



# Formation Zend Framework 2+

Romain Bohdanowicz

Twitter : @bioub

<http://www.formation.tech/>





# Introduction



- Romain Bohdanowicz  
Ingénieur EFREI 2008, spécialité Ingénierie Logicielle
- Expérience  
Formateur/Développeur Freelance depuis 2006  
Plus de 9 500 heures de formation animées
- Langages  
Expert : HTML / CSS / JavaScript / PHP / Java  
Notions : C / C++ / Objective-C / C# / Python / Bash / Batch
- Certifications  
PHP 5 / PHP 5.3 / PHP 5.5 / Zend Framework 1
- Particularités  
Premier site web à 12 ans (HTML/JS/PHP), Triathlète à mes heures perdues
- Et vous ?  
Langages ? Expérience ? Utilité de cette formation ?



- Qu'est-ce qu'un framework web ?
  - Un ensemble de bibliothèques, dont chacune se préoccupera de problématique précise (envoi de mail, gestion des formulaires, ...)
  - Un cadre de travail au travers d'une architecture recommandée (via des Design Patterns, des squelettes d'application, de documentation sur les bonnes pratiques d'architecture...)

# Framework - Historique



- Java  
Struts (2000), Spring (2003), GWT (2006), Play (2007)...
- Ruby  
Ruby on Rails (2005), Sinatra (2007)...
- Python  
Django (2005)...
- JavaScript  
Express (2009), AngularJS (2010), Ember.js (2011), Meteor (2012), Sails.js (2012)...



- Micro-framework
  - Avantages : Performance, facilité d'apprentissage
  - Inconvénients : Peu adaptés aux projets volumineux
  - Principaux frameworks : Slim (2010), Expressive (2016)
- « Convention over configuration » framework
  - Avantages : Développement rapide
  - Inconvénients : Conventions à connaître, difficile à personnaliser
  - Principaux frameworks : CakePHP (2005), CodeIgniter (2006), Laravel (2011), Phalcon (2012)
- Framework extensibles
  - Avantages : Adaptables à toutes les situations
  - Inconvénients : Nécessite de configurer, développement ralenti
  - Principaux frameworks : Symfony (2005), Zend Framework (2006)



- Historique
  - Zend Framework 1 (2006) : encore très utilisé en production
  - Zend Framework 2 (2012) : meilleures pratiques, plus extensible/modulable
  - Zend Framework 3 (2016) : plus découplé, squelette MVC ou Expressive
- Licence BSD Modifiée
  - Permissive, code utilisable et modifiable, y compris dans des projets propriétaires
  - Doit reproduire le copyright dans le code source et la documentation du produit
- Auteurs de Zend Framework 2
  - Plus de 500 contributeurs
  - Les plus actifs : Matthew Weier O'Phinney, Ralph Schindler, Evan Coury, prolic, Maks3w, Marc Bennewitz, Abdul Malik Ikhsan, Enrico Zimuel, Rob Allen, Michaël Gallego, Vincent Blanchon, Ben Scholzen, Marco Pivetta...
  - Consultez leurs blogs, suivez les sur Twitter.



- Getting started

Tutoriel de démarrage, à faire avant toute chose pour comprendre les grands principes du Framework.

<https://docs.zendframework.com/tutorials/getting-started/overview/>

- Reference Guide

Chaque composant y est présenté avec des exemples. Ne suffit pas toujours.

<https://docs.zendframework.com>

- Webinars

Vidéoconférence enregistrées par les développeurs du framework.

[http://www.zend.com/en/webinars/recorded/show-by-topic/242\\_zend+framework](http://www.zend.com/en/webinars/recorded/show-by-topic/242_zend+framework)





- Blog  
<https://framework.zend.com/blog>
- Blogs des développeurs  
Contiennent plein d'articles et de tutoriaux.
- Code source  
Parfois certaines fonctionnalités sont non documentées.
- Google, StackOverflow...  
Une bonne recherche évite parfois un long article.



- Forum  
<https://discourse.zendframework.com>
- Slack  
<https://zendframework-slack.herokuapp.com>
- Newsletter  
<https://tinyletter.com/mwopzend>
- Groupe Facebook  
Plusieurs milliers de membres :  
<https://www.facebook.com/groups/zendframework2/>
- Meetup Paris Zend Framework  
Groupe Parisien qui organise des conférences régulièrement  
<http://www.meetup.com/fr/Paris-Zend-Framework-Meetup/>



- **Zend Framework 2 - Industrialisez vos développements PHP**

*Sébastien CHAZALLET, Editions ENI, 2013*

- Le meilleur choix pour commencer, création d'un projet pas à pas, bon rappels des outils PHP, des concepts de sécurité, de design patterns et des notions d'optimisation de performance

- **Zend Framework 2 - Développez des applications web mobiles (PHP, HTML5, JavaScript, NoSQL)**

*Cédric DERUE, Editions ENI, 2013*

- Orienté sur la création de Service Web REST, présente Zend Studio, MongoDB via Doctrine ODM, rappels sur les Design Patterns, chapitres sur JavaScript et hébergement sur OpenShift

- **Au coeur de Zend Framework 2**

*Vincent BLANCHON, lulu.com, 2012*

- Le plus technique, intéressant pour comprendre le fonctionnement du framework, à éviter réserver aux utilisateurs avancés de Zend Framework

# Zend Framework - Evolutions depuis ZF1



- Utilisation des Namespaces  
ex : `Zend_Soap_AutoDiscover` devient `Zend\Soap\AutoDiscover`
- Architecture MVC  
L'architecture devient plus modulaire afin de permettre de récupérer facilement des pages développés par d'autres (ex : module `ZFCUser` pour la gestion des utilisateurs)
- Injection de Dépendance  
Zend Framework 2 encourage l'utilisation de l'injection de dépendance pour faciliter l'écriture des tests unitaires (disparition des Singletons et Compositions)
- Evolutions des Composants  
Certains composants ont été ajoutés (`Zend\ServiceManager`), d'autres fortement modifiés (`Zend\Form`) et certains supprimés (`Zend_Registry`)



# PHP Objet



- Une classe : un concept
- Un objet : une représentation en mémoire de ce concept

```
<?php
namespace Application\Entity;

class Contact
{
    public $id;
    public $prenom;
    public $nom;
    public $email;
    public $telephone;
}
```

# PHP Objet - Encapsulation



- Principe d'encapsulation

Un objet doit être vu comme une boîte noire, son fonctionnement interne peut-être complexe, son utilisation doit être simple. Les propriétés ne sont jamais publiques. Utilisation de getters/setters, permet également de limiter à la lecture seule et d'encapsuler les règles de validation.

```
<?php
namespace Application\Entity;

class Contact
{
    protected $prenom;
    protected $nom;

    public function getNom()
    {
        return $this->nom;
    }

    public function setNom($nom)
    {
        $this->nom = $nom;
    }

    public function getPrenom()
    {
        return $this->prenom;
    }

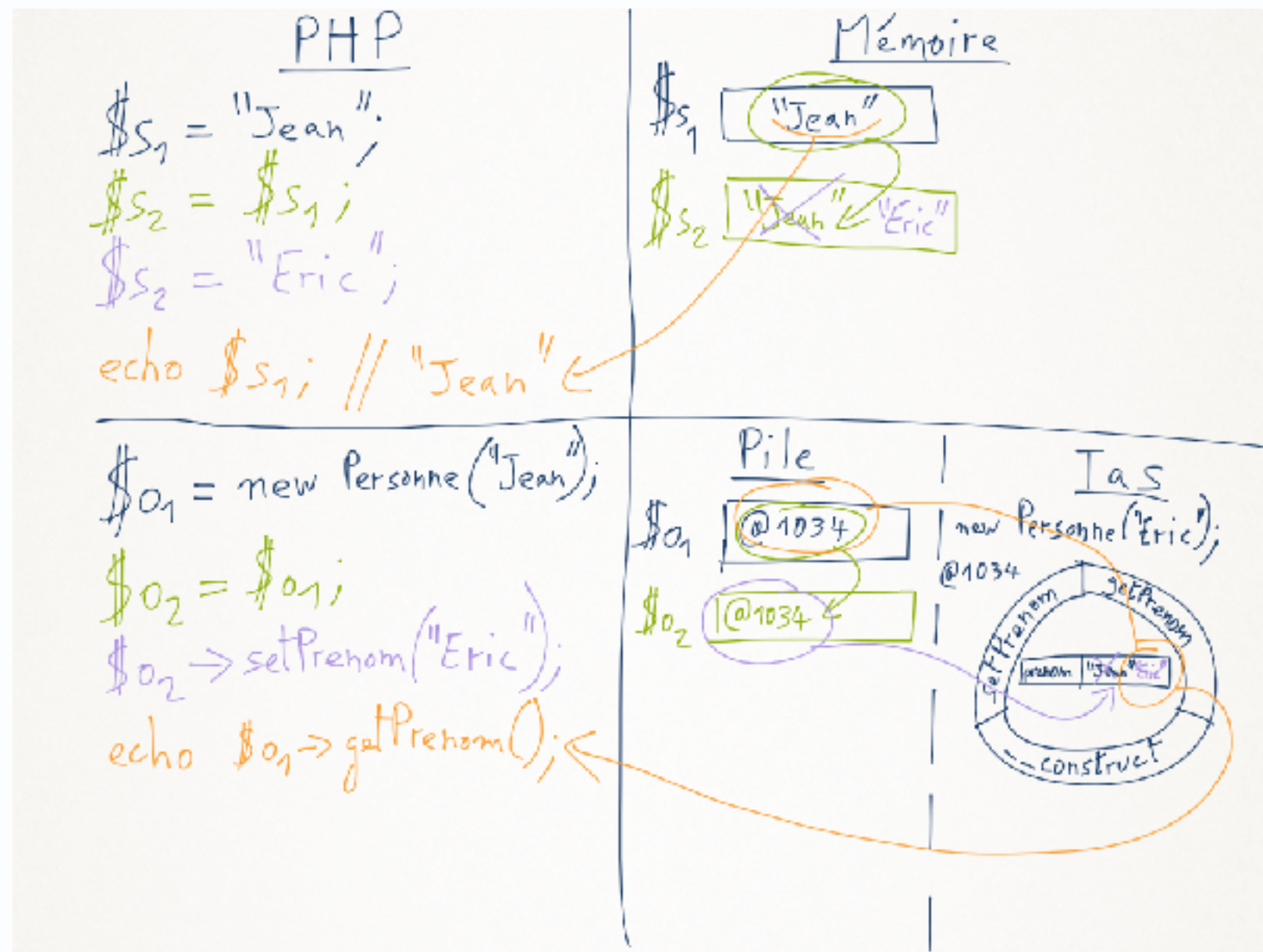
    public function setPrenom($prenom)
    {
        $this->prenom = $prenom;
    }
}
```

# PHP Objet - Encapsulation



## ▸ Référence

A la différence des autres types (int, boolean, float, string, array...) les objets se manipulent au travers de références.





# PHP Objet - Association



## ▸ Association

Un des propriété d'un objet contient une référence vers un autre objet.

```
<?php
namespace Application\Entity;

class Contact
{
    protected $id;
    protected $prenom;
    protected $nom;
    protected $email;
    protected $telephone;

    /**
     * @var Societe
     */
    protected $societe;
}
```

```
<?php
namespace Application\Entity;

class Societe
{
    protected $nom;
    protected $siteweb;
}
```



## ▸ Héritage

Une classe réutilise les membres d'une autre classe.

```
<?php
namespace Application\Entity;

class Contact
{
    protected $id;
    protected $prenom;
    protected $nom;
    protected $email;
    protected $telephone;

    /**
     * @var Societe
     */
    protected $societe;
}
```

```
<?php
namespace Application\Entity;

class Salarie extends Contact
{
    protected $salaire;
}
```



## ▸ Interface

La définition d'une liste de méthodes, implémenter une interface oblige à implémenter ses méthodes.

Dans ZF2 : leur nom se termine toujours par Interface

```
<?php
namespace Application\Entity;
use Zend\Stdlib\ArraySerializableInterface;
class Contact implements ArraySerializableInterface
{
    // ...

    public function exchangeArray(array $array)
    {
        foreach ($array as $key => $value) {
            if (property_exists($this, $key)) {
                $this->$key = $value;
            }
        }
    }

    public function getArrayCopy()
    {
        return get_object_vars($this);
    }
}
```

```
<?php
namespace Zend\Stdlib;
interface ArraySerializableInterface
{
    public function exchangeArray(array $array);
    public function getArrayCopy();
}
```



- Classe abstraite

Une classe qui n'a pour vocation à être utilisée qu'au travers d'un héritage. Peut également imposer l'implémentation de certaines méthodes comme une interface. Comme pour une interface, c'est la garantie que certaines méthodes seront bien présentes (programmation par contrat).

