

e-application - 2020

1

2. Découverte de Symfony 5

Matthieu Kowalczyk

Ingénieur en technologies de l'information @CGI

Nathan Levy

Ingénieur en technologies de l'information @CGI



matthieu.kowalczyk@cgi.com
nathan.levy@cgi.com

CGI



UNIVERSITÉ
DE MONTPELLIER

Agenda

1. Symfony5, les outils
2. Pourquoi un framework
3. Pourquoi Symfony
4. Comprendre les concepts de bases de Symfony5
5. L'architecture logicielle
6. TD n°2 part 1
7. Le moteur de template de Sf5 : Twig
8. TD n°2 part 2
9. Configuration
10. Service Container

1 – Symfony5, les outils

- Composer
- Symfony CLI

Composer

- 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. Le développement a débuté en avril 2011 et a donné lieu à une première version sortie le 1er mars 2012. Développé au début par Nils Adermann et Jordi Boggiano (qui continuent encore aujourd'hui à le maintenir), le projet est maintenant disponible sur la plateforme GitHub. Il est ainsi développé par toute une communauté.
- Composer est fortement inspiré de npm pour Node.js et de bundler pour Ruby.
- [Installer Composer](#).
- Le référentiel des libs : [Packagist](#).



Le client composer

- Installer une *lib* au sein d'un projet :
composer require symfony/yaml
- Installer toutes les *libs* définies dans le **composer.json** :
composer install
- Mettre à jour les *libs* d'un projet :
composer update
- Installer globalement (*root* ou *user*) une *lib* dans son système :
composer global require symfony/symfony-installer
- **Aller plus loin avec la ligne de commande.**

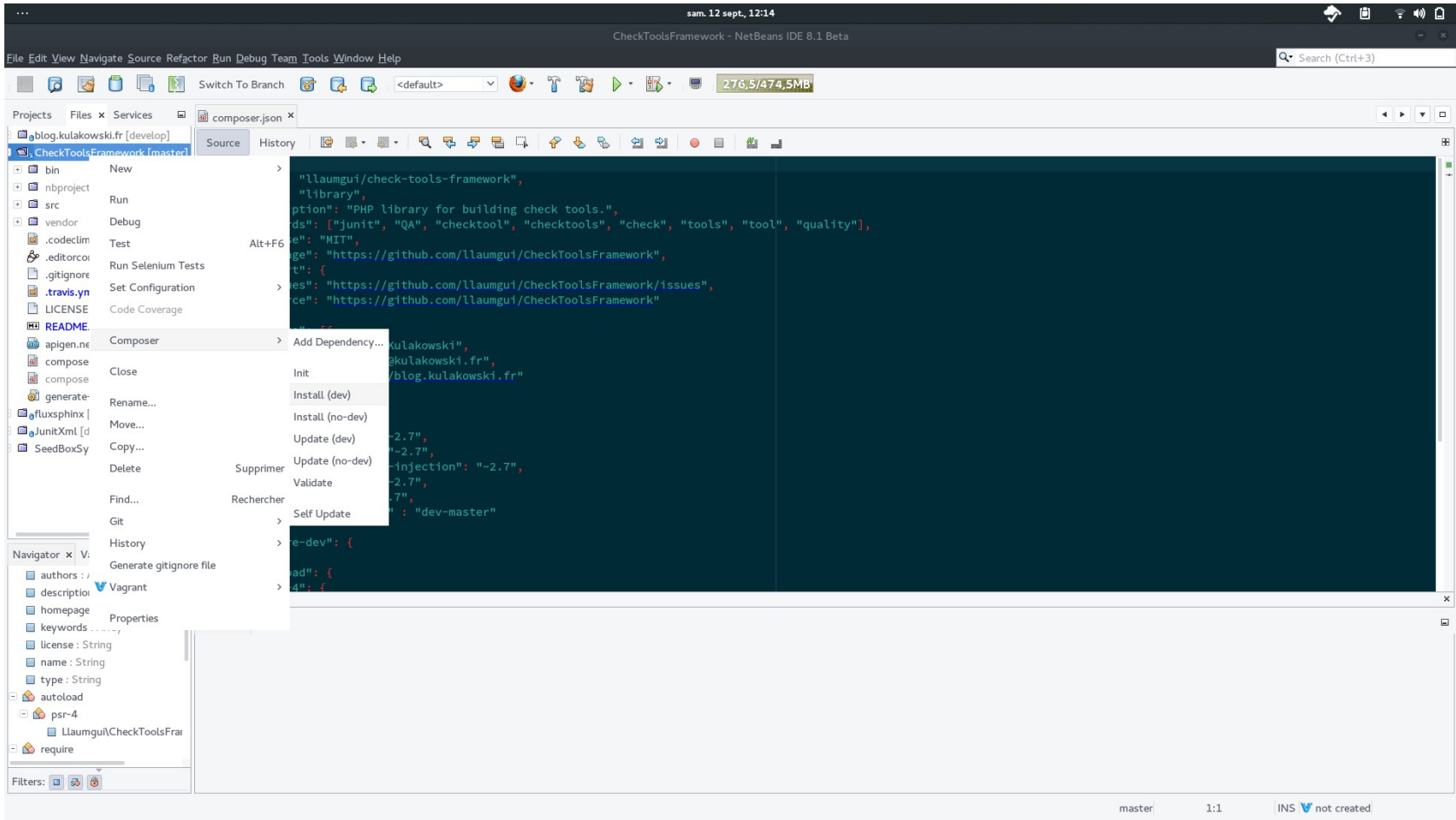
composer.json & composer.lock

- Le *composer.json* est un fichier au format JSON qui regroupe les dépendances du projet dans un **format particulier** :

