesController@search', 'as' => 'search'])->where([
    'ville' => 'Lyon',
    'zipcode' => '69002',
    'type' => '[a-z]+' ]);
```

11. Comment déclarer des routes implicites derrière le contrôleur PhotoContrôleur ?

```
Route::controller('photo', 'PhotoController');
```

Voir <http://laravel.com/docs/4.2/controllers#restful-resource-controllers>

php artisan controller:make PhotoController

```
Route::resource('photo', 'PhotoController');
```

OU

```
Route::resource('photo', 'PhotoController',  
    array('only' => array('index', 'show')));
```

```
Route::resource('photo', 'PhotoController',  
  
    array('except' => array('create', 'store', 'update', 'destroy')));
```

12. Créer un groupe de route sécurisé derrière / admin et une route à l'intérieur qui pointe vers /dashboard pour le contrôleur AdminController ?

```
Route::group(['prefix' => 'admin', 'middleware' => 'auth'], function () {  
    Route::get('/dashboard', ['uses' => 'AdminController@dashboard',  
        'as' => 'admin.dashboard']);  
});
```

13. Pourquoi utiliser un namespace depuis le Routing ? Pouvez-vous en donner un exemple concret

Cela peut servir à assigner le même namespace à un groupe de contrôleurs. (exemple : Pour un backoffice et son ensemble de Contrôleurs dans un sous-namespace « BackOffice »)

Exemple :

```
Route::group(['namespace' => 'User'], function()  
{  
    // contrôleurs à l'intérieur avec le namespace 'User'  
});
```

14. Montrer 3 manières de créer un jeton de sécurité de type CSRF et expliquer à quoi sert ce jeton?

Le jeton CSRF sert à se prémunir contre les attaques de type Cross Sites Request Forgeries, en générant une chaîne de caractère aléatoire qui permettra de vérifier que l'utilisateur authentifié est bien celui qui a effectué une requête.

On peut le générer directement entre des balises PHP :

```
<?php echo csrf_field(); ?>
```

En injectant le token dans un champ de formulaire en vue :

```
<input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
```

En utilisant la syntaxe de du moteur de template Blade :

```
{!! csrf_field() !!}
```

15. Comment créer un contrôleur depuis la ligne de commande ?

Via la commande en console :

```
php artisan make:controller NomDuController
```

16. Comment retrouve t-on depuis le contrôleur l'utilisateur 666 depuis la classe User?

```
class User
{
    public function findUser()
    {
        $user = User::where->('id', 666)->first();
    }
}
```

ou `User::find(666)` ou mieux `User::findOrFail(666)`

17. A quoi sert la ligne suivante dans une action de contrôleur : `$this->middleware('auth');`

A récupérer l'utilisateur identifié

Non, à s'assurer que dans une action de Contrôleur que nous sommes bien authentifié pour exécuter le reste de cette action (c'est un Middleware filtrant la requête)

17. Comment efface t-on le cache en ligne de commande ? Comment génère t-on le cache pour les routes en ligne de console ?

```
php artisan cache:clear
```

```
php artisan route:cache
```

18. Comment déclarer de routes implicites de certaines méthodes de contrôleurs ?

```
Route::resource('photo', 'PhotoController',
    array('only' => array('index', 'show')));
```

```
Route::resource('photo', 'PhotoController',
```

```
array('except' => array('create', 'store', 'update', 'destroy')));
```

19. Comment récupère t-on le paramètre en POST «url» depuis une action de contrôleurs ? Donnez un exemple d'implémentation

```
$name = Input::get('name');
```

En utilisant l'injection dépendance de la classe Request :

exemple :

```
public function store(Request $request)
{
    $url = $request->input('url');
}
```

20. Comment envois t-on des données du Routing au Contrôleur?

On peut passer des paramètres dans la route (en GET comme en POST)

Exemple pour les paramètres « id » et « user » qui seront envoyés au contrôleur « Car »