Dans ZF2 leur nom commence toujours par Abstract.

```
<?php
namespace Zend\Mvc\Controller;

abstract class AbstractActionController extends AbstractController
{
    // ...
}
```

```
<?php
namespace AddressBook\Controller;

use Zend\Mvc\Controller\AbstractActionController;

class ContactController extends AbstractActionController
{
    // ...
}
```

# PHP Objet - Classe abstraite



- PHP FIG  
PHP Framework Interop Group, groupe de travail regroupant les principaux créateurs de frameworks/bibliothèques créant des normes PHP.  
<http://www.php-fig.org/>
- PSR-0  
Autoloading Standard
- PSR-1  
PSR-0 + Basic Coding Standard
- PSR-2  
PSR-1 + Coding Style Guide
- PSR-3  
Logger Interface
- PSR-4  
Improved Autoloading
- PSR-7  
HTTP Message Interface



# Installation



- Prérequis

Zend Framework 1 : PHP 5.2 (5.2.4 puis 5.2.11 à partir de ZF 1.12)

Zend Framework 2 : PHP 5.3.3

Zend Framework 2.3 : PHP 5.3.23

Zend Framework 2.5 : PHP 5.5

Zend Framework 3 : PHP 5.6 (PHP 7.1 pour les nouveaux composants ou versions majeures)

Apache + mod\_rewrite ou équivalent

Certains codes optionnels présents dans le framework nécessitent PHP 5.4 (traits)

Certains composants nécessitent des extensions de PHP particulières.

Le framework encourage le déploiement via Git

- 2 approches possibles de Zend Framework :

Mode « Glue » : utilisation de certains composants dans une application architecturée sans Zend Framework

Mode « Full-stack » : utilisation de l'architecture de Zend Framework (composant Zend\MVC)



# Composer



# Composer - Introduction



- Utilité de composer

Composer est un système de gestion de dépendances dans un projet PHP.  
Il permet de télécharger et de mettre à jour automatiquement les bibliothèques utilisés dans nos projets ainsi que les bibliothèques dont elles dépendent.

- Equivalents dans d'autres langages :

Ruby : gem, bundler

Python : pip

JavaScript : npm et bower

Objective-C : CocoaPods

- Prérequis :

PHP 5.3.2+

Pour certains packages : git, svn or hg

- Documentation :

Officielle : <https://getcomposer.org/doc/>

Cheatsheet : <http://composer.json.jolicode.com>

# Composer - Installation



- Installation de Composer

Avec curl

```
curl -sS https://getcomposer.org/installer | php
```

Avec PHP

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

Téléchargement Manuel

Télécharger le « latest snapshot » sur <http://getcomposer.org/download/>

Composer peut s'installer localement (dans un projet) ou globalement (utile pour ne pas avoir à reconfigurer notre IDE à chaque projet).

- Utilisation d'un proxy

Définir une variable d'environnement *http\_proxy*

```
setenv http_proxy http://login:pass@hote_du_proxy
```

Eventuellement configurer les variables d'environnement suivantes à *false*

```
setenv HTTP_PROXY_REQUEST_FULLURI false
```

```
setenv HTTPS_PROXY_REQUEST_FULLURI false
```

# Composer - Principales commandes



- `composer self-update`  
Met à jour automatiquement composer lui-même
- `composer init`  
Crée un fichier *composer.json* en mode interactif
- `composer require`  
Ajoute une dépendance dans le fichier *composer.json* et l'installe
- `composer install (--no-dev)`  
Installe les dépendances présentent dans le fichier *composer.lock* ou si inexistant dans le fichier *composer.json*
- `composer update`  
Met à jour automatiquement les bibliothèques en tenant compte des versions souhaitées par le fichier *composer.json*
- `composer dump-autoload -o`  
Optimise l'autochargeur de classes en éditant le fichier *vendor/composer/autoload\_classmap.php*

# Composer - Le fichier composer.json



- "require": {}, dépendances générales du projet  
"require-dev": {}, dépendances dans un environnement de développement
- Packagist : annuaire contenant les noms et versions des bibliothèques à utiliser pour composer :  
<https://packagist.org>
- Plus de détails et autres options du le fichier composer.json :  
<https://getcomposer.org/doc/04-schema.md>  
<http://composer.json.jolicode.com>

```
{
  "name": "AddressBookZF2",
  "description": "Un carnet d'adresse utilisant Zend Framework 2",
  "license": "BSD-3-Clause",
  "authors": [{
    "name": "Romain Bohdanowicz",
    "email": "romain.bohdanowicz@gmail.com"
  }],
  "require": {
    "php": ">=5.3.3",
    "zendframework/zendframework": "2.3.*",
    "doctrine/doctrine-orm-module": "0.*"
  },
  "require-dev": {
    "zendframework/zftool": "dev-master",
    "zendframework/zend-developer-tools": "dev-master"
  }
}
```



Mode « Glue »

# Mode « Glue » - Introduction



- Objectif  
Envoyer un mail via un serveur SMTP en utilisant Zend\Mail.
- Voici notre application existante

```
<!-- index.php -->
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Zend/Mail</title>
</head>
<body>
<form method="post">
<div>
Titre :
<input type="text" name="titre">
</div>
<div>
Message :
<textarea name="message"></textarea>
</div>
<div>
<input type="submit">
</div>
</form>
</body>
</html>
```

A screenshot of a web browser window titled 'Zend/Mail'. The address bar shows 'localhost:8080'. The page content includes a form with two input fields: 'Titre :' (a text input) and 'Message :' (a text area). Below these fields is a 'Valider' button. The browser's interface includes back, forward, and search buttons, as well as a star icon for bookmarks.



- Installation de Zend\Mail

Nous allons installer le composant Zend\Mail dans un répertoire « vendor » à la racine de notre application.

- Plusieurs options possibles :

- Télécharger une archive de Zend Framework

Nécessite d'intervenir sur les fichiers, long, difficile à déployer, inclusion du framework complet

- Utilisation de composer

La meilleure option, télécharge automatiquement toute les dépendances, mise à jour et déploiement via des commandes simples et scriptable

# Zend Framework - Utilisation en mode « Glue »



## ▸ Dépendance

Dans ce projet nous n'avons pas besoin du framework complet, mais seulement de la bibliothèque Zend\Mail et de ses dépendances.

## ▸ Versionnage sémantique

Dans 2.3.1 :

- 2 est la version majeure, le changement de version majeure marque une rupture de compatibilité ascendante (le gros de l'appli est à réécrire)
- 3 est la version mineure, pas ou peu de rupture de comptabilité ascendante (lire le guide de migration), ajout de fonctionnalité et donc pas de compatibilité descendante si elles sont utilisées
- 1 est la version de correctif, les modifications du code sont des correctifs de bugs et de sécurité

Il faut donc toujours être sur la version correctif la plus récente. Dans le fichier composer.json cela peut s'écrire : "2.3.\*", "~2.3.1", ou ">=2.3, <2.4 »

Plus d'infos sur le versionnage sémantique : <http://semver.org/lang/fr/>

```
{
  "name": "romain/zend_mail",
  "authors": [
    {
      "name": "Romain Bohdanowicz",
      "email": "romain.bohdanowicz@gmail.com"
    }
  ],
  "require": {
    "zendframework/zend-mail": "2.3.*"
  }
}
```





- Installation des dépendances

*composer update* analyse le fichier *composer.json*, vérifie sur le serveur si une nouvelle version de la bibliothèque ou de ses dépendances est disponible, puis les déploie automatiquement dans le répertoire *vendor* (répertoire contenant les bibliothèques externes).

- Déploiement/partage du projet

Une fois les bibliothèques installées, le projet testé, nous souhaitons l'exporter vers un autre environnement (passage vers la production par exemple).

Cette étape étant critique il ne faut pas installer de nouvelles versions des dépendances au risque d'introduire des bugs.

*composer update* crée donc un fichier *composer.lock* avec les versions exactes installées. La commande *composer install* va réinstaller précisément ces dépendances sans chercher de nouvelles versions.

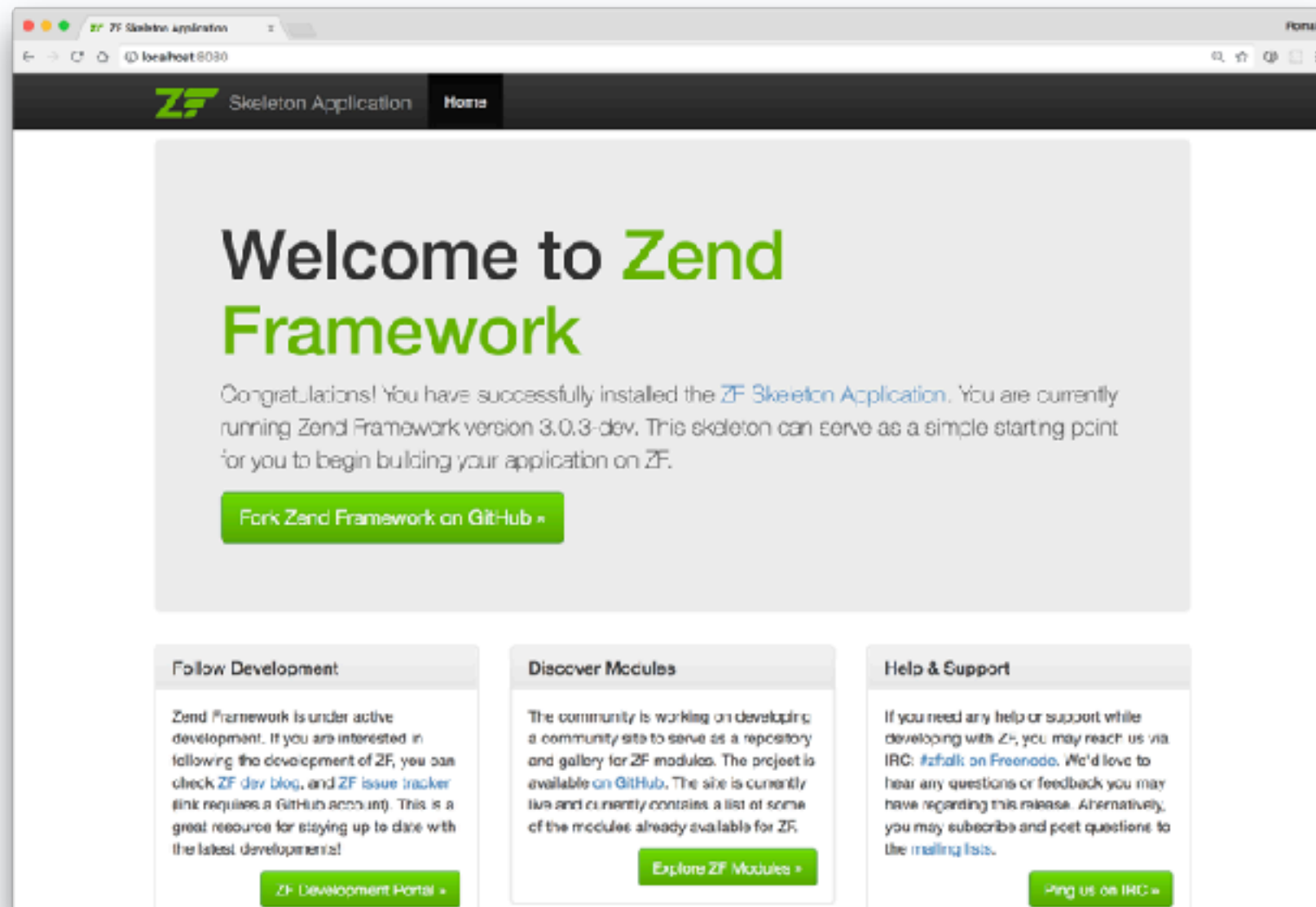


Mode « Full-stack »

# Mode « Full-Stack » - Introduction



- Objectif  
Installer le squelette d'application fourni par les développeurs de Zend Framework 2
- Résultat



# Mode « Full-Stack » - Built-in Server



- Création du squelette

```
composer create-project -sdev zendframework/skeleton-  
application CHEMIN_DU_PROJET
```

- Built-in Server

Depuis PHP 5.4, PHP inclus un serveur HTTP qui permet pendant le développement de se passer d'Apache ou autre serveur web

- Lancement de l'application

```
composer serve
```



- Définition

ZFTool est un outil en ligne de commande créé par les développeur de Zend Framework et permettant de simplifier la création de fichiers.