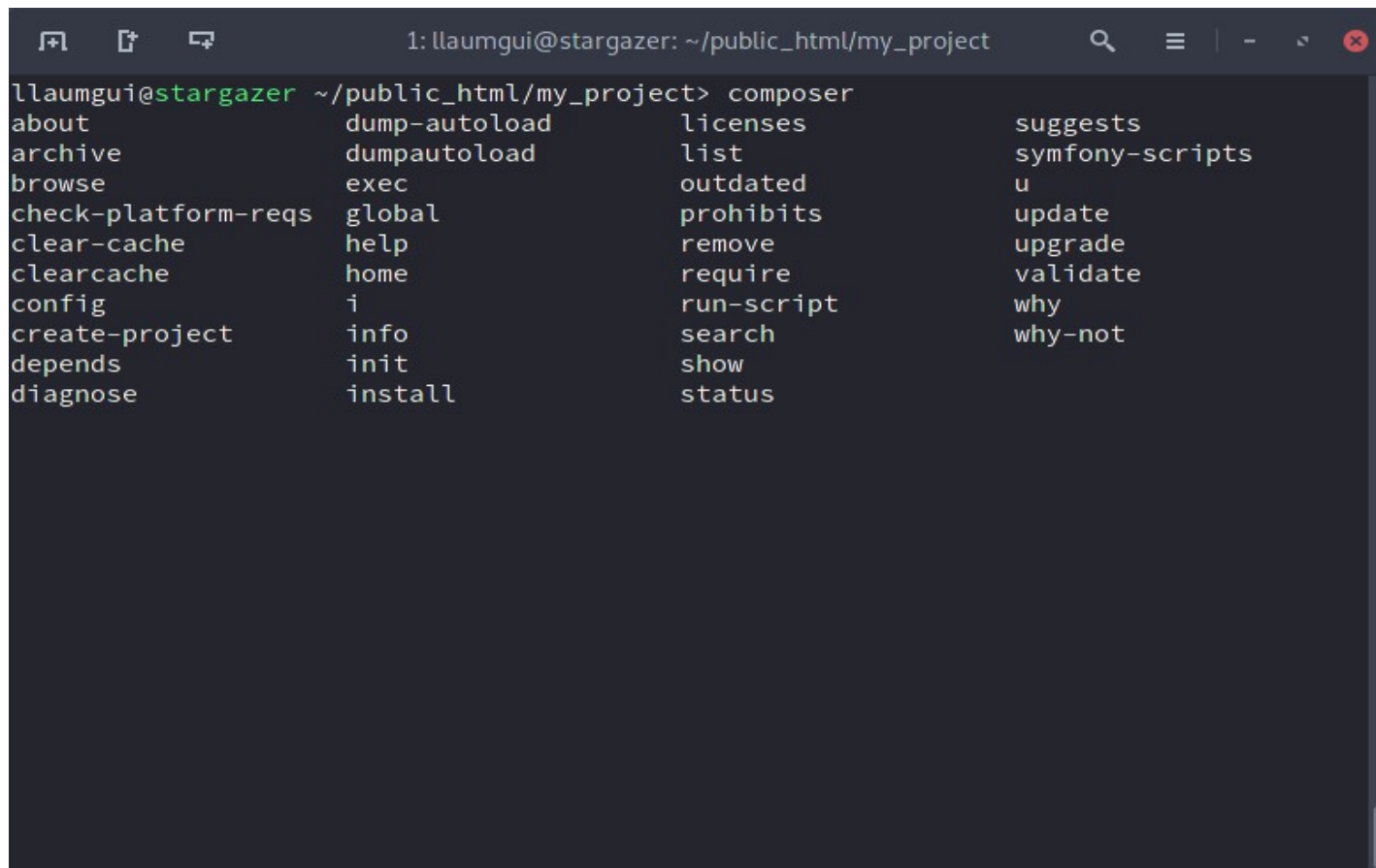
```
{  
  "require": {  
    "monolog/monolog": "1.0.*@beta",  
    "acme/foo": "@dev"  
  }  
}
```

- Le *composer.lock* est une sorte de cache des références exactes induites par le *composer.json* à un instant « t ».

Composer dans votre EDI (ex : Netbeans)



Composer dans votre shell (ex : Oh My ZSH)

A terminal window with a dark background and light text. The title bar shows the user 'llaumgui@stargazer' and the directory '~/public_html/my_project'. The prompt is 'llaumgui@stargazer ~/public_html/my_project>'. The command 'composer' has been entered, and the output is a list of available commands arranged in four columns.

```
llaumgui@stargazer ~/public_html/my_project> composer
about          dump-autoload  licenses       suggests
archive        dumpautoload   list           symfony-scripts
browse         exec           outdated      u
check-platform-reqs global        prohibits     update
clear-cache    help          remove        upgrade
clearcache     home         require       validate
config         i            run-script    why
create-project info         search        why-not
depends         init        show
diagnose       install    status
```


Symfony CLI

- **Symfony CLI** permet d'exécuter Symfony en ligne de commande.
- Intégration de la console Symfony.
- Intégration du serveur PHP de Symfony.
- Intégration de composer sans memory limit.
- Permet de créer un projet Symfony.

symfony new my_project_name --full

symfony new my_project_name

Installer Symfony 5 avec composer

- Créer un nouveau projet de site web:
`composer create-project symfony/website-skeleton my_project_name`
- Créer un nouveau projet de site web en précisant la version:
`composer create-project symfony/website-skeleton:4.4.99 my_project_name`
- Pour une version nude :
`create-project symfony/skeleton my_project_name`
- [Documentation d'installation de Symfony.](#)

2 – Pourquoi un framework ?

- Les avantages d'un framework
- Les similitudes entre framework

Un framework c'est quoi ?

En programmation informatique, un framework est un kit de composants logiciels structurels, qui servent à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture). En programmation orientée objet un framework est typiquement composé de classes mères qui seront dérivées et étendues par héritage en fonction des besoins spécifiques à chaque logiciel qui utilise le framework.

Les framework sont utilisés pour modeler l'architecture des logiciels applicatifs, des applications web, des middleware et des composants logiciels. Les framework sont achetés par les ingénieurs, puis incorporés dans des logiciels applicatifs mis sur le marché, ils sont par conséquent rarement achetés et installés séparément par un utilisateur final.

Des tentatives de francisation du terme ont été faites. On trouve ainsi parfois les termes cadre d'applications, proposé par l'Office québécois de la langue française, canevas ou cadriciel – terme en usage depuis au moins 1997. (© *Wikipedia*)

Un framework pourquoi ?

- Développement haut niveau (la roue existe déjà), plus valorisant pour les équipes.
- Gain de temps = Gain de productivité = € £ \$.
- Maturité du code.
- Une contrainte afin de cadrer les développements et les équipes.
- Communauté adhérant à la solution.
- Documentation.

Les différents types de frameworks 1/2

Comme abordé précédemment, il existe des frameworks pour tout :

- Frameworks CSS :
 - Bootstrap by Twitter
 - Pure
 - Etc...
- Frameworks JavaScript
 - Angular
 - React
 - VueJS
 - Node
 - Etc...

Les différents types de frameworks 2/2

- Frameworks php :
 - Symfony
 - Laravel
 - Codeigniter
 - Etc...
- Frameworks Python
 - Django
 - Etc...