```
Route::post('/cars/{id}/{user}', ['uses' => 'CarController@store',
'as' => 'car.store'])
```

21. Comment envois t-on des données du Contrôleur à la Vue?

En retournant une variable (ou un tableau) à la vue de la façon suivante :

```
public function create()
{
    // Création de 2 nouveaux objets
    $actors = new Actors();
    $directors = new Directors();
    // Création d'un tableau de données « datas »
    $datas = [
        'actors' => $actors->actors(),
        'directors' => $directors->directors()
    ];
    // Envoi du tableau « datas » à la vue
    return view('Movies/create', $datas);
}
```

22. Comment récupère ton l'URI courante en contrôleur ? Et l'URL complète ?

```
// Récupération de l'URI
$uri = $request->path();

// Récupération de l'URL
$url = $request->url();
```

23. Comment déterminer si un paramètre est présent en POST depuis le Controlleur ?

```
if ($request->has('name')) {  
    //  
}
```

24. Comment récupérer tous les paramètres d'un coup en GET ou POST ?

```
$input = $request->all();
```

25. Comment depuis le controlleur savoir si je suis en GET ou en POST ? Comment sait-on depuis le controlleur si je suis en Ajax ?

Méthode GET ou POST :

```
// Récupération de la méthode  
$method = $request->method();  
  
// On peut alors effectuer des actions selon la méthode  
if ($request->isMethod('post')) {  
    //  
}
```

Ajax :

```
if (Request::ajax()) {  
    //  
}
```

26. Comment récupérer à la fois le login et l'email en POST depuis le controlleur ?

```
public function user(Request $request)  
{  
    $login = $request->input('login');  
    $email = $request->input('email');  
  
}
```

ou \$input = Input::only('username', 'password');

27. Comment valide-t-on un fichier uploadé depuis le controlleur ?

```
if ($request->file('photo')->isValid())  
{  
    //  
}
```

28. Comment récupère t-on un fichier uploadé depuis le contrôleur ?

```
$file = $request->file('photo');
```

ou `$file = Input::file('photo');`

29. Comment créer un message flash en Laravel et à quoi sert-il ?

Les messages flash servent à afficher un message à l'utilisateur une fois une méthode de contrôleur effectuée.

Il ne s'affiche qu'une seule fois et passe par le mécanisme de Session

```
Session::flash('success', "Votre commentaire a été posté");
```

30. À quoi servent les FormRequest dans Laravel ? Écrivez un FormRequest

Les FormRequest servent à la validation de données de formulaires.

Exemple pour la classe « Movies »

```
class MoviesRequest extends FormRequest
{
    // Fonction pour vérifier si l'utilisateur est authentifié et a le droit de modifier la ressource
    public function authorize()
    {
        return true;
    }

    /**
     * Validation des champs : Retourne un tableau de validation par champ
     * @return array
     */
    public function rules()
    {
        return [
            'type' => 'required',
            'title' => 'required|min:2',
            'synopsis' => 'required|min:10|max:200',
            'image' => 'image',
            'trailer' => 'regex:/<iframe src="http:\/\/"/',
            'duree' => 'integer|max:5',
            'annee' => 'date_format:Y',
        ];
    }

    /**
     * Personnalisation des messages qui s'afficheront lorsque les conditions de validation ne seront pas remplies
     */
    public function messages()
```

```
{
    return [
        'required' => 'Ce champ est obligatoire',
        'min' => 'Ce champ doit faire plus de :min caractères',
        'max' => 'Ce champ doit faire moins de :max caractères',
        'integer' => 'Ce champ doit être un chiffre',
        'regex' => 'Mauvais format, merci d\'enregistrer un :attributs valide',
        'date_format' => 'L\'année doit être au format YYYY',
    ];
}
```

31. Comment retourne t-on une réponse JSON depuis le Contrôleur ? Et une réponse HTML ?