- Installation

Ajouter la ligne `"zendframework/zftool": "dev-master"` dans la section *require-dev* de *composer.json*

```
{  
    "require-dev": {  
        "zendframework/zftool": "dev-master"  
    }  
}
```

- Utilisation

A exécuter depuis la racine du projet :

`vendor/bin/zf.php`

ou sous Windows

`vendor\bin\zf.php.bat`



## ▸ Principales commandes

- Création d'un module :  
`vendor/bin/zf.php create module <name>`
- Création d'un contrôleur :  
`vendor/bin/zf.php create controller <name> <module>`
- Création d'une action et de sa vue associée :  
`vendor/bin/zf.php create action <name> <controllerName> <module>`

# ZFTool - Présentation



```
mbp-de-romain:ZF2SkeletonApplication remain$ vendor/bin/zf.php
ZFTool - Zend Framework 2 command line Tool
```

---

## ZFTool

---

### Basic information:

```
zf.php modules [list]          show loaded modules
zf.php version | --version     display current Zend Framework version
```

### Diagnostics

```
zf.php diag [options] [module name]    run diagnostics
```

```
[module name]    (Optional) name of module to test
-v --verbose     Display detailed information.
-b --break       Stop testing on first failure
-q --quiet       Do not display any output unless an error occurs.
--debug          Display raw debug info from tests.
```

### Application configuration:

```
zf.php config list              list all configuration options
zf.php config get <name>        display a single config value, i.e. "config get db.host"
zf.php config set <name> <value> set a single config value (use only to change scalar values)
```

### Project creation:

```
zf.php create project <path>    create a skeleton application
```

```
<path>    The path of the project to be created
```

# ZFTool - Présentation



## Module creation:

```
zf.php create module <name> [<path>]    create a module
```

<name> The name of the module to be created  
<path> The root path of a ZF2 application where to create the module

## Controller creation:

```
zf.php create controller <name> <module> [<path>]    create a controller in module
```

<name> The name of the controller to be created  
<module> The module in which the controller should be created  
<path> The root path of a ZF2 application where to create the controller

## Action creation:

```
zf.php create action <name> <controllerName> <module> [<path>]    create an action in a controller
```

<name> The name of the action to be created  
<controllerName> The name of the controller in which the action should be created  
<module> The module containing the controller  
<path> The root path of a ZF2 application where to create the action

## Classmap generator:

```
zf.php classmap generate <directory> <classmap file> [--append|-a] [--overwrite|-w]
```

<directory> The directory to scan for PHP classes (use "." to use current directory)  
<classmap file> File name for generated class map file or - for standard output. If not supplied, defaults to autoload\_classmap.php inside <directory>.  
--append | -a Append to classmap file if it exists  
--overwrite | -w Whether or not to overwrite existing classmap file

## Zend Framework 2 installation:

```
zf.php install zf <path> [<version>]
```

<path> The directory where to install the ZF2 library  
<version> The version to install, if not specified uses the last available





# Architecture de Zend Framework

# Architecture MVC - Présentation



- Définition

L'architecture MVC est un Design Pattern apparu en Smalltalk et très répandu dans les frameworks web

- Objectif

L'objectif est de séparer les responsabilités de 3 types de composants : le Modèle (Model), la Vue (View), le Contrôleur (Controller)

- Documentation :

<http://martinfowler.com/eaaCatalog/modelViewController.html>

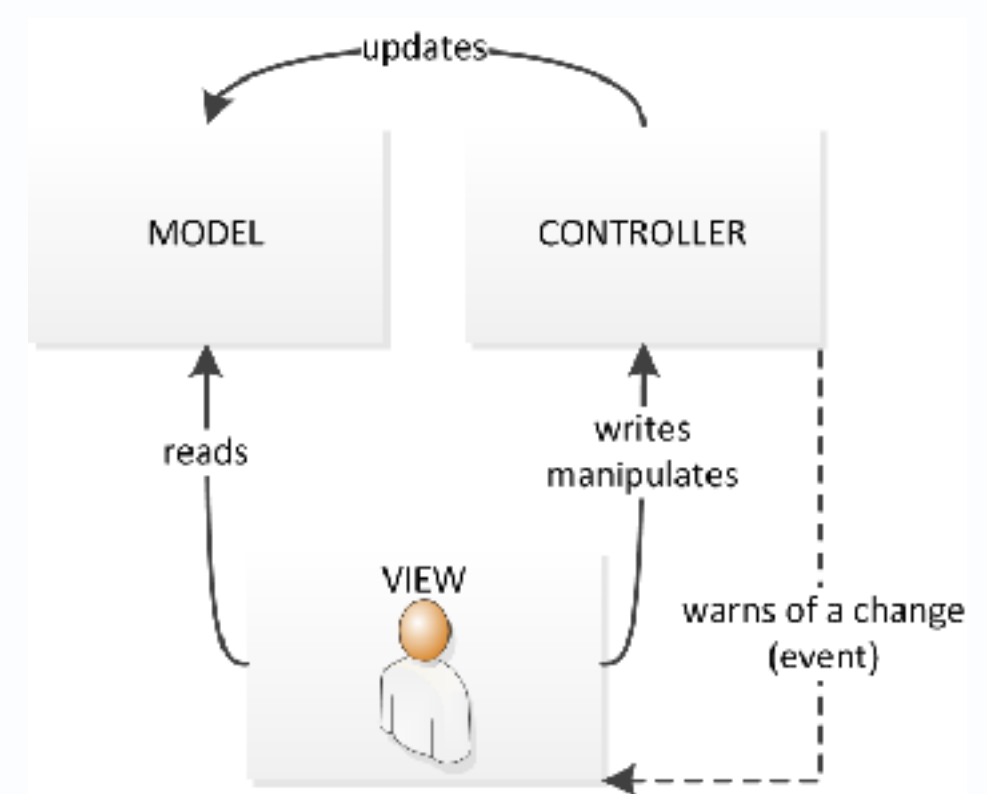
<http://martinfowler.com/eaaDev/uiArchs.html>

<http://fr.wikipedia.org/wiki/Modèle-vue-contrôleur>

# Architecture MVC - Présentation



- Modèle  
Données, accès aux données, validation
- Vue  
Rendu. Se limiter à :
  - affichage de variable
  - bloc conditionnels if .. else if .. else (ex : afficher ou non message d'erreur, menu qui dépend d'une authentification)
  - boucles foreach (uniquement foreach, ce qui impose d'avoir trié/filtré au préalable)
  - appel à des fonctions de filtrage, de formatage, de rendu (parfois appelées aides de vues)
- Contrôleur  
Analyse de la requête, interrogation du Modèle, transmission des données à la vue, gestion des erreurs, des redirections...



# Architecture MVC - Présentation



- Zend\MVC

L'architecture MVC de Zend Framework est mise en place dans le composant Zend\MVC.

- Conventions

Zend Framework 2 repose sur un certain nombre de conventions, mais celles-ci pourraient être modifiées dans une utilisation avancée du framework.



- Module

L'une des principales nouveautés de Zend Framework 2 par rapport à la version 1 est de reposer intégralement sur la notion de Module.

Un module est un ensemble de composants réutilisables, pouvant contenir ou non une architecture MVC. Ils peuvent également permettre d'architecturer notre application.

Exemple de module :

- Gestion des utilisateurs (inscription, authentication...)
- Back-end / Front-end
- Interfaçage avec une autre bibliothèque (Doctrine, ZFCTwig...)

Une liste de modules open-source est présente sur :

<https://zfmodules.com>

- On retrouve la même notion sur Symfony sous l'appellation Bundle



# Squelette d'application

# Squelette d'application - Squelette d'application



- composer.json/.lock  
Dépendances de l'application
- config/  
Configuration globale
- data/  
Données volatiles/temporaires (sessions, cache, bases sqlite, logs...)
- init\_autoloader.php  
Autochargeur de classes (cf vendor/composer)
- module/  
Modules spécifiques à l'application
- public/  
Fichiers statiques
- vendor/  
Classes/Modules externes à l'application (réutilisables)

```
— composer.json
— composer.lock
— config
  — application.config.php
  — autoload
    — .gitignore
    — global.php
    — local.php.dist
— data
  — cache
— init_autoloader.php
— module
— public
  — .htaccess
  — css
  — fonts
  — img
  — index.php
  — js
— vendor
  — autoload.php
  — bin
  — composer
    — ClassLoader.php
    — autoload_classmap.php
    — autoload_namespaces.php
    — autoload_psr4.php
    — autoload_real.php
    — installed.json
  — zendframework
```

# Squelette d'application - Configuration globale



- application.config.php  
Configuration des modules, du cache de config
- autoload/  
Configuration de l'application (base de données, logs, cache, configuration des modules vendor).
- autoload/\*.global.php  
Configurations communes à tous les environnements (poste de développement, serveur d'intégration, de production)
- autoload/\*.local.php  
Configurations locales à un environnement (login, passwords...). Le fichier .gitignore s'assure que dans le cadre d'un déploiement via git, les fichier \*.local.php ne soient pas transférés.

Les fichiers \*.global.php et \*.local.php sont fusionnés par ZF2

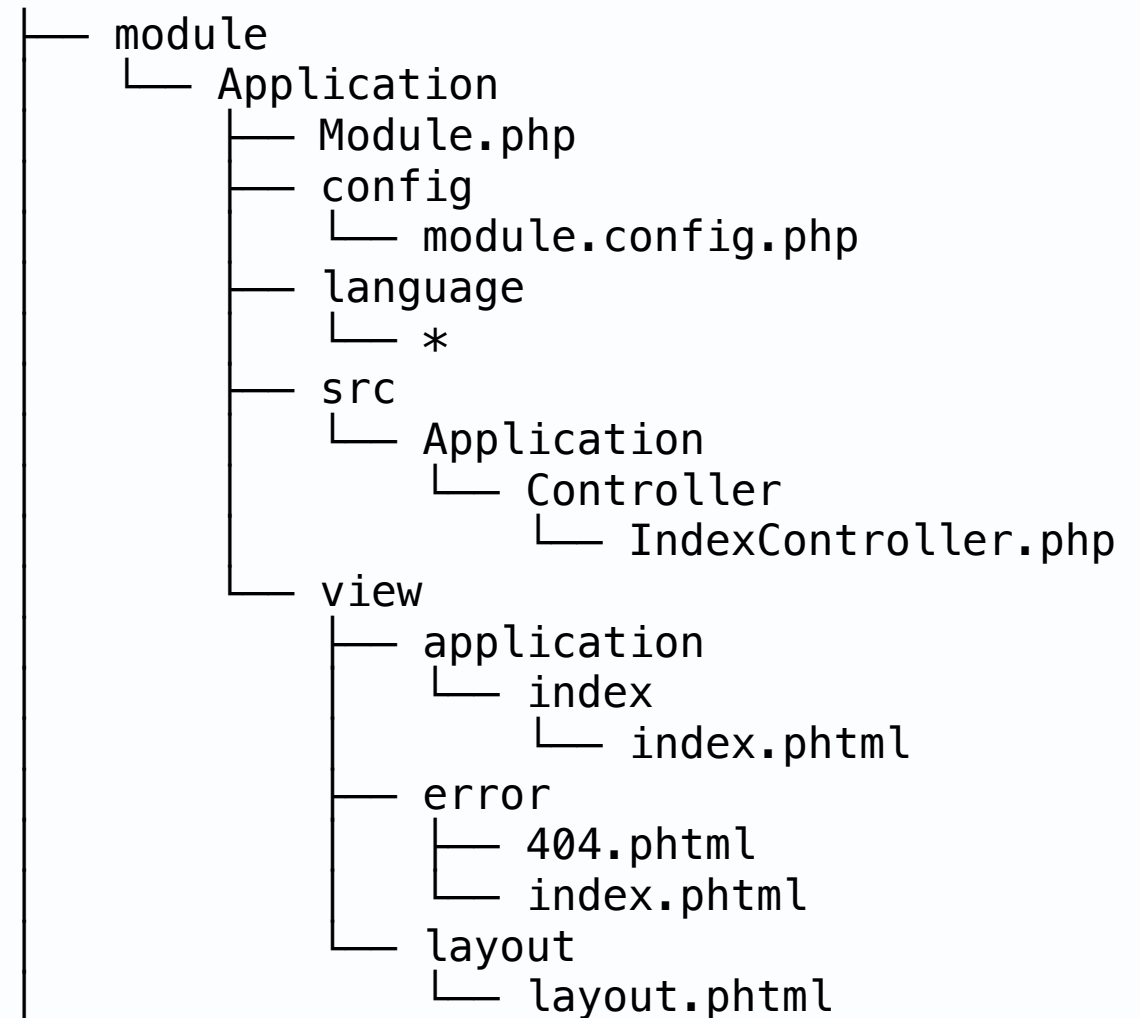
```
config
├── application.config.php
├── autoload
│   ├── .gitignore
│   ├── db.global.php
│   ├── db.local.php
│   ├── doctrine.global.php
│   ├── doctrine.local.php
│   └── zenddevelopertools.local.php
```