Des composants et « Design Pattern » proches

1/3

En informatique, et plus particulièrement en développement logiciel, un patron de conception (en anglais : « design pattern ») est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels.

Un patron de conception est issu de l'expérience des concepteurs de logiciels. Il décrit sous forme de diagrammes un arrangement récurrent de rôles et d'actions joués par des modules d'un logiciel, et le nom du patron sert de vocabulaire commun entre le concepteur et le programmeur. D'une manière analogue à un patron de couture, le patron de conception décrit les grandes lignes d'une solution, qui peuvent ensuite être modifiées et adaptées en fonction des besoins.

Les patrons de conception décrivent des procédés de conception généraux et permettent en conséquence de capitaliser l'expérience appliquée à la conception de logiciel. Ils ont une influence sur l'architecture logicielle d'un système informatique. (© Wikipedia)

Des composants et « Design Pattern » proches 2/3



Framework Ikéa,
Composant Billy



Framework Alinéa,
Composant Book'in

Des composants et « Design Pattern » proches 3/3

Des composants répondant à des besoins :

- Bâtir une architecture logicielle : **MVC**.
- Une application multilingues : **i18n** (fr) & **l10n** (fr_fr).
- L'utilisation d'une base de données : **ORM**.
- Une mise en cache : **cache**.
- L'utilisation de fichiers de configurations : **configuration**.
- L'enregistrement des événements : **log**.
- La séparation fond / forme : **templates**.
- Etc...

3 – Pourquoi un Symfony ?

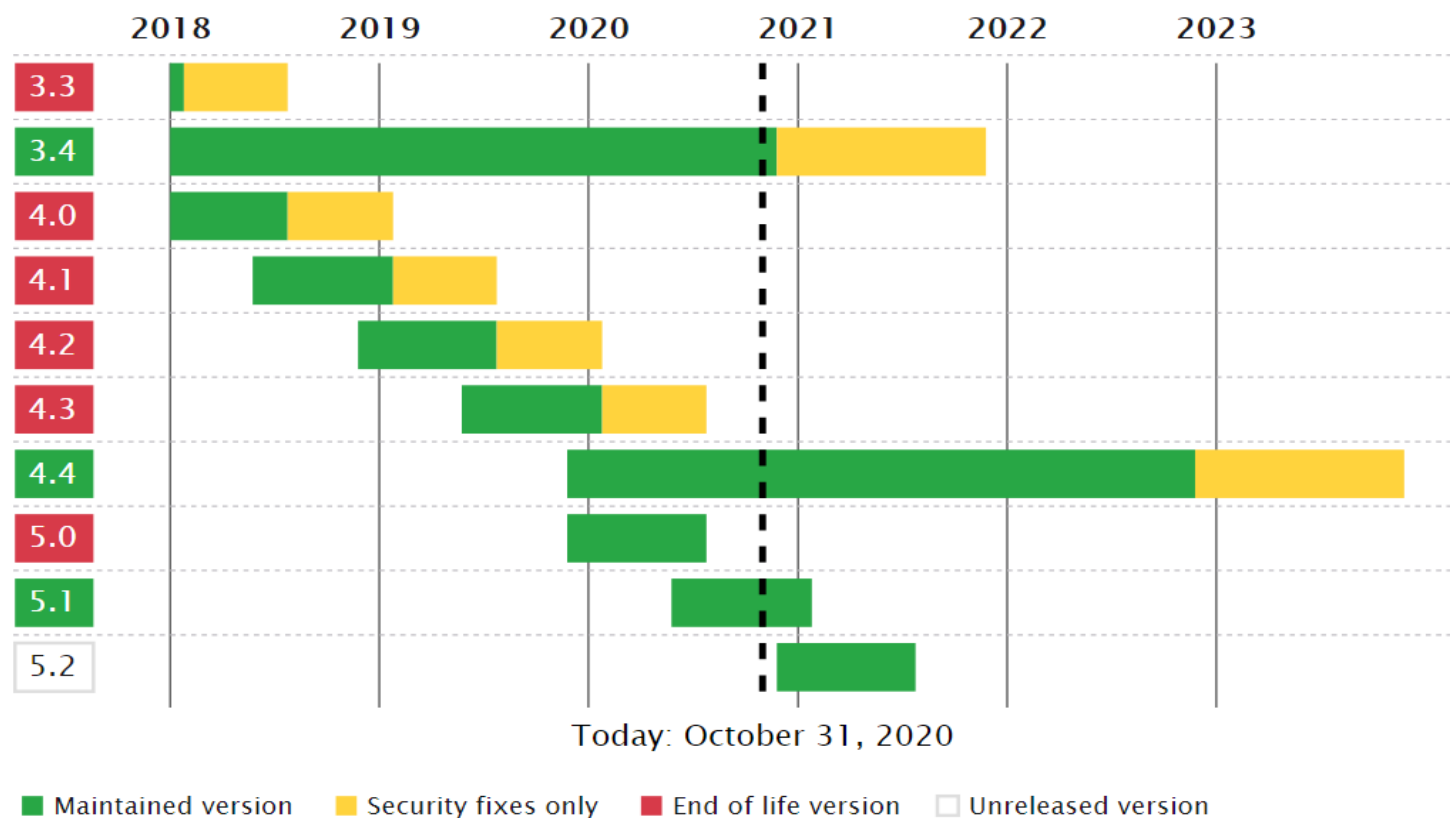
- Présentation
- Les forces
- Les faiblesses

Symfony en quelques mots & chiffres

- Le site du framework Symfony a été lancé en **octobre 2005** (maturité). À l'origine du projet, on trouve une agence web française, **Sensio**, qui a développé ce qui s'appelait à l'époque Sensio Framework pour ses propres besoins et a ensuite souhaité en partager le code avec la communauté des développeurs PHP. Le projet est alors devenu **symfony** (car le créateur voulait garder les initiales Sf comme "Sensio Framework"), puis **Symfony** à partir de la **version 2.0**.
- **La version 3.0** de Symfony est une évolution de la branche 2 qui casse la compatibilité PHP 5.3.3 pour requérir 5.5.9.
- **La version 4.0** de Symfony casse la compatibilité PHP 5 pour **requérir PHP 7.1**. Elle introduit une nouvelle architecture logicielle basée sur **Flex et les recipes**.
- **La version 5.0** est un upgrade de la version 4, peu de changements majeurs. Beaucoup de deprecations en moins.
- Licence : **MIT**.
- Sources disponibles sur GitHub : <https://github.com/symfony> .
- Exemples de grosses utilisations : **Dailymotion**, Yahoo ! (**Delicio.us**), **Mappy**, etc...
- Intégré dans : **eZ Platform** (CMS en full stack Sf2), **Drupal 9** (CMS utilisant les composants), etc...