```
// Réponse en JSON
return response()->json(['user' => 'Bob', 'city' => 'Lyon']);

// Réponse HTML
```

32. Comment retourne t-on un fichier à télécharger depuis le Contrôleur ?

```
return response()->download($pathToFile);
```

33. Comment fait-on une redirection en Laravel depuis un nom de route ? Et depuis une url ?

```
// Redirection via une route
return redirect()->route('login');

// Redirection via une URL
return redirect('home/dashboard');
```

34. A quoi sert le système de section dans Blade ? Pouvez-vous donner un exemple concret...

Blade permet de découper les pages HTML en plusieurs parties, pour par exemple définir un (ou plusieurs) layout(s) général qui inclueront à l'intérieur plusieurs sections (ex : en-tête, footer...) Exemple de layout comportant une section « sidebar », et qui affiche le contenu de sections « title » et « content » avec la directive @yield :

```
<html>
    <head>
        <title>Nom du site - @yield('title')</title>
    </head>
    <body>
```



```

        @section('sidebar')
            Sidebar principale du site.
        @show
        <div class="container">
            @yield('content')
        </div>
    </body>
</html>

```

35. Comment récupère t-on l'utilisateur connecté derrière une partie sécurisée du site ?

Grâce à la façade Auth :

```
$user = Auth::user();
```

36. Comment déterminer si l'utilisateur est connecté depuis une action de Contrôleur ?

```

if (Auth::check()) {
    // ...
}

```

37. Qu'est-ce qu'une vue partielle ?

C'est une vue qui sera appelée dans une autre, évitant ainsi la répétition de code.

Pars standard, les vues partielles sont stockées dans un dossier `Partial` et avec un underscore `_` au début du nom du fichier : `_header.blade.php`

38. Quel est la méthode utilisée pour vérifier si l'utilisateur a le bon login et mot de passe par rapport à la base de données ?

Grâce à la méthode « attempt »

```

    public function authenticate()
    {
        if (Auth::attempt(['email' => $email, 'password' => $password])) {
            // Authentication passed...
            return redirect()->intended('dashboard');
        }
    }
}

```

39. Comment hérite t-on d'une super vue `LayoutEmail.blade.php` ?

```
@extends('LayoutEmail.blade.php')
```

40. Comment enregistre t-on une variable «search» en session avec Laravel 5 et comment récupère t-on sa valeur?

```
Session::put('search', 'value');
```

```
$search = Session::get('search');
```

41. Que signifie @unless en Blade ?

Laravel n'exécutera les instructions à l'intérieur des balises @unless que si la condition n'est pas vérifiée :
Même chose que @if (! condition)

```
@unless (Auth::check())  
    Vous n'êtes pas enregistré.  
@endunless
```

=> à moins que l'utilisateur soit vérifié, on affiche un message « Vous n'êtes pas enregistré »

42. Comment crée t-on une boucle en Blade avec une condition de tableau vide ?

```
@forelse ($users as $user)  
    <li>{{ $user->name }}</li>  
@empty  
    <p>No users</p>  
@endforelse
```

43. Comment inclure une vue en Blade, quel est son intérêt principal et comment injecter une variable lors de l'inclusion ?

@include permet d'inclure une « sous-vue » dans une vue pour éviter de répéter du code HTML. Toutes les variables sont utilisables dans la sous-vue, mais on peut transmettre des variables supplémentaires

Exemple d'inclusion d'une vue « view.name » avec l'inclusion de données

```
@include('view.name', ['some' => 'data'])
```

44. Comment peut t-on verrouiller l'accès à un admin sécurisé uniquement pour les utilisateurs parisiens selon le zipcode?

Avec le mécanisme d'Authorisation derrière l'Authentification, et notamment son système de Gate :

45. Comment envoie t-on un E-Mail avec sujet, contenu, destinataire avec Laravel ?