# Squelette d'application - Architecture d'un module



- Module.php  
Configuration au format PHP
- config/module.config.php  
Configuration au format clé/valeur
- src/Application  
Classes PHP (contrôleurs, formulaires, validateurs, services...)
- view/  
Fichiers vues



# Squelette d'application - Répertoire public



- Le répertoire public doit être la racine du site web de sorte que tous les autres ne soit pas accessible via HTTP.
- Premièrement créer un nom de domaine local  
Sur un poste de développement : éditer le fichier `/etc/hosts` (Mac, Linux) ou `C:\Windows\System32\drivers\etc\hosts`  
Ajouter une ligne du type :  
`127.0.0.1 skeletonapplication.zf2`
- Ensuite créer un Virtual Host dans la configuration d'Apache (`conf/extra/httpd-vhost.conf` ou équivalent)

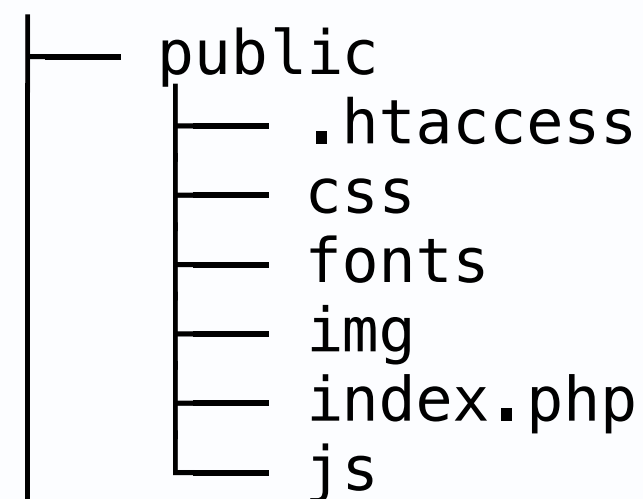
```
<VirtualHost *:80>  
    DocumentRoot "/Users/romain/www/ZF2SkeletonApplication/public"  
    ServerName skeletonapplication.zf2  
  
    <Directory "/Users/romain/www/ZF2SkeletonApplication/public">  
        Options Indexes FollowSymLinks  
        AllowOverride All  
        Require all granted  
    </Directory>  
</VirtualHost>
```

# Squelette d'application - Répertoire public



## ► .htaccess

Ecrase la configuration d'Apache pour le répertoire en cours. Le fichier redirige toutes les requêtes qui ne pointe ni vers un dossier, fichier ou lien symbolique vers le fichier index.php



**RewriteEngine On**

**RewriteCond** %{REQUEST\_FILENAME} -s [OR]

**RewriteCond** %{REQUEST\_FILENAME} -l [OR]

**RewriteCond** %{REQUEST\_FILENAME} -d

**RewriteRule** ^.\*\$ - [NC,L]

**RewriteCond** %{REQUEST\_URI}::\$1 ^(/.+)(.+)::\2\$

**RewriteRule** ^(.\*) - [E=BASE:%1]

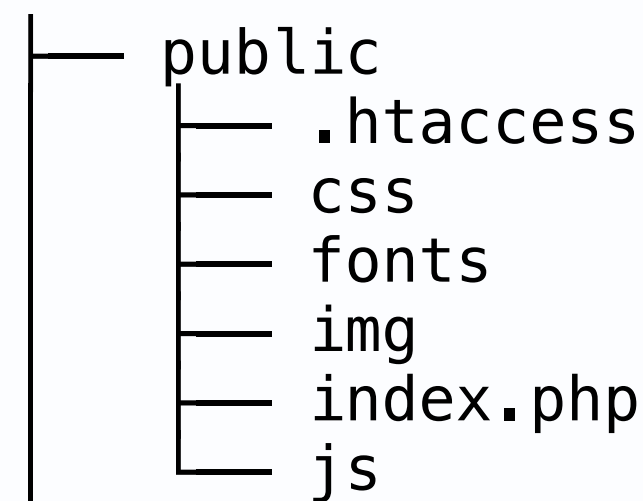
**RewriteRule** ^(.\*)\$ %{ENV:BASE}index.php [NC,L]

# Squelette d'application - Répertoire public



## ▸ index.php

Point d'entrée de l'application, toute les requêtes traitées par le framework passe par ce fichier (en Design Pattern on parle de Front Controller). Les erreurs 404 ne sont donc plus traitées par Apache mais par ZF2.



```
<?php
/**
 * This makes our life easier when dealing with paths. Everything is relative
 * to the application root now.
 */
chdir(dirname(__DIR__));

// Decline static file requests back to the PHP built-in webserver
if (php_sapi_name() === 'cli-server' && is_file(__DIR__ . parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH))) {
    return false;
}

// Setup autoloading
require 'init_autoloader.php';

// Run the application!
Zend\Mvc\Application::init(require 'config/application.config.php')->run();
```



# Contrôleurs et Routes



- Contrôleur

Catégorie de page, exemple : pages liés à un compte

- Action

Action possible sur ce type de page (inscription, afficher son profil, modifier son mot de passe, connexion/déconnexion)

```
<?php
namespace Application\Controller;

use Zend\Mvc\Controller\AbstractActionController;
use Zend\View\Model\ViewModel;

class IndexController extends AbstractActionController
{
    public function indexAction()
    {
        return new ViewModel();
    }
}
```

# Router - Configuration dans module.config.php



- controllers

Pour pouvoir accéder à nos pages, il faut expliquer à Zend Framework comment instancier les classes contrôleurs (voir le chapitre sur le service manager)

- router

Une fois le contrôleur créé et déclaré, il faut définir les configurer le Router. Une route fait le lien entre l'action d'un contrôleur et une URL (qui jusque là menait à une erreur 404)

```
return array(  
    'router' => array(  
        'routes' => array(  
            'home' => array(  
                'type' => 'Literal',  
                'options' => array(  
                    'route' => '/',  
                    'defaults' => array(  
                        'controller' => 'Application\Controller\Index',  
                        'action' => 'index',  
                    ),  
                ),  
            ),  
        ),  
    ),  
    // ...  
    'controllers' => array(  
        'invokables' => array(  
            'Application\Controller\Index' => 'Application\Controller\IndexController'  
        ),  
    ),  
    // ...  
);
```

# Router - Configuration dans module.config.php



- Literal

Une simple chaîne de caractères.

Ex : /, /inscription, /contacts

```
'home' => array(  
    'type' => 'Literal',  
    'options' => array(  
        'route' => '/',  
        'defaults' => array(  
            'controller' => 'Application\Controller\Index',  
            'action' => 'index',  
        ),  
    ),  
)
```



# Router - Types de routes



- Segment  
Une URL avec un ou plusieurs paramètres.  
Ex : /actualites/31245, /membre/romain
- Ne pas confondre avec la Query String : /actualites?id=31245  
Les 2 sont possibles, depuis le contrôleur :

```
'show' => array(  
    'type' => 'Segment',  
    'options' => array(  
        'route' => '[:id]',  
        'constraints' => array(  
            'id' => '[1-9][0-9]*'  
        ),  
        'defaults' => array(  
            'controller' => 'AddressBook\Controller>Contact',  
            'action' => 'show'  
        )  
    )  
)  
,
```

```
// Paramètres de la route Segment  
$id = $this->params("id");  
  
// Paramètres de la route Query String  
$id = $this->request->getQuery('id')
```

# Router - Types de routes



- ▶ Hostname

Pour les sous-domaines : romain.monsite.com, eric.monsite.com

- ▶ Method

Pour avoir une action différente en fonction de la method HTTP :

GET -> afficherFormAction

POST -> traiterFormAction

- ▶ Regex

En utilisant une Expression Régulière, on lui préférera Segment, plus simple à utiliser et qui est converti en Regex.

- ▶ Scheme

En fonction du protocole utilisé (HTTP ou HTTPS)

# Router - Types de routes



## ▸ Part

Pour définir des préfixes communs à plusieurs routes.

```
'contact' => [
  'type' => 'Literal',
  'options' => array(
    'route' => '/contact',
  ),
  'may_terminate' => false,
  'child_routes' => [
    'show' => [
      'type' => 'Segment',
      'options' => [
        'route' => '/:id',
        'constraints' => [
          'id' => '[1-9][0-9]*'
        ],
        'defaults' => [
          'controller' => 'AddressBook\Controller>Contact',
          'action' => 'show'
        ]
      ]
    ],
  ],
  'add' => [
    'type' => 'Literal',
    'options' => [
      'route' => '/add',
      'defaults' => [
        'controller' => 'AddressBook\Controller>Contact',
        'action' => 'add'
      ]
    ]
  ],
],
// ...
]
```



- Zend\Mvc\Controller\AbstractActionController  
Contient la méthode *onDispatch* qui permet d'appeler la bonne action ainsi que la méthode *createHttpNotFoundModel* qui crée une erreur 404 :

```
// Créé une erreur 404
if (!$contact) {
    return $this->createHttpNotFoundModel($this->response);
}
```

- La classe AbstractController est sa généralisation qui contient une référence vers la requête, la réponse et l'accès aux plugins de contrôleur.
- 2 autres spécialisations existent :
  - AbstractRestController, permet de faciliter la création de service Restful
  - AbstractConsoleController, pour créer des programme console



- Définition  
Les plugins de contrôleur sont des méthodes qui aident à l'écriture des contrôleurs.
- FlashMessenger  
Permet d'écrire un message dans la session avant une redirection. Puis de le récupérer facilement depuis un autre contrôleur ou vue.
- Redirect  
Retourne une réponse contenant une redirection vers une route.

```
public function addAction()  
{  
    // ...  
    $this->flashMessenger()->addSuccessMessage('Le contact a bien été modifié');  
    return $this->redirect()->toRoute('home');  
    // ...  
}  
  
public function listAction()  
{  
    // ...  
    $messages = $this->flashMessenger()->getSuccessMessages();  
  
    return array(  
        'messages' => $messages,  
    );  
}
```



- Forward  
Permet de rediriger vers une autre action en interne (sans redirection HTTP).
- Identity  
Permet de récupérer l'identité de l'utilisateur authentifié.
- Layout  
Permet de définir un autre layout depuis le contrôleur.
- Params  
Pour récupérer les paramètres provenant d'une route, d'une superglobale.
- Post/Redirect/Get et File Post/Redirect/Get  
Permet d'éviter que le navigateur affiche une alerte lorsqu'on navigue dans l'historique contenant une requêtes de type POST.
- Url  
Crée une URL depuis une route.
- Plus de détails : <http://framework.zend.com/manual/current/en/modules/zend.mvc.plugins.html>



- "Thin controllers, fat models"
  - Une action doit rester courte (en moyenne, 15-20 lignes maximum)
  - Il ne doit pas agir directement avec la persistance (pas d'appels en base !). Il doit communiquer avec la couche service pour ça.



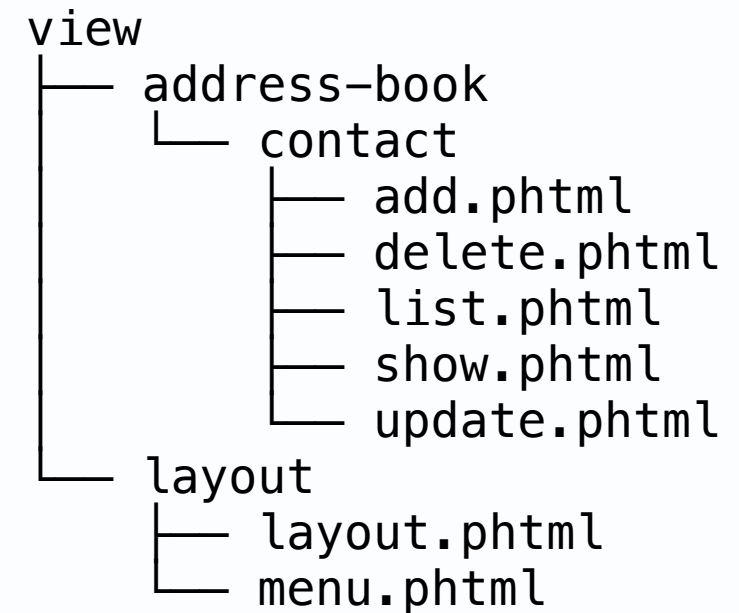
# Vues



# Vues - Configuration dans module.config.php



- `template_path_stack`  
Déclare le répertoire qui contient les vues
- Convention du repertoire view  
[module] / [contrôleur] / [action].phtml  
Le nom des répertoires et des fichiers est à la convention spinal-case. (AddressBook devient address-book)
- `template_map`  
Permet de changer la convention pour certaines vues.



```
'view_manager' => [  
    'template_path_stack' => [  
        __DIR__ . '/../view'  
    ]  
],
```

# Vues - Appel d'une vue



- Depuis un contrôleur  
Retourner une instance de ViewModel (voir aussi JsonModel et FeedModel pour les Web Services).  
Le constructeur de ViewModel reçoit en entrée un tableau associatif dont les clés deviendront des variables dans le contexte de la vue.
- Vous pouvez également retourner directement un tableau associatif, le framework l'injectera dans un objet ViewModel.

```
public function showAction()  
{  
    $contact = new Contact("Romain", "Bohdanowicz");  
  
    return new ViewModel (array(  
        "contact" => $contact  
    ));  
}
```

# Vues - Appel d'une vue



- Depuis une autre vue

```
<body>

<?php echo $this->partial('layout/menu.phtml'); ?>

<!-- Partial cloisonne les variables, pour transmettre une valeur : -->
<?php echo $this->partial('layout/menu.phtml', array('variable' => 'valeur')); ?>

<div class="container">
```

- Eviter les inclusions du type

```
<?php echo $this->partial('layout/header.phtml'); ?>

<!-- Code de l'action -->

<?php echo $this->partial('layout/footer.phtml'); ?>
```

- Le code qui commence dans le fichiers header.phtml se termine dans footer.phtml. Donc impossible de vérifier la syntaxe ou de reformater le code.
- La méthode partial est ce qu'on appelle une aide de vue (méthode permettant de simplifier l'écriture de la vue). Elle est attaché à l'objet contenant la vue par le framework.