Symfony, une roadmap claire

Symfony Releases Calendar



Les points forts de Symfony 5

- Une documentation faisant partie intégrante du framework et elle aussi sous licence libre : <http://symfony.com/doc/current/index.html>,
- Un bundle e-Commerce tiers : <http://www.sylius.com/>,
- Des bundles officiels & tiers pour vous aider à développer,
- Des bonnes pratiques et des règles ([Coding Standards](#), [Best Practices](#)),
- Une approche orientée services
- Disponible en full stack ou en components,
- Full MVC,
- Une intégration au sein de pas mal d'EDI.

Les points faibles de Symfony5

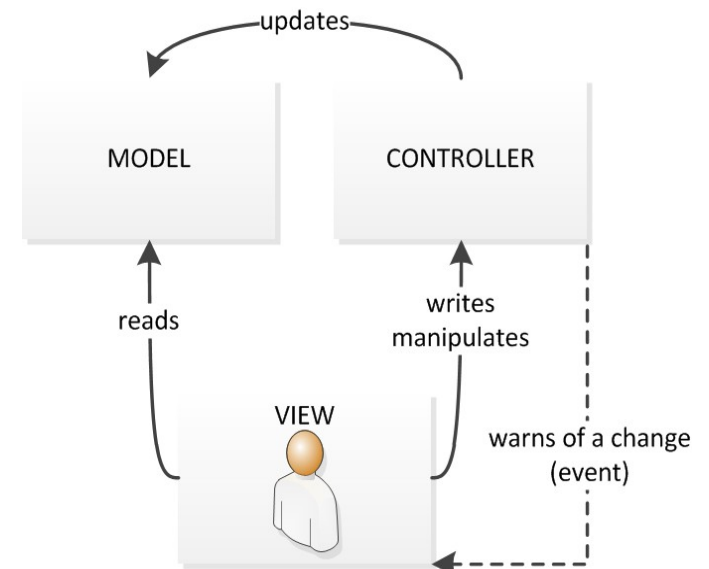
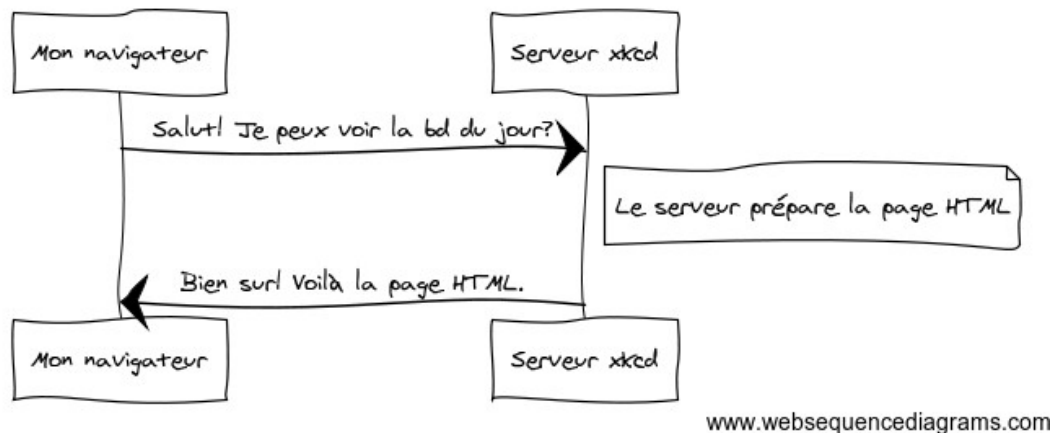
- Riche donc demande de bonnes compétences pour en maîtriser toutes les facettes,
- Lourd en environnement de développements,

4 – Comprendre les concepts de bases de Symfony5

- Les fondamentaux de Symfony5 et HTTP
- La requêtes & le contrôleur
- HttpFoundation

L'HTTP 1/3

- Il est important de comprendre le fonctionnement de l'HTTP, car c'est sur ce principe simple que repose la base de Symfony5 : router la bonne requête vers le bon contrôleur.
- En effet, Symfony5 est un framework full MVC.



L'HTTP 2/3

URL	Fichier
/index.php	exécute index.php
/index.php/contact	exécute index.php
/index.php/blog	exécute index.php

```
// index.php

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

$request = Request::createFromGlobals();

$path = $request->getPathInfo(); // Le chemin de l'URI demandée

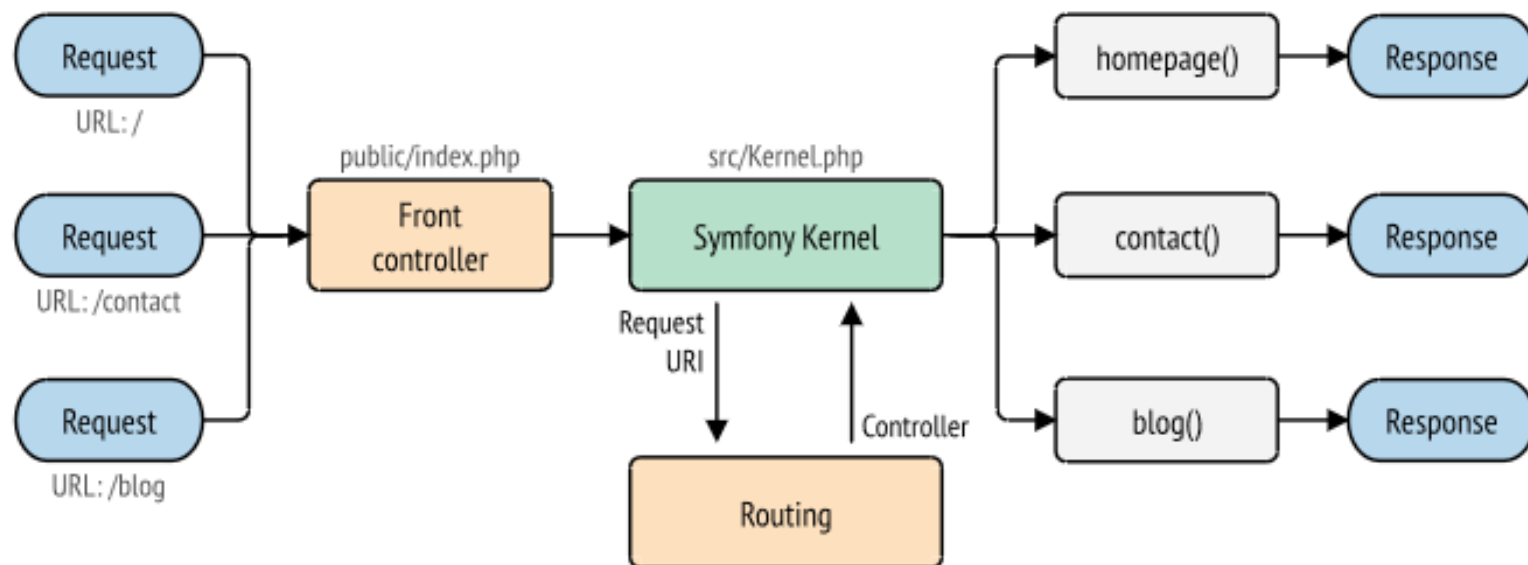
if (in_array($path, array('/', '/'))) {
    $response = new Response('Bienvenue sur le site.');
```

```
} elseif ($path == '/contact') {
    $response = new Response('Contactez nous');
```

```
} else {
    $response = new Response('Page non trouvée.', 404);
}
```

```
$response->send();
```