```
Mail::send('email.salut', $data, function ($message) {  
    $message->from('from@example.com', 'Laravel');  
    $message->to('to@example.com')->cc('copie@example.com');  
    $message->subject('Sujet du mail');  
    $message->getSwiftMessage();  
});
```

46. A quoi peut servir un Middleware dans Laravel 5 ?

Les middlewares permettent de filtrer les requêtes HTTP. Le middleware Auth par exemple permet de vérifier si l'utilisateur courant est identifié.

C'est une classe qui s'interpose entre le Routing et le Controller.

47. Qu'est ce qu'un Service Provider et quelle est son utilité ?

Un service Provider est une classe en service, c'est à dire chargée au lancement de l'application dans laquelle nous pouvons injecter des autres services. Le provider peut être enregistré dans config.php et alié pour faciliter son accès.

48. Quels sont les avantages d'une ORM tel que Eloquent ?

Un ORM permet de :

- simplifier l'accès aux données
- manipuler des objets à partir de bases de données
- rendre l'accès aux données indépendant du SGBD
- Garantir la sécurité tel que l'injection SQL
- Optimiser le Cache des requêtes via les objets mis en cache (Objets Proxys - Lazy Loading)

49. Quels sont les intérêts à créer des scopes depuis le Model ?

L'intérêt principal est de pouvoir écrire des requêtes simples qui pourront être réutilisées et chaînées pour obtenir des requêtes plus complexes.

Également, Avoir une séparation entre la logique des Contrôleurs et celle du Model sur le MVC

50. Ecrivez 4 requêtes avec Query Builder et les mêmes depuis Eloquent, qui limite à 4 objets le résultat trié par created_at ?

```
// Avec le query builder
$nombre = DB::table('movies')
->where('deleted_at', NULL)
->orderBy('created_at')
->take(4)
->get();

// Avec Eloquent
$nombre = Movies::where('deleted_at', NULL)
->orderBy('created_at')
->take(4)
->get();
```

51. A quoi servent les relationships dans Laravel ?

Elles servent à définir les relations entre les classes dans l'ORM Eloquent.
Exemple de relation « One to One » dans une classe « User » :

```
class User extends Model
{
    public function phone()
    {
        return $this->hasOne('App\Phone');
    }
}
```

On peut ainsi obtenir facilement le résultat d'une relation :

```
$phone = User::find(1)->phone;
```

52. Pourquoi Laravel sort-il les résultats de requête derrière la classe Collection ? Pouvez-vous en donner une utilité ?

La classe Collection fournit des méthodes pour manipuler rapidement et simplement des données. Ces méthodes peuvent en supplément être utilisées à la chaîne.

Cette classe suis la classe Query Builder et est autoamtiquement executés après find() all() etc...

53. Quel sont les différences en Eloquent de la méthode get(), all() et toArray()

get() : retourne l'élément correspondant à une clé donnée

all() : retourne le tableau complet représenté par la collection

toArray() : convertit la collection en un array

54. Quelles sont les différences entre Query Builder et Eloquent ? Pouvez vous donner quelques exemples ?

Eloquent est un ORM complet, alors que le Query Builder se contente de construire une requête pour interroger une base de données

Exemple : cf question 51

55. A quoi sert de créer des ligne de console en Laravel 5 ?

Les lignes de commande permettent d'effectuer rapidement des actions préconçues. On peut également les utiliser pour programmer des tâches CRON.

Créer des classes et execution de php en ligne de console (module PHP-CLI)

56. A quoi sert [DB::Raw](#) dans Eloquent ou Query Builder ?

Laravel laisse la possibilité d'écrire des requêtes SQL « en dur » grâce à [DB::raw](#)

56. Comment encrypte t-on une variable sous Laravel depuis le controlleur et comment la

décrypter?

Service Crypt

```
// Encrypt
$user->fill([
    'secret' => Crypt::encrypt($request->secret)
])->save();

// Decrypt
$decrypted = Crypt::decrypt($encryptedValue);
```