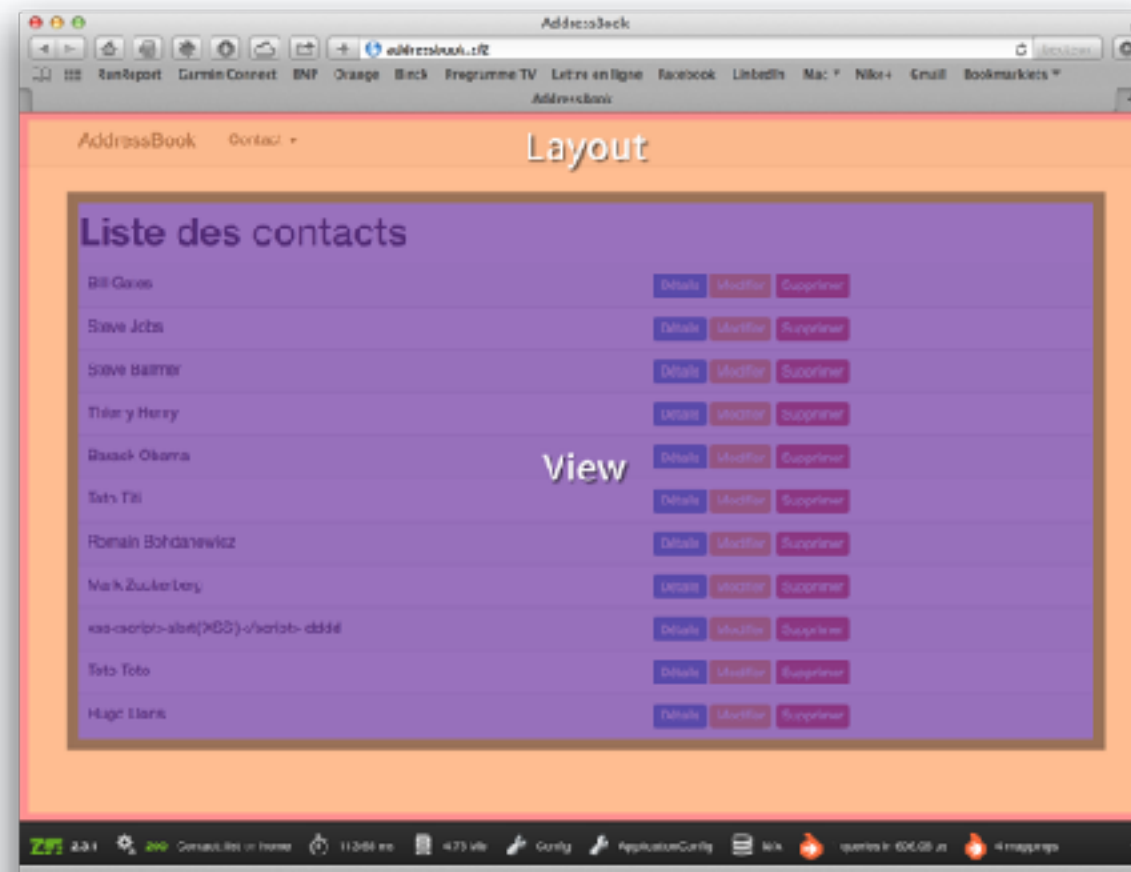


- ▶ Two Step View

Le Layout est une implémentation du Design Pattern Two Step View : <http://martinfowler.com/eaCatalog/twoStepView.html>

Le rendu de la vue se fait en premier puis est injecté dans le rendu du layout.  
C'est le framework qui s'occupe du layout, la vue n'a pas à connaître son existence.

- ▶ Pour activer le layout d'un module, il faut définir un fichier layout/layout.phtml





- Inclusion de la vue

```
<?php echo $this->content; ?>
```

- Aide de vue basePath

Permet de faire en sorte que les liens ne dépendent pas de l'arborescence de la racine du site. L'application pourra donc être déployer de la même façon sur <http://monsite.com/> ou <http://monsite.com/sous-repertoire/>

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>AddressBook</title>
  <link rel="stylesheet" href="<?php echo $this->basePath('css/bootstrap.css'); ?>">
</head>
<body>

<?php echo $this->partial('layout/menu.phtml'); ?>

<div class="container">

  <?php echo $this->content; ?>

</div>

<script src="<?php echo $this->basePath('js/jquery.min.js'); ?>"></script>
<script src="<?php echo $this->basePath('js/bootstrap.js'); ?>"></script>
</body>
</html>
```



## ▸ Layout du module Application

```
<?php echo $this->doctype(); ?>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <?php echo $this->headTitle('ZF2 '. $this->translate('Skeleton Application'))->setSeparator(' - ')-
>setAutoEscape(false) ?>

    <?php echo $this->headMeta()
      ->appendName('viewport', 'width=device-width, initial-scale=1.0')
      ->appendHttpEquiv('X-UA-Compatible', 'IE=edge'); ?>

    <?php echo $this->headLink(array('rel' => 'shortcut icon', 'type' => 'image/vnd.microsoft.icon', 'href' =>
$this->basePath() . '/img/favicon.ico'))
      ->prependStylesheet($this->basePath() . '/css/style.css'); ?>

    <?php echo $this->headScript()
      ->prependFile($this->basePath() . '/js/bootstrap.min.js')
      ->prependFile($this->basePath() . '/js/jquery.min.js'); ?>
  </head>
  <body>
    <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
      <div class="container">
        <div class="navbar-header">
          <a class="navbar-brand" href="<?php echo $this->url('home') ?>">&nbsp;<?php echo $this->translate('Skeleton Application') ?
></a>
        </div>
        <ul class="nav navbar-nav">
          <li class="active"><a href="<?php echo $this->url('home') ?>"><?php echo $this->translate('Home') ?
></a></li>
        </ul>
      </div>
    </nav>
    <div class="container">
      <?php echo $this->content; ?>
    </div> <!-- /container -->
    <?php echo $this->inlineScript() ?>
  </body>
</html>
```



- Doctype  
Permet d'utiliser un doctype défini dans un fichier de configuration
- HeadTitle, HeadMeta, HeadScript, HeadStyle, HeadLink, InlineScript  
Aides de vue à rendre au niveau du layout et permettant d'être personnalisé au niveau d'une vue.
- FlashMessenger  
Pour afficher un message directement depuis une vue.
- Partial  
Permet d'inclure un fragment de page.
- Url  
Permet de générer une URL à partir d'une route.
- Autres aides de vues et exemples :  
<http://framework.zend.com/manual/current/en/modules/zend.view.helpers.html>



- Créer sa propre aide de vue

```
<?php
namespace AddressBook\View\Helper;

use Zend\Form\Form;

class BootstrapFormGroup extends \Zend\Form\View\Helper\AbstractHelper
{
    public function __invoke(Form $form, $elementName, $elementPlugin = 'formElement')
    {
        if ($form->getMessages($elementName)) {
            $output = '<div class="form-group has-error">';
        } else {
            $output = '<div class="form-group">';
        }

        $output .= // ...

        $output .= '</div>';

        return $output;
    }
}
```

```
<?php
// module.config.php
return array(
    // ...
    'view_helpers' => array(
        'invokables' => array(
            'bootstrapFormGroup' =>
            'AddressBook\View\Helper\BootstrapFormGroup',
        )
    ),
    // ...
);
```



# Vues - Moteurs de templates



- Templates

Les utilisateurs familiers d'autres Frameworks ou langage de templates pourront les utiliser via des modules tiers.

- Twig

Module : zf-commons/zfc-twig

<https://github.com/ZF-Commons/ZfcTwig>

- Smarty

Module : murganikolay/smarty-module

<https://github.com/MurgaNikolay/SmartyModule>

- Mustache

Module : widmogrod/zf2-mustache-module

<https://github.com/widmogrod/zf2-mustache-module>



# Model



Zend\Db



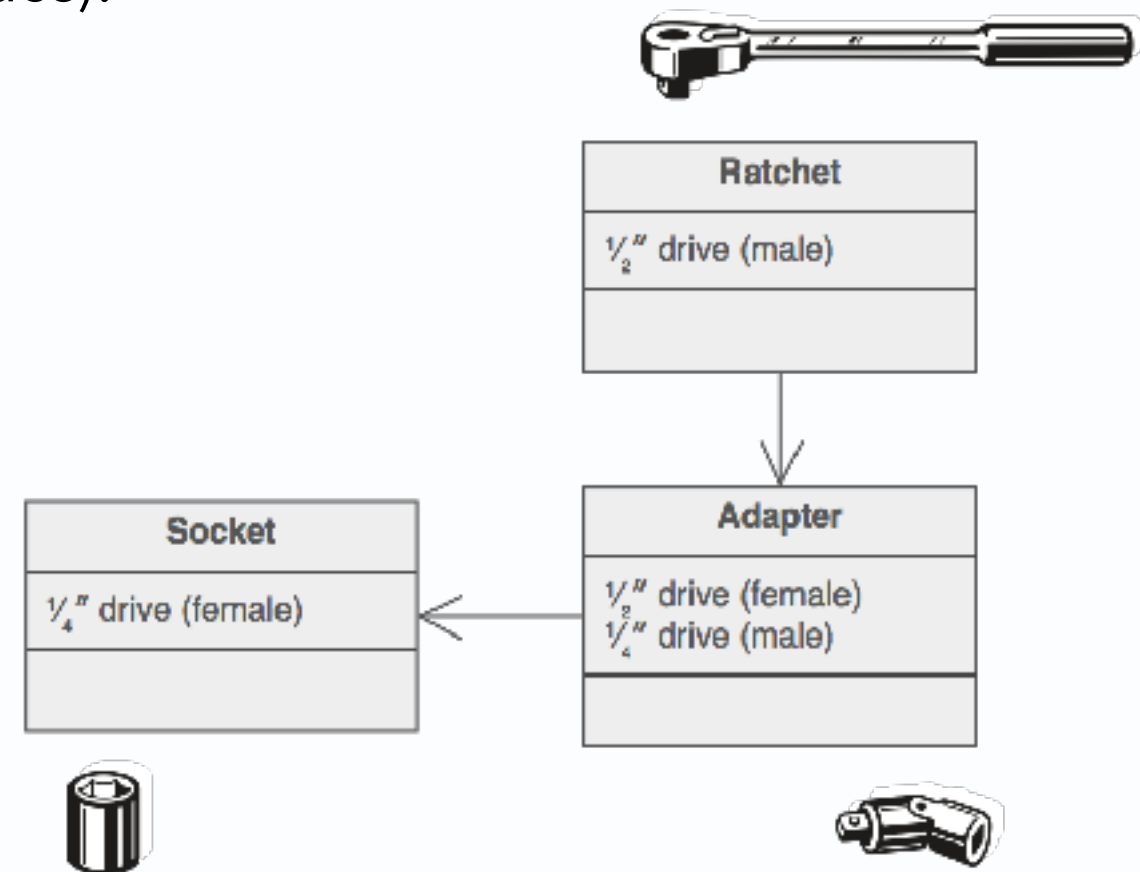
▸ Design Pattern Adapter (Adaptateur)

Le rôle d'une classe Adapter est de rendre compatible 2 classes qui ne le serait pas normalement.

▸ Dans Zend, la classe Adapter dépend d'une classe Driver (qui implémente l'interface DriverInterface).

▸ Les drivers préexistants :

- IbmDb2
- Mysqli
- Oci8
- Pdo
- Pgsq
- Sqlsrv



# Zend\Db - Configuration



- Pour la récupérer dans un contrôleur on utilise une fabrique et le gestionnaire de services (Service Manager). L'adaptateur permet ensuite d'exécuter des requêtes SQL.