L'HTTP 3/3



Aller plus loin : [Symfony and HTTP Fundamentals](#).

Manipuler l'HTTP avec HttpFoundation

```
use Symfony\Component\HttpFoundation\Request;

$request = Request::createFromGlobals();

// l'URI demandée (par exemple: / about) sans aucun paramètre
$request->getPathInfo();

// obtenir respectivement des variables GET et POST
$request->query->get('foo');
$request->request->get('bar', 'valeur par défaut si bar est inexistant');

// obtenir les variables SERVER
$request->server->get('HTTP_HOST');

// obtenir une instance de UploadedFile identifiée par foo
$request->files->get('foo');

// obtenir la valeur d'un COOKIE value
$request->cookies->get('PHPSESSID');

// obtenir un entête de requête HTTP request header, normalisé en minuscules
$request->headers->get('host');
$request->headers->get('content_type');

$request->getMethod();           // GET, POST, PUT, DELETE, HEAD
$request->getLanguages();        // un tableau des langues que le client accepte
```

Router ma requête sur mon contrôleur 2/3

Voici un cas concret dans une application *Sf5 full stack* utilisant YAML et les @nnotations. ([Doc](#))

```
1  // src/Controller/LuckyController.php
2  namespace App\Controller;
3
4  use Symfony\Component\HttpFoundation\Response;
5  use Symfony\Component\Routing\Annotation\Route;
6
7  class LuckyController
8  {
9      /**
10       * @Route("/lucky/number/{max}", name="app_lucky_number")
11       */
12       public function number($max)
13       {
14           $number = random_int(0, $max);
15
16           return new Response(
17               '<html><body>Lucky number: '.$number.'</body></html>'
18           );
19       }
20  }
```


Router ma requête sur mon contrôleur 3/3

YAML

XML

PHP

```
1  # config/routes.yaml
2  blog_list:
3      path: /blog
4      # the controller value has the format 'controller_class::method_name'
5      controller: App\Controller\BlogController::list
6
7      # if the action is implemented as the __invoke() method of the
8      # controller class, you can skip the '::method_name' part:
9      # controller: App\Controller\BlogController
```

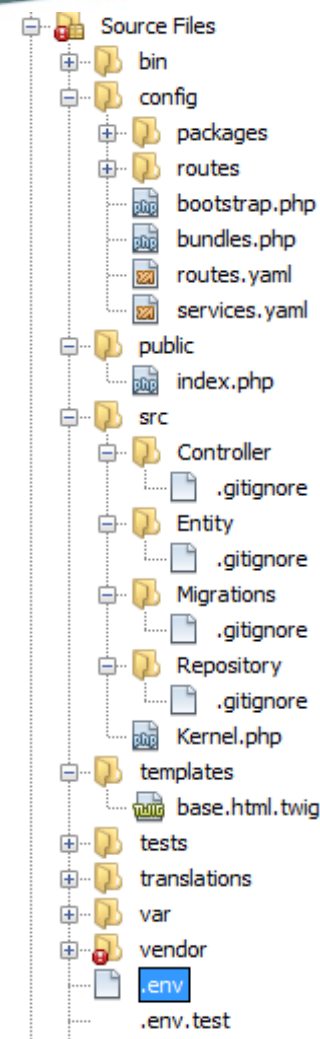


5 - L'architecture logicielle

- L'arborescence applicative
- Les fichiers importants
- Les environnements

L'arborescence d'un projet Symfony 5

- **bin** : console de Sf5, phpUnit et binaire nécessaires au projet.
- **config** : configuration de mon application.
- **config/packages** : configuration dédiée aux packages. 1 fichier = 1 package.
- **config/routes** : configuration des routes particulières (par environnement).
- **config/bootstrap.php** : la gestion des autoloads.
- **config/bundles.php** : la gestion du chargement des bundles (anciennement alouée à AppKernel.php).
- **public** : Racine (DocumentRoot) du serveur web (ex Apache).
- **src** : code source de l'application.
- **templates** : templates de l'application.
- **var/cache** (app/cache sous Sf2) : cache de l'application par environnements.
- **var/log** (app/cache sous Sf2) : logs de l'application.
- **vendor** : les bundles third party.
- **.env*** : Simulateur de variables d'environnements.



Le bundles.php

```
<?php

return [
    Symfony\Bundle\FrameworkBundle\FrameworkBundle::class => ['all' => true],
    Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle::class => ['all' => true],
    Doctrine\Bundle\DoctrineCacheBundle\DoctrineCacheBundle::class => ['all' => true],
    Doctrine\Bundle\DoctrineBundle\DoctrineBundle::class => ['all' => true],
    Doctrine\Bundle\MigrationsBundle\DoctrineMigrationsBundle::class => ['all' => true],
    Symfony\Bundle\SecurityBundle\SecurityBundle::class => ['all' => true],
    Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle::class => ['all' => true],
    Symfony\Bundle\TwigBundle\TwigBundle::class => ['all' => true],
    Symfony\Bundle\WebProfilerBundle\WebProfilerBundle::class => ['dev' => true, 'test' => true],
    Symfony\Bundle\MonologBundle\MonologBundle::class => ['all' => true],
    Symfony\Bundle\DebugBundle\DebugBundle::class => ['dev' => true, 'test' => true],
    Symfony\Bundle\MakerBundle\MakerBundle::class => ['dev' => true],
    Symfony\Bundle\WebServerBundle\WebServerBundle::class => ['dev' => true],
];
```



Le
chargement de
tous mes bundles.

Symfony Flex

- **Symfony Flex** est un plugin de composer
- Grâce à un système de « recipe » il permet à composer d'installer et configurer automatiquement des libs
- Plus besoin d'ajouter ses bundles dans le fichier config/Bundles.php
- Plus besoin de créer à la main les fichiers de configurations.

```
Symfony operations: 2 recipes (8301f2c414f09fccabeb0b3e27cc356f)
- Configuring symfony/twig-bundle (>=3.3): From github.com/symfony/recipes:master
- WARNING jolicode/gif-exception-bundle (>=1.0): From github.com/symfony/recipes-contrib:master
  The recipe for this package comes from the "contrib" repository, which is open to community contributions.
  Do you want to execute this recipe?
  [y] Yes
  [n] No
  [a] Yes for all packages, only for the current installation session
  [p] Yes permanently, never ask again for this project
  (defaults to n): y
- Configuring jolicode/gif-exception-bundle (>=1.0): From github.com/symfony/recipes-contrib:master
```

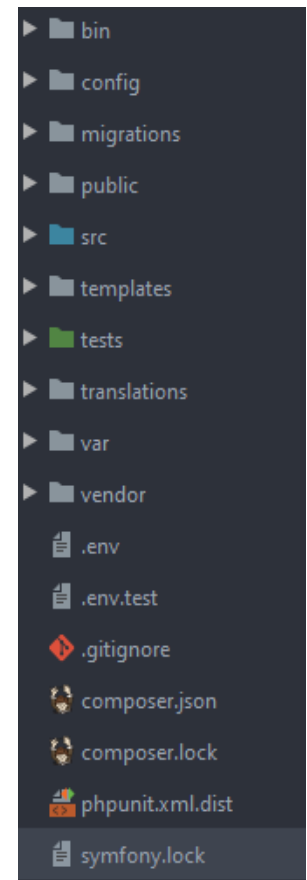
L'index.php, là où tout commence

```
1  <?php
2
3  use App\Kernel;
4  use Symfony\Component\ErrorHandler\Debug;
5  use Symfony\Component\HttpFoundation\Request;
6
7  require dirname(__DIR__).'/config/bootstrap.php';
8
9  if ($_SERVER['APP_DEBUG']) {
10     umask(0000);
11
12     Debug::enable();
13 }
14
15 if ($trustedProxies = $_SERVER['TRUSTED_PROXIES'] ?? $_ENV['TRUSTED_PROXIES'] ?? false) {
16     Request::setTrustedProxies(explode(',', $trustedProxies), Request::HEADER_X_FORWARDED_ALL ^ Request::HEADER_X_FORWARDED_HOST);
17 }
18
19 if ($trustedHosts = $_SERVER['TRUSTED_HOSTS'] ?? $_ENV['TRUSTED_HOSTS'] ?? false) {
20     Request::setTrustedHosts([$trustedHosts]);
21 }
22
23 $kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);
24 $request = Request::createFromGlobals();
25 $response = $kernel->handle($request);
26 $response->send();
27 $kernel->terminate($request, $response);
```

Les différentes configurations

Symfony5 permet des **configurations par environnement**. Ceci est idéal pour pouvoir modifier des paramètres entre les différents environnements que connaît une application professionnelle :

- Développement,
- Intégration,
- Recette,
- Préproduction,
- Production.



Aller plus loin : [Configuration Environments](#).

6 – Installer Symfony5 : TD n°2 (part 1)

- Installer Symfony5

A rendre pour le :
04/01/2021

Description du projet final

38

Dans le cadre de ce cours et afin de mettre en pratique l'utilisation du framework Symfony, nous allons réaliser un blog.

Celui-ci devra être héberger sur la plateforme Cloud Heroku.

Le blog devra être :

En HTML5 (**sémantique** et **valide**). Utilisant un **framework CSS** (Bootstrap, pure, etc).

Le blog comprend 2 contrôleurs (Blog & Crud) et 5 routes (Action) :

homepage => page d'accueil (/) listant les x derniers articles (page à page),

post => page article (/post/\$id) affichant le contenu d'un article

newPost => Ajout

editPost => Edition

deletePost => Suppression

En gris les sujet à traiter lors des prochains cours

*Le blog implémente FosUser pour se connecter (la page de login **doit être habillée**) et les routes CRUD doivent être protégées.*

Récupérer les données provenant d'une API externe et mettre en place une API pour récupérer les 5 articles les plus récents de votre blog.

Le code source et le blog seront stockés dans le « cloud ».

Installation de Symfony5 1/2

Il existe plusieurs façons d'installer Symfony5 :

- Via Composer,
- Via Symfony CLI (recommandé)

Installation de Symfony5 2/2

- Installer votre projet :
symfony new my_project_name --full

Ou

composer create-project symfony/website-skeleton my_project_name

- Puis lancer le serveur avec la commande « *symfony server:start* »

Remarque : pour développer plus facilement, nous allons utiliser le serveur embarqué dans Sf5 et non pas apache ou nginx.

Si vous voulez faire tourner Sf5 via un serveur d'application, installez-le à la racine de votre serveur (*DocumentRoot* sous apache).