```
// config/autoload/db.local.php
return array(
    'db' => array(
        'username' => 'root',
        'password' => '',
    )
);
```

```
// config/autoload/db.global.php
return array(
    'db' => array(
        'driver' => 'Pdo_Mysql',
        'hostname' => 'localhost',
        'database' => 'address_book',
        'charset' => 'UTF8',
    )
);
```

```
public function indexAction()
{
    $adapterFactory = new AdapterServiceFactory();
    $adapter = $adapterFactory->createService($this->serviceLocator);

    $result = $adapter->query("SHOW DATABASES")->execute();

    while($row = $result->next()) {
        var_dump($row["Database"]);
    }
}
```

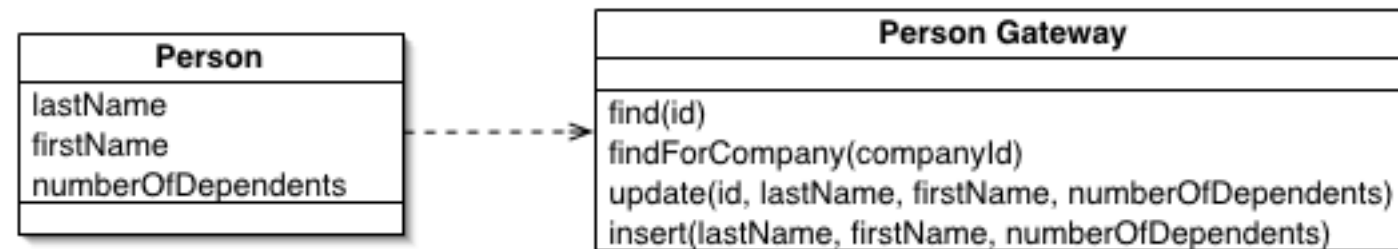
# Zend\Db - Table Data Gateway



- Table Data Gateway

Un objet qui contient toute les requêtes vers une table d'une base relationnelle. Une instance gère tous les enregistrements.

<http://martinfowler.com/eaCatalog/tableDataGateway.html>



- Zend\Db\TableGateway\TableGateway

Une implementation de ce design pattern. Permet d'interagir de manière object avec les enregistrements d'une table (SELECT, INSERT, UPDATE, DELETE).

Intègre une protection contre les injections SQL, si la requête contient des paramètres (WHERE, VALUES, SET...), alors la requête est préparée.



## ► Lecture

```
public function indexAction()
{
    $adapterFactory = new AdapterServiceFactory();
    $adapter = $adapterFactory->createService($this->serviceLocator);

    $contactGateway = new TableGateway("contact", $adapter);

    var_dump($contactGateway->select()->toArray());
}
```

## ► Ecriture

```
public function addAction()
{
    $adapterFactory = new AdapterServiceFactory();
    $adapter = $adapterFactory->createService($this->serviceLocator);

    $contactGateway = new TableGateway("contact", $adapter);

    $post = $this->params()->fromPost();

    $contactGateway->insert($post);
}
```



- ResultSet sous forme d'objet

```
public function indexAction()
{
    $id = $this->params()->fromQuery("id");

    $adapterFactory = new AdapterServiceFactory();
    $adapter = $adapterFactory->createService($this->serviceLocator);

    $resultSetPrototype = new ResultSet();
    $resultSetPrototype->setArrayObjectPrototype(new Contact());
    $contactGateway = new TableGateway("contact", $adapter, null, $resultSetPrototype);

    var_dump($contactGateway->select(array("id" => $id))->current());
}
```

- Dans cet exemple, Contact est un objet Entité qui implémente ArraySerializableInterface et ses méthodes exchangeArray(array) et getArrayCopy()

```
public function exchangeArray(array $array)
{
    foreach ($array as $key => $value) {
        if (property_exists($this, $key)) {
            $this->$key = $value;
        }
    }
}

public function getArrayCopy()
{
    return get_object_vars($this);
}
```





## ▸ Zend\Db\Select

Permet de construire une requête SQL de manière Objet.

```
public function listeAction()
{
    $page = $this->params()->fromQuery('page', 0);
    $nbResults = $this->params()->fromQuery('nbResults', 0);

    $adapterFactory = new AdapterServiceFactory();
    $adapter = $adapterFactory->createService($this->serviceLocator);

    $select = (new Select())
        ->from('contact')
        ->order('prenom');

    if ($nbResults) {
        $select->limit($nbResults);
    }

    if ($page) {
        $select->offset($page);
    }

    var_dump($select->getSqlString($adapter->getPlatform()));
}
```



# Doctrine

# Doctrine - Introduction



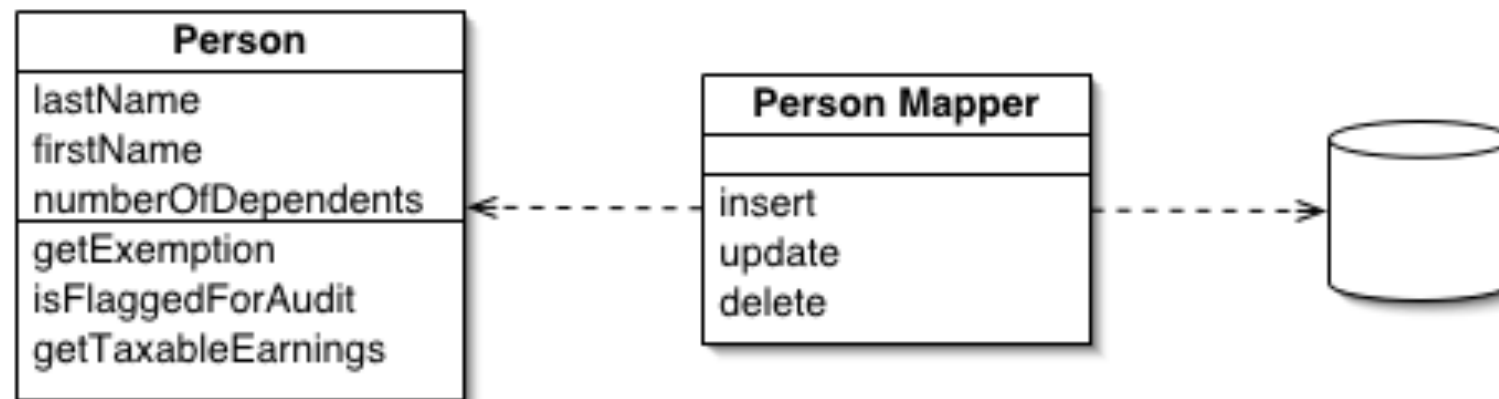
- DBAL, ORM, ODM

Doctrine est une bibliothèque qui intègre un DBAL (Database Abstraction Layer), un ORM (Object Relational Mapping) et un ODM (Object Document Mapper).

- Data Mapper

Doctrine ORM est une implémentation du Design Pattern Data Mapper, c'est un composant qui permet de communiquer avec une base de données de manière objet. Il est inspiré de la bibliothèque Hibernate en Java.

<http://martinfowler.com/eaCatalog/dataMapper.html>



- Installation via composer

```
"require": {
    // ...
    "doctrine/doctrine-orm-module": "0.8.*"
},
```

# Doctrine - Configuration



## ▸ DBAL, ORM, ODM

```
<?php
// doctrine.global.php
return array (
    'doctrine' => array (
        'connection' => array (
            // default connection name
            'orm_default' => array (
                'driverClass' => 'Doctrine\DBAL\Driver\PDOMySql\Driver',
                'params' => array (
                    'host' => '127.0.0.1',
                    'port' => '3306',
                    'dbname' => 'address_book'
                )
            )
        )
    ),
    'driver' => array (
        // defines an annotation driver with two paths, and names it `my_annotation_driver`
        'AddressBook_Driver' => array (
            'class' => 'Doctrine\ORM\Mapping\Driver\AnnotationDriver',
            'cache' => 'array',
            'paths' => array (
                __DIR__ . '/../../module/AddressBook/src/AddressBook/Entity'
            )
        ),
        'orm_default' => array (
            'drivers' => array (
                'AddressBook\Entity' => 'AddressBook_Driver'
            )
        )
    )
);
```

```
<?php
// doctrine.local.php
return array(
    'doctrine' => array(
        'connection' => array(
            // default connection name
            'orm_default' => array(
                'params' => array(
                    'user' => 'root',
                    'password' => ''
                )
            )
        )
    )
);
```

```
<?php
// application.config.php
return array(
    'modules' => array(
        'ZendDeveloperTools',
        'DoctrineModule',
        'DoctrineORMModule',
        'ZfcBase',
        'ZfcUser',
        'ZfcUserDoctrineORM',
        'Application',
        'AddressBook',
    ),
    // ...
);
```



## ▸ DoctrineModule Command Line Interface

L'installation de doctrine-module ajoute une programme en ligne de commande pour générer du code :

vendor/bin/doctrine-module (UNIX)

vendor\bin\doctrine-module.bat (Windows)

```
> vendor/bin/doctrine-module
DoctrineModule Command Line Interface version 0.8.0
...
Available commands:
  help                Displays help for a command
  list                Lists commands
dbal
  dbal:import          Import SQL file(s) directly to Database.
  dbal:run-sql         Executes arbitrary SQL directly from the command line.
orm
  orm:clear-cache:metadata Clear all metadata cache of the various cache drivers.
  orm:clear-cache:query   Clear all query cache of the various cache drivers.
  orm:clear-cache:result  Clear all result cache of the various cache drivers.
  orm:convert-dl-schema   Converts Doctrine 1.X schema into a Doctrine 2.X schema.
  orm:convert-mapping     Convert mapping information between supported formats.
  orm:convert:dl-schema   Converts Doctrine 1.X schema into a Doctrine 2.X schema.
  orm:convert-mapping     Convert mapping information between supported formats.
  orm:ensure-production-settings Verify that Doctrine is properly configured for a production environment.
  orm:generate-entities   Generate entity classes and method stubs from your mapping information.
  orm:generate-proxies    Generates proxy classes for entity classes.
  orm:generate-repositories Generate repository classes from your mapping information.
  orm:generate:entities   Generate entity classes and method stubs from your mapping information.
  orm:generate:proxies    Generates proxy classes for entity classes.
  orm:generate:repositories Generate repository classes from your mapping information.
  orm:info               Show basic information about all mapped entities
  orm:run-dql             Executes arbitrary DQL directly from the command line.
  orm:schema-tool:create  Processes the schema and either create it directly on EntityManager Storage Connection or
generate the SQL output.
  orm:schema-tool:drop    Drop the complete database schema of EntityManager Storage Connection or generate the
corresponding SQL output.
  orm:schema-tool:update  Executes (or dumps) the SQL needed to update the database schema to match the current mapping
metadata.
  orm:validate-schema     Validate the mapping files.
```

# Doctrine - Principales commandes



- Générer le mapping depuis une base de données

Le mapping peut être au format XML, YAML, Annotations ou PHP

```
./vendor/bin/doctrine-module orm:convert-mapping --from-database --  
namespace=Application\Entity\ annotation module/Application/src
```

- ```
./vendor/bin/doctrine-module orm:convert-mapping --from-database --  
filter="Contact|Societe" --namespace=Application\Entity\ annotation module/  
Application/src
```

- Générer les entités depuis le mapping

Si le mapping est au format annotation alors Doctrine est obligé de créer les entités et leurs propriétés, cependant il manque constructeur et accesseurs.

```
vendor/bin/doctrine-module orm:generate-entities module/Application/src
```

- Générer la base de données depuis le mapping

Pour vérifier que la requête SQL est correcte

```
vendor/bin/doctrine-module orm:schema-tool:update --dump-sql
```

Pour exécuter la requête SQL

```
vendor/bin/doctrine-module orm:schema-tool:update --force
```



- `@ORM\Entity`  
Pour déclarer la classe persistente
- `@ORM\Table(name="contact")`  
Pour déclarer à quelle table est associée cette entité  
Permet également de définir des index et contrainte unique.