7 - Le moteur de template de Sf5 : Twig

- La syntaxe
- Tags
- Filtres
- Fonctions
- Les assets

La syntaxe

- `{{ ... }}`: « Dit quelque chose »: écrit une variable ou le résultat d'une expression dans le template,
- `{% ... %}`: « Fait quelque chose »: un tag qui contrôle la logique du template ; il est utilisé pour exécuter des instructions comme la boucle for par exemple,
- `{# #}`: Commentaire, peut être utilisée sur plusieurs lignes comme la syntaxe PHP `/* commentaire */` qui est équivalente.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Bienvenue sur Symfony !</title>
5   </head>
6   <body>
7     <h1>{{ page_title }}</h1>
8
9     <ul id="navigation">
10       {% for item in navigation %}
11         <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
12       {% endfor %}
13     </ul>
14   </body>
15 </html>
```

Filtres et tags

- Twig contient également des filtres, qui modifient le contenu avant de le rendre. Le filtre suivant met la variable title en majuscule avant de la rendre :

```
{{ title | upper }}
```

- Twig est fourni avec une longue liste de **tags** et de **filtres** qui sont disponibles par défaut. Vous pouvez même ajouter **vos propres extensions** à Twig si besoin.

Fonctions

Comme vous le verrez tout au long de la documentation, Twig supporte aussi les **fonctions**, et de nouvelles fonctions peuvent être ajoutées.

Par exemple, la fonction suivante utilise le tag standard *for* et la fonction *cycle* pour écrire dix balises *div* en alternant les classes *odd* et *even* :

```
1 {% for i in 0..10 %}  
2     <div class="{ cycle(['odd', 'even'], i) }">  
3         <!-- some HTML here -->  
4     </div>  
5 {% endfor %}
```


Twig et le cache

Twig est rapide. Chaque template Twig est compilé en une classe PHP natif qui est rendue à l'exécution.

Les classes compilées sont stockées dans le répertoire `var/cache/{environment}/twig` (où `{environment}` est l'environnement, par exemple `dev` ou `prod`) et elles peuvent être utiles dans certains cas pour déboguer.

Pour vider votre cache :

```
symfony console cache:clear
```

L'héritage de template et le layout 1/2

L'une des forces Twig par rapport aux autres moteurs de templates et d'implémenter le concept d'héritage de templates.

- Tout d'abord définissons un layout ainsi que des blocs :

```
1  {# app/Resources/views/base.html.twig #}  
2  <!DOCTYPE html>  
3  <html>  
4      <head>  
5          <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
6          <title>{% block title %}Test Application{% endblock %}</title>  
7      </head>  
8      <body>  
9          <div id="sidebar">  
10             {% block sidebar %}  
11                 <ul>  
12                     <li><a href="/">Home</a></li>  
13                     <li><a href="/blog">Blog</a></li>  
14                 </ul>  
15             {% endblock %}  
16         </div>  
17  
18         <div id="content">  
19             {% block body %}{% endblock %}  
20         </div>  
21     </body>  
22 </html>
```

L'héritage de template et le layout 2/2

- Ensuite, déclarons que mon template hérite de mon layout et surchargeons les blocs :

```
1  {# src/Acme/BlogBundle/Resources/views/Blog/index.html.twig #}  
2  {% extends '::base.html.twig' %}  
3  
4  {% block title %}My cool blog posts{% endblock %}  
5  
6  {% block body %}  
7      {% for entry in blog_entries %}  
8          <h2>{{ entry.title }}</h2>  
9          <p>{{ entry.body }}</p>  
10         {% endfor %}  
11     {% endblock %}
```

Rendu de template TWIG

```
// the `render()` method returns a `Response` object with the
// contents created by the template
return $this->render('blog/index.html.twig', [
    'category' => '...',
    'blog_entries' => ['...', '...'],
]);

// the `renderView()` method only returns the contents created by the
// template, so you can use those contents later in a `Response` object
$contents = $this->renderView('blog/index.html.twig', [
    'category' => '...',
    'blog_entries' => ['...', '...'],
]);

return new Response($contents);
```

```
1  {# src/Acme/BlogBundle/Resources/views/Blog/index.html.twig #}
2  {% extends '::base.html.twig' %}
3
4  {% block title %}My cool blog posts{% endblock %}
5
6  {% block body %}
7      {% for entry in blog_entries %}
8          <h2>{{ entry.title }}</h2>
9          <p>{{ entry.body }}</p>
10         {% endfor %}
11     {% endblock %}
```

Lier vos assets

- Des images :

```

```

- Des CSS :

```
<link href="{{ asset('css/blog.css') }}" rel="stylesheet" />
```

- Versionner vos assets :

```

```

- Dans la vie réelle :

```
<link href="{{ asset('bundles/acmedemo/css/contact.css') }}"  
rel="stylesheet" />
```

Less, SaaS avec Webpack Encore

```
1  // webpack.config.js
2  // ...
3
4  Encore
5      // ...
6
7      // enable just the one you want
8
9      // processes files ending in .scss or .sass
10     .enableSassLoader()
11
12     // processes files ending in .less
13     .enableLessLoader()
14
15     // processes files ending in .styl
16     .enableStylusLoader()
17 ;
```

Aller plus loin :

- CSS Preprocessors: Sass, LESS, Stylus, etc.
- Managing CSS and JavaScript.

8 – Installer Symfony5 : TD n°2 (part 2)

- Exploiter le dossier SRC

Ma première page

Pour les besoins du projet, nous allons réaliser un contrôleur (BlogController) et 2 routes :

- / => index
- /post/idudpost => vue de détail

Dans un premier temps, nous nous contenterons que l'index rende le texte « Page d'accueil » via un template Twig et que la page post rende juste le paramètre passé en argument à l'url.

Indices :

- Contrôleur,
- Rendre des templates.

Conclusion

- Maintenant nous avons une base solide pour commencer nos devs !

9 – Configuration

- Les fichiers
- Créer ses entités
- parameters

Les fichiers de configurations

- Comme évoqué précédemment il y a plusieurs façon d'écrire des fichiers de configurations :
 - **YAML,**
 - XML,
 - PHP,
 - Pas de fichier (**@notation**).
- Les fichiers de configurations se chargent de la sorte :
 - *Les fichier du dossier config/package*
 - *Eux-mêmes surcharger par les fichier du dossier config/package/{env}*

```
1 # app/config/config.yml
2 imports:
3   - { resource: parameters.yml }
4   - { resource: security.yml }
5   - { resource: services.yml }
6
7 framework:
8   secret:          '%secret%'
9   router:          { resource: '%kernel.root_dir%/config/routing.yml' }
10  # ...
11
12 # Twig Configuration
13 twig:
14   debug:           '%kernel.debug%'
15   strict_variables: '%kernel.debug%'
16
17 # ...
```

La clé « *parameters* »

- Chaque bundle peut apporter son espace de nom ~ clé.
- Il existe un espace de nom spécial appelé « *parameters* » *il est défini dans le fichier config/service.yaml.*
- Il est utilisé pour définir des variables qui ne sont pas référencées dans d'autres fichiers de configurations.
- Ces variables sont récupérables par la suite.