```
<?php

namespace AddressBook\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Contact
 *
 * @ORM\Table(name="contact", uniqueConstraints={@ORM\UniqueConstraint(name="email_UNIQUE", columns={"email"})},
indexes={@ORM\Index(name="fk_contact_societe_idx", columns={"societe_id"}), @ORM\Index(name="fk_contact_membre1_idx",
columns={"membre_id"})})
 * @ORM\Entity
 */
class Contact
{
}
}
```



## ▸ @ORM\Column

Pour lier une propriété à une colonne, attributs **name** id le nom de colonne est différent, **length** taille, **nullable** (true/false), **type** parmi :

- **string** (string <> VARCHAR)
- **integer** (int <> INT)
- **smallint** (int <> SMALLINT)
- **bigint** (string <> BIGINT)
- **boolean** (boolean)
- **decimal** (float <> DECIMAL avec des options precision et scale)
- **date** (\DateTime <> DATETIME)
- **time** (\DateTime <> TIME)
- **datetime** (\DateTime <> DATETIME/TIMESTAMP)
- **text** (string <> TEXT), object (serialize(object) <> unserialize(TEXT))
- **array** (serialize(object) <> unserialize(TEXT))
- **float** (float <> FLOAT (séparateur décimale .))

## ▸ Il est également possible de définir ses propres types (peut-être utiliser pour les ENUM) :

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/cookbook/mysql-enums.html>

## ▸ @ORM\Id

Si la colonne est la primaire.

## ▸ @ORM\GeneratedValue(strategy="IDENTITY")

Si la clé primaire est générée par le SGBDR (strategy="IDENTITY" pour MySQL/SQLite/MSSQL), (strategy="SEQUENCE" pour PostgreSQL/Oracle), (strategy="AUTO" pour être portable)



# Doctrine - Annotations



## ▸ Exemple :

```
/**
 * @var integer
 *
 * @ORM\Column(name="id", type="integer", nullable=false)
 * @ORM\Id
 * @ORM\GeneratedValue(strategy="IDENTITY")
 */
private $id;

/**
 * @var string
 *
 * @ORM\Column(name="prenom", type="string", length=45, nullable=false)
 */
private $prenom;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="date_naissance", type="date", nullable=true)
 */
private $dateNaissance;

/**
 * @var int
 *
 * @ORM\Column(name="taille", type="integer", nullable=true)
 */
private $taille;
```



## ▸ Associations

Doctrine ORM permet de définir directement les associations entre les entités et s'occupera de faire la plupart des jointures et insertions multiples automatiquement.

- OneToOne  
Relation 1..1
- OneToMany  
Relation 1..n du côté 1
- ManyToOne  
Relation 1..n du côté n (du côté de la clé étrangère)
- ManyToMany  
Relation n..m

## ▸ Unidirectionnelle / Bidirectionnelle

La relation peut-être unidirectionnelle (une entité en connaît une autre mais l'inverse n'est pas vrai) ou bidirectionnelle (chaque entité connaît l'autre).

Les relations bidirectionnelles doivent déclarer la relation opposées (mappedBy/inversedBy).



## ▸ Exemple AddressBook\Entity>Contact :

```
class Contact
{
    /**
     * @var \Application\Entity\Membre
     *
     * @ORM\OneToOne(targetEntity="AddressBook\Entity\Membre")
     * @ORM\JoinColumns({
     *     @ORM\JoinColumn(name="membre_id", referencedColumnName="id")
     * })
     */
    private $membre;

    /**
     * @var \Application\Entity\Societe
     *
     * @ORM\ManyToOne(targetEntity="Application\Entity\Societe", inversedBy="contacts")
     * @ORM\JoinColumns({
     *     @ORM\JoinColumn(name="societe_id", referencedColumnName="id")
     * })
     */
    private $societe;

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\ManyToMany(targetEntity="Application\Entity\Association")
     * @ORM\JoinTable(name="adhesion",
     *     joinColumns={
     *         @ORM\JoinColumn(name="contact_id", referencedColumnName="id")
     *     },
     *     inverseJoinColumns={
     *         @ORM\JoinColumn(name="association_id", referencedColumnName="id")
     *     }
     * )
     */
    private $associations;
}
```

# Doctrine - Annotations



## ▸ Exemple AddressBook\Entity\Societe :

```
class Societe
{
    // ...

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\OneToMany(targetEntity="Application\Entity>Contact", mappedBy="societe")
     */
    private $contacts;

    /**
     * Constructor
     */
    public function __construct()
    {
        $this->contacts = new \Doctrine\Common\Collections\ArrayCollection();
    }
}
```

# Doctrine - EntityManager et EntityRepository



- Doctrine\ORM\EntityManager  
L'objet responsable de la persistance des entités.
- Doctrine\ORM\EntityRepository  
L'objet responsable de la récupération des entités.
- Portabilité vers Doctrine\ODM  
Pour être portable vers Doctrine\ODM, utiliser les interfaces Doctrine\Common\Persistence\ObjectManager et Doctrine\Common\Persistence\ObjectRepository dans les DocBlocks.

```
class ContactController extends AbstractActionController
{
    // ...

    /**
     * @return \Doctrine\Common\Persistence\ObjectManager
     */
    public function getEntityManager()
    {
        return $this->getServiceLocator()->get('Doctrine\ORM\EntityManager');
    }

    /**
     * @return \Doctrine\Common\Persistence\ObjectRepository
     */
    public function getRepository()
    {
        return $this->getEntityManager()->getRepository('AddressBook\Entity>Contact');
    }
}
```



## ▸ Doctrine\ORM\EntityManager

Principales méthodes :

- `persist($entity)`  
Ajoute l'entité à la liste des entités persistantes (INSERT ou UPDATE)
- `remove($entity)`  
Retire l'entité de la liste des entités persistantes (DELETE)
- `flush()`  
Exécute les requêtes SQL les plus optimisées correspondant aux précédents appels à `persist` et `remove`.

```
public function addAction()
{
    // ...
    $em = $this->getEntityManager();
    $em->persist($contact);
    $em->flush();
    // ...
}

public function deleteAction()
{
    // ...
    $em = $this->getEntityManager();
    $em->remove($contact);
    $em->flush();
    // ...
}
```

# Doctrine - EntityManager



## ▶ Doctrine\ORM\EntityRepository

Principales méthodes :

- `find($id)`  
Récupère une \$entité avec sa clé primaire.
- `findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)`  
Récupère une liste d'entités avec différents critères
- `findAll()`  
Récupère toutes les entités sans critères et sans tri (équivalent à `findBy(array())`)
- `findOneBy(array $criteria, array $orderBy = null)`  
Récupère une entité avec différents critères

```
public function listAction()
{
    $listeContacts = $this->getRepository()->findBy(array(), array('prenom' => 'ASC', 'nom' => 'ASC'));

    return array(
        'listeContacts' => $listeContacts,
    );
}

public function showAction()
{
    $contact = $this->getRepository()->find($this->params("id"));

    return new ViewModel(
        array(
            "contact" => $contact
        )
    );
}
```

# Doctrine - Requêtes « complexes »



- extends Doctrine\ORM\EntityRepository

Parfois les requêtes ne peuvent pas s'écrire où ne sont pas bien optimisés, il faut alors créer une classe Repository qui héritera de Doctrine\ORM\EntityRepository  
Les requêtes peuvent alors s'écrire :

- En SQL

Il faudra alors expliquer à Doctrine comment cette requête se traduit en entité

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/native-sql.html>

- Avec le QueryBuilder

Equivalent de Zend\Db\Sql

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html>

- En DQL

Language propre à Doctrine proche de SQL mais manipulant des entités.

- Création du Repository

Modifier l'annotation @ORM\Entity d'une entité pour :

```
@ORM\Entity(repositoryClass="AddressBook\Entity\Repository\ContactRepository")
```

Exécuter la commande :

```
vendor/bin/doctrine-module orm:generate:repositories module/AddressBook/src
```



# Doctrine - Requêtes « complexes »



## ▸ Exemple

Dans showAction du contrôleur Contact, nous requérons une entité par sa clé primaire. Dans la vue nous demandons d'accéder à sa Société liée.

Par défaut Doctrine fait 2 requêtes SQL, nous aimerions en faire 1 seule requêtes avec une jointure.

```
// AddressBook/Controller/ContactController
public function showAction()
{
    $id = $this->params("id");

    $contact = $this->getRepository()->find($id);

    return new ViewModel(
        array(
            "contact" => $contact
        )
    );
}
```

```
<!-- view/address-book/contact/show.phtml -->
<?=$this->escapeHtml($contact->getSociete()->getNom())?>
```

» DoctrineORMModule

» Queries for doctrine.sql\_logger\_collector.orm\_default

SQL:

```
SELECT t0.id AS id1, t0.prenom AS prenom2, t0.nom AS nom3, t0.telephone AS
telephone4, t0.email AS email5, t0.membre_id AS membre_id6, t0.societe_id AS
societe_id7 FROM contact t0 WHERE t0.id = ?
```

Params: 0 => string '1' (length=1)

Types: 0 => string 'integer' (length=7)

Time: 0.00042605400085449

SQL:

```
SELECT t0.id AS id1, t0.nom AS nom2, t0.ville AS ville3 FROM societe t0
WHERE t0.id = ?
```

Params: 0 => int 3



2 queries in 876.19 µs



4 mappings

# Doctrine - Requêtes « complexes »



## ▸ Exemple

Avec le Repository, on obtient une seule requête (temps d'exécution divisé par 2).

```
// AddressBook/Entity/Repository/ContactRepository

class ContactRepository extends EntityRepository
{
    public function findWithSociete($id)
    {
        $dql = "SELECT c, s
                FROM AddressBook\Entity\Contact c
                LEFT JOIN c.societe s
                WHERE c.id = :id";

        return $this->getEntityManager()->createQuery($dql)
            ->setParameter("id", $id)
            ->getSingleResult();
    }
}
```

```
// AddressBook/Controller/ContactController

public function showAction()
{
    $id = $this->params("id");

    $contact = $this->getRepository()->findWithSociete($id);

    return new ViewModel(
        array(
            "contact" => $contact
        )
    );
}
```

```
» DoctrineORMModule
» Queries for doctrine.sql_logger_collector.orm_default

SQL:
SELECT c0_.id AS id0, c0_.prenom AS prenom1, c0_.nom AS nom2,
c0_.telephone AS telephone3, c0_.email AS email4, s1_.id AS id5, s1_.nom AS
nom6, s1_.ville AS ville7, c0_.membre_id AS membre_id8, c0_.societe_id AS
societe_id9 FROM contact c0_ LEFT JOIN societe s1_ ON c0_.societe_id =
s1_.id WHERE c0_.id = ?

Params:          0 => string '1' (length=1)
Types:           0 => int 2
Time:            0.0004878044128418

➔ 1 queries in 487.80 µs ➔ 4 mappings
```



Zend\Form



- Présentation

Les formulaires permettent de gérer les formulaires HTML de manière objet, mais également au sens validation de données entrantes (par exemple dans Web Service).

- Changements depuis ZF1

Dans Zend Framework 1, les formulaires avaient trop de responsabilités, ils est désormais beaucoup plus découplé :

- Zend\Form : le formulaire lui-même
- Zend\InputFilter : validation
- Zend\Form\View : rendu

# Zend\Form - Création d'un formulaire



## ▸ Création

```
<?php

namespace AddressBook\Form;

use Zend\Form\Element;
use Zend\Form\Form;

class ContactForm extends Form
{
    public function __construct($em)
    {
        parent::__construct("contact");

        $this->add(
            array(
                "name" => "prenom",
                "options" => array(
                    "label" => "Prénom",
                ),
                "attributes" => array(
                    "type" => "text"
                ),
            )
        );
    }
}
```

# Zend\InputFilter - Validation des entrées



## ▸ Validation

```
<?php

namespace AddressBook\InputFilter;

use Zend\Filter\StringTrim;
use Zend\InputFilter\InputFilter;
use Zend\InputFilter\Input;
use Zend\Validator\NotEmpty;
use Zend\Validator\StringLength;

class ContactInputFilter extends InputFilter
{
    public function __construct()
    {
        // Prénom
        $input = new Input("prenom");

        $validator = new NotEmpty();
        $validator->setMessage("Le prénom est obligatoire",
            NotEmpty::IS_EMPTY);
        $input->getValidatorChain()->attach($validator);

        $validator = new StringLength();
        $validator->setMax(45);
        $validator->setMessage("Le prénom ne doit pas dépasser %max% caractères",
            StringLength::TOO_LONG);
        $input->getValidatorChain()->attach($validator);

        $filter = new StringTrim();
        $input->getFilterChain()->attach($filter);

        $this->add($input);
    }
}
```



## ▸ Un contrôleur

```
public function addAction()
{
    $form = new ContactForm($this->getEntityManager());

    if ($this->request->isPost()) {
        $form->setInputFilter(
            new ContactAddInputFilter($this->getRepository())
        );
        $form->setData($this->request->getPost());

        if ($form->isValid()) {
            $contact = new Contact();
            $hydrator = new DoctrineObject($this->getEntityManager());
            $hydrator->hydrate($form->getData(), $contact);

            $em = $this->getEntityManager();
            $em->persist($contact);
            $em->flush();

            $this->flashMessenger()->addSuccessMessage('Le contact a bien été ajouté');

            return $this->redirect()->toRoute('home');
        }
    }

    return array(
        'form' => $form,
    );
}
```



- ▶ Rapide :

```
<?php echo $this->form($form); ?>
```

- ▶ Détaillé :

```
<h1>Ajouter un contact</h1>
```

```
<?php echo $this->form()->openTag($form); ?>
```

```
<?php echo $this->formLabel($form->get("prenom")); ?>
```

```
<?php echo $this->formElement($form->get("prenom")); ?>
```

```
<?php echo $this->formElementErrors($form->get("prenom")); ?>
```

```
<button type="submit">Ajouter</button>
```

```
<?php echo $this->form()->closeTag(); ?>
```





# Service Manager

# Service Manager - Composition



## ▸ Composition

Une composition est un type d'association forte entre 2 objet. La destruction d'un objet entrainerait la destruction de l'objet associé.

Exemple : Un objet Tasse est composée de Café

```
<?php
namespace EspressoComposition;

class Cafe
{
    protected $variete;
    protected $provenance;

    public function __construct($provenance, $variete)
    {
        $this->provenance = $provenance;
        $this->variete = $variete;
    }
}
```

```
<?php
namespace EspressoComposition;

class Tasse
{
    protected $contenu;

    public function __construct() {
        $this->contenu = new Cafe("Arabica", "Mexique");
    }
}
```

```
<?php
require_once 'autoload.php';

$tasseDeCafe = new \EspressoComposition\Tasse();
```

## ▸ Mauvaise Pratique

La composition est désormais considéré comme une mauvaise pratique.

Premièrement la classe Tasse n'est très réutilisable, elle ne peut contenir que du café. De plus il n'est pas possible d'écrire d'écrire un test unitaire de Tasse, puisqu'il faudrait en même temps tester Café.

Globalement il faut essayer de proscrire l'utilisation de new il l'intérieur d'une classe (à l'exception des Values Objects (DateTime, ArrayObject, etc...))

# Service Manager - Injection de Dépendance



## ▸ Solution

La solution est simple, pour éviter le new dans cette classe nous allons injecter la dépendance.

```
<?php
namespace EspressoInjection;

interface Liquide {
}
```

```
<?php
namespace EspressoInjection;

class Cafe implements Liquide
{
    protected $variete;
    protected $provenance;

    public function __construct($provenance, $variete)
    {
        $this->provenance = $provenance;
        $this->variete = $variete;
    }
}
```

```
<?php
namespace EspressoInjection;

class Tasse
{
    protected $contenu;

    public function __construct(Liquide $contenu) {
        $this->contenu = $contenu;
    }
}
```

```
<?php
require_once 'autoload.php';

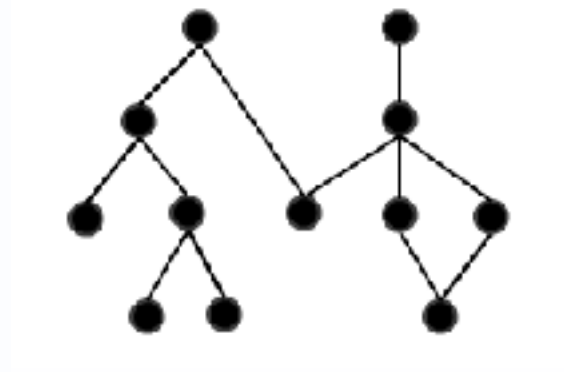
n' $cafe = new \EspressoInjection\Cafe();
    $tasseDeCafe = new \EspressoInjection\Tasse($cafe);
```

# Service Manager - Injection de Dépendance



- Conteneur d'injection de dépendance (DIC)

Le problème lorsqu'on injecte les dépendances est qu'on peut parfois se retrouver avec des dépendances complexes :



- Dans ce cas il devient utile d'utiliser un conteneur d'injection de dépendance qu'on aura configuré au préalable. En PHP il existe quelques DIC connus :

- Pimple par Fabien Potencier
- Dice par Tom Butler
- PHP-DI par Matthieu Napoli
- Symfony\Container intégré Symfony2
- Zend\Di & Zend\ServiceManager intégrés ZF 2

- Zend\Di est très simple à configurer mais peu performant (utilise la Reflection), Zend\ServiceManager est le plus utilisé.

Documentation : <http://framework.zend.com/manual/current/en/modules/zend.service-manager.quick-start.html>

# Service Manager - Injection de Dépendance



- Configuration d'une dépendance

- module.config.php

Clés possibles :

- **abstract\_factories**, valeurs contenant des noms de classes qui implémentent `Zend\ServiceManager\AbstractFactoryInterface` ou un callback PHP
- **aliases**, tableau associatif contenant nom de l'alias (clé) et service aliasé (valeur)
- **factories**, valeurs contenant des noms de classes qui implémentent `Zend\ServiceManager\FactoryInterface` ou un callback PHP
- **invokables**, valeurs contenant des noms de classes à instancier directement (pas de paramètres passé au constructeur)
- **services**, les valeurs sont des objets (comparable à `Zend_Registry` dans ZF1)
- **shared**, permet de spécifier quels services ne sont pas partagés (il le sont par défaut)

- Module.php

Implémenter l'interface `Zend\ModuleManager\Feature\ServiceProviderInterface` et sa méthode `getServiceConfig`

# Service Manager - Injection de Dépendance



- Exemple  
module.config.php

```
'service_manager' => array(
    'factories' => array(
        'Hydrator\DoctrineObject' => function(\Zend\ServiceManager\ServiceManager $sm) {
            return new DoctrineObject($sm->get('Doctrine\ORM\EntityManager'));
        },
        'AddressBook\Form\ContactForm' => function(\Zend\ServiceManager\ServiceManager $sm) {
            $form = new \AddressBook\Form\ContactForm($sm->get('Doctrine\ORM\EntityManager'));
            return $form;
        },
        'AddressBook\Form\ContactAddForm' => function(\Zend\ServiceManager\ServiceManager $sm) {
            $form = $sm->get('AddressBook\Form\ContactForm');
            $em = $sm->get('Doctrine\ORM\EntityManager');
            $repository = $em->getRepository('AddressBook\Entity\Contact');
            $inputFilter = new \AddressBook\InputFilter\ContactAddInputFilter($repository);
            $form->setInputFilter($inputFilter);
            return $form;
        },
    ),
    'invokables' => array(
        'AddressBook\Entity\Contact' => 'AddressBook\Entity\Contact'
    )
)
```

# Service Manager - Injection de Dépendance



## ▸ Exemple Contrôleur

```
public function addAction()
{
    $form = $this->getServiceLocator()->get('AddressBook\Form>ContactAddForm');

    if ($this->request->isPost()) {
        $form->setData($this->request->getPost());

        if ($form->isValid()) {
            $contact = $this->getServiceLocator()->get('AddressBook\Entity>Contact');
            $hydrator = $this->getServiceLocator()->get('Hydrator\DoctrineObject');
            $hydrator->hydrate($form->getData(), $contact);

            $em = $this->getEntityManager();
            $em->persist($contact);
            $em->flush();

            $this->flashMessenger()->addSuccessMessage('Le contact a bien été ajouté');

            return $this->redirect()->toRoute('home');
        }
    }

    return array(
        'form' => $form,
    );
}
```



# Event Manager



# Event Manager - Introduction



- Zend Framework 2 est event-driven
- La boucle MVC est réalisée via un ensemble d'événements
  - Event : un évènement est toujours nommé ("dispatch", "render"...). Un objet peut lancer un évènement ( trigger ).
  - Listener : un listener est un objet qui écoute ( listen ) un ou plusieurs évènements.
- Il permet de simuler les "hooks" de Zend Framework 1 (postDispatch, preDispatch...)
- Utile dans certains cas pour rendre l'architecture plus flexible
- Peut également compliquer la lecture du programme (programmation spaghetti)

# Event Manager - Exemple



## ▸ Exemple

```
namespace Application\Service;

class TweetService
{
    public function sendTweet($content)
    {
        // Envoyer le tweet via l'API Twitter ...

        // Envoyer un mail
        $mail = new Mail();
        $transport = new Smtplib();
        $transport->send($mail);
    }
}
```

```
namespace Application\Service;

class TweetService
{
    public function sendTweet($content)
    {
        // Envoyer le tweet via l'API Twitter ...

        // Envoyer un mail
        $eventManager->trigger('sendTweet', array('content' => $content));
    }
}
```



- Événements MVC

Utiliser les constantes de la classe `Zend\Mvc\MvcEvent`

1. `bootstrap` : effectue différentes tâches d'initialisation (comme la création du `ViewManager`).
2. `route` : effectue les opérations de routage (matching...).
3. `dispatch` : dispatche l'action au bon contrôleur.
4. `dispatch_error` : événement lancé si une erreur se produit pendant le processus de dispatch (par exemple, si ZF 2 est incapable de trouver le bon contrôleur).
5. `render` : prépare les données pour les envoyer à la vue.
6. `render_error` : événement lancé si une erreur se produit pendant le rendu de la vue.
7. `finish` : événement lancé dès que le processus MVC est terminé.



# Tests



- Objectifs  
Automatiser l'exécution des tests en vue de détecter plus rapidement les problèmes.
- Types de tests
  - **Unitaire** : tests des méthodes d'une classe
  - **Intégration** : teste l'intégration entre plusieurs classes
  - **Fonctionnels** : teste l'application du point de vue de l'utilisateur



- Frameworks de tests  
Contiennent les classes et programmes console pour faciliter l'écriture de tests.
- Principaux frameworks PHP
  - **PHPUnit** : le plus utilisé, intégré à Zend Framework 2
  - **atoum** : grande communauté française
  - **Behat** : permet de générer le test en fonction d'un scénario écrit en anglais



- Voici une simple classe de Log

```
namespace Log;

class LoggerFile
{
    protected $fic;

    public function __construct($chemin) {
        $this->fic = fopen($chemin, "a");
    }

    public function error($message) {
        fwrite($this->fic, date(DATE_W3C) . " [ERROR] $message\n");
    }

    public function __destruct() {
        fclose($this->fic);
    }
}
```



## ▸ Son test

```
class LoggerFileTest extends PHPUnit_Framework_TestCase
{
    static protected $fichier;

    static public function setUpBeforeClass() {
        self::$fichier = __DIR__."/../error.log";
    }

    public function tearDown() {
        if(file_exists(self::$fichier)) {
            unlink(self::$fichier);
        }
    }

    public function testConstructorCreatesFile()
    {
        $logger = new \Log\LoggerFile(self::$fichier);
        $this->assertFileExists(self::$fichier);
    }

    public function testErrorIsWritten() {
        $logger = new \Log\LoggerFile(self::$fichier);

        $logger->error("Message");

        $contenuFichier = file_get_contents(self::$fichier);
        $this->assertContains("[ERROR] Message", $contenuFichier);
    }
}
```





## ▸ Exemple avec Mock (emulation) de Doctrine

```
class AlbumControllerTest extends AbstractHttpControllerTestCase
{
    public function testListActionContainsName()
    {
        $repositoryMock = $this->getMockBuilder('AddressBook\Entity\Repository\ContactRepository')
            ->disableOriginalConstructor()
            ->getMock();

        $entityMock = $this->getMockBuilder('AddressBook\Entity\Contact')
            ->getMock();

        $entityMock->expects($this->any())
            ->method('getId')
            ->will($this->returnValue(1));

        $entityMock->expects($this->any())
            ->method('getPrenom')
            ->will($this->returnValue("Romain"));

        $entityMock->expects($this->any())
            ->method('getNom')
            ->will($this->returnValue("Bohdanowicz"));

        $repositoryMock->expects($this->once())
            ->method('findBy')
            ->will($this->returnValue(array($entityMock)));

        $serviceManager = $this->getApplicationServiceLocator();
        $serviceManager->setAllowOverride(true);
        $serviceManager->setService('AddressBook\Entity\Repository\ContactRepository', $repositoryMock);

        $this->dispatch('/');
        $this->assertQueryContentContains('tr td', 'Romain Bohdanowicz');
    }
}
```



# Performances



## ▸ Checklist

Avant de passer en production vérifier :

- mettre à jour PHP
- désactiver xdebug
- activer opcache
- désactiver zend developer tools
- composer dump-autoload -o
- cache doctrine
- cache de config
- .htaccess dans le virtual host
- composer autoload pour les modules



- Autres ressources...
  - Stop aux frameworks ? (Forum PHP 2014)  
<http://www.youtube.com/watch?v=ep3Oztvy0rk>
  - Aller plus loin sur le mocking dans les tests (Forum PHP 2014)  
<http://www.youtube.com/watch?v=AHizK2kpukk>