```
1  # app/config/config.yml
2  # ...
3
4  parameters:
5      locale: en
6
7  framework:
8      # ...
9
10     # any string surrounded by two % is replaced by that parameter value
11     default_locale: "%locale%"
12
13     # ...
```

« locale » défini dans *parameters*.

« locale » exploité ailleurs.

Le fichier «.env»

- **Fichier ultra sensible !**
- Il ne doit **pas être versionné**.
- Il permet de simuler des variables d'environnements serveur
- Il contient toute les données sensibles (secret, accès base de données, etc.).
- En prod on utilise de vraies variables d'environnement

On exploite ailleurs.

On défini dans .env

```
framework:  
  secret: '%env(APP_SECRET) %'  
  #csrf_protection: true  
  #http_method_override: true
```

```
###> symfony/framework-bundle ###  
APP_ENV=dev  
APP_SECRET=905190bdd026685ce49ac36358b59192  
#TRUSTED_PROXIES=127.0.0.1,127.0.0.2  
#TRUSTED_HOSTS='^localhost|example\.com$'  
###< symfony/framework-bundle ###
```

Aller plus loin : créer ça configuration

- Symfony permet de créer un espace de nom dédié à son bundle, pouvant utiliser un fichier spécifique.
- Il permet également de contrôler les données renseignées : nature, absence d'informations requises, valeurs par défauts, etc...
- Je vous invite à découvrir tout cela dans la doc.

Aller plus loin : [How to Create Friendly Configuration for a Bundle.](#)

Débuguer

Une ligne de commande permet de récupérer toutes les configurations disponibles :

`php bin/console config:dump-reference [extension alias]`

```

llaumgui@stargazer ~/public_html/gro_blog> php bin/console config:dump-reference

Available registered bundles with their extension alias if available
=====

-----
Bundle name      Extension alias
-----
DebugBundle      debug
DoctrineBundle   doctrine
FrameworkBundle   framework
GrdBlogBundle
MonologBundle     monolog
SecurityBundle    security
SensioDistributionBundle sensio_distribution
SensioFrameworkExtraBundle sensio_framework_extra
SensioGeneratorBundle
SwiftmailerBundle swiftmailer
TwigBundle        twig
WebProfilerBundle web_profiler
-----

// Provide the name of a bundle as the first argument of this command to dump its default configuration. (e.g.
// config:dump-reference FrameworkBundle)

```

4 – Le Service Container

- Un service c'est quoi ?
- Créer un service
- Utiliser un service

Un service c'est quoi ?

- Un Service désigne tout objet PHP qui effectue une sorte de **tâche** « **globale** ».
- C'est un nom générique utilisé en informatique pour décrire un objet qui est créé dans un **but précis** (par ex. l'envoi des e-mails).
- Chaque service est utilisé tout au long de votre application lorsque vous avez besoin de la fonctionnalité spécifique qu'il fournit.
- Vous n'avez pas à faire quelque chose de spécial pour fabriquer un service : il suffit d'écrire une classe PHP avec un code qui accomplit une tâche spécifique.

Créer un service 1/2

- Exposer une classe en tant que service est simple et passe par un fichier de configuration.
- On peut aussi lui passer des arguments au constructeur.

```
1  # app/config/config.yml
2  services:
3      my_mailer:
4          class:      Acme\HelloBundle\Mailer
5          arguments:  [sendmail]
```

Notez le paramètre passé au service.

Conseil : Encore mieux, utilisez un fichier *services.yml* dans votre bundle pour y exposer les services de votre bundle.

Créer un service 2/2

- Depuis symfony 2.8 un nouveau système autowire/autoconfigure permet la déclaration implicite de service sans devoir spécifier les configuration dans le fichier services .yml.

```
5
6 services:
7   _defaults:
8     autowire: true
9     autoconfigure: true
10    public: false
11
12   AppBundle\:
13     resource: '../src/AppBundle/*'
14     exclude: '../src/AppBundle/{Entity,Repository,Tests}'
```

Utiliser un service

- Utiliser le service est alors simple grâce à l'injection de dépendance ou du *get* :

```
class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction(LoggerInterface $logger)
    {
        $logger->alert('Test');
        $router = $this->get('router');

        return $this->render('default/index.html.twig', []);
    }
}
```

Accessible depuis « *\$this* » car nous sommes dans un *controller*.

Argument de services

- Il est possible de passer des arguments de type string.

```
1 # app/config/services.yml
2 parameters:
3     app.mailer.transport: sendmail
4
5 services:
6     app.mailer:
7         class:      AppBundle\Mailer
8         arguments:  ['%app.mailer.transport%']
```

- Mais la chose devient intéressante lorsque l'on passe des services et que l'on fait de l'injection de services !

```
1 // src/AppBundle/Newsletter/NewsletterManager.php
2 namespace AppBundle\Newsletter;
3
4 use AppBundle\Mailer;
5
6 class NewsletterManager
7 {
8     protected $mailer;
9
10    public function __construct(Mailer $mailer)
11    {
12        $this->mailer = $mailer;
13    }
14
15    // ...
16 }
```

```
1 # app/config/services.yml
2 services:
3     app.mailer:
4         # ...
5
6     app.newsletter_manager:
7         class:      AppBundle\Newsletter\NewsletterManager
8         arguments:  ['@app.mailer']
```

Pour conclure

- Dans Symfony il faut utiliser au maximum les services et oublier les singletons.
- Vous verrez par la suite que nous utiliserons des composants natifs de Sf5 qui sont eux aussi des services.
- Une fois vos classes de service exposées vous pouvez les récupérer partout au sein de votre application.
- Différents type d'arguments sont passables aux services et vous pouvez également leur passer des services (@ en YAML) et ainsi faire de l'injection de service.
- Vous pouvez également appeler des fonction « call » lors de l'initialisation d'un service.

Conseil : Il est possible de passer le service container (@service_container), l'objet qui contient (entre autre) tous les services. C'est facile, **mais c'est mal !**

Aller plus loin : [Service Container](#).

9 - Outillage & industrialisation

: Les VCS

- Définition.
- Les différents types de VCS.
- Focus sur GIT.
- Les workflows.

Définition

Un **logiciel de gestion de versions** (ou **VCS** en anglais, pour **V**ersion **C**ontrol **S**ystem) est un logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. Il permet notamment de retrouver les différentes versions d'un lot de fichiers connexes.

Il existe aussi des logiciels et services de gestion de versions **décentralisé** (distribué) (ou **DVCS** en anglais, pour **D**istributed **V**ersion **C**ontrol **S**ystem). Git et Mercurial sont deux exemples de logiciel de gestion de versions décentralisé et sont disponibles sur la plupart des systèmes Unix et Windows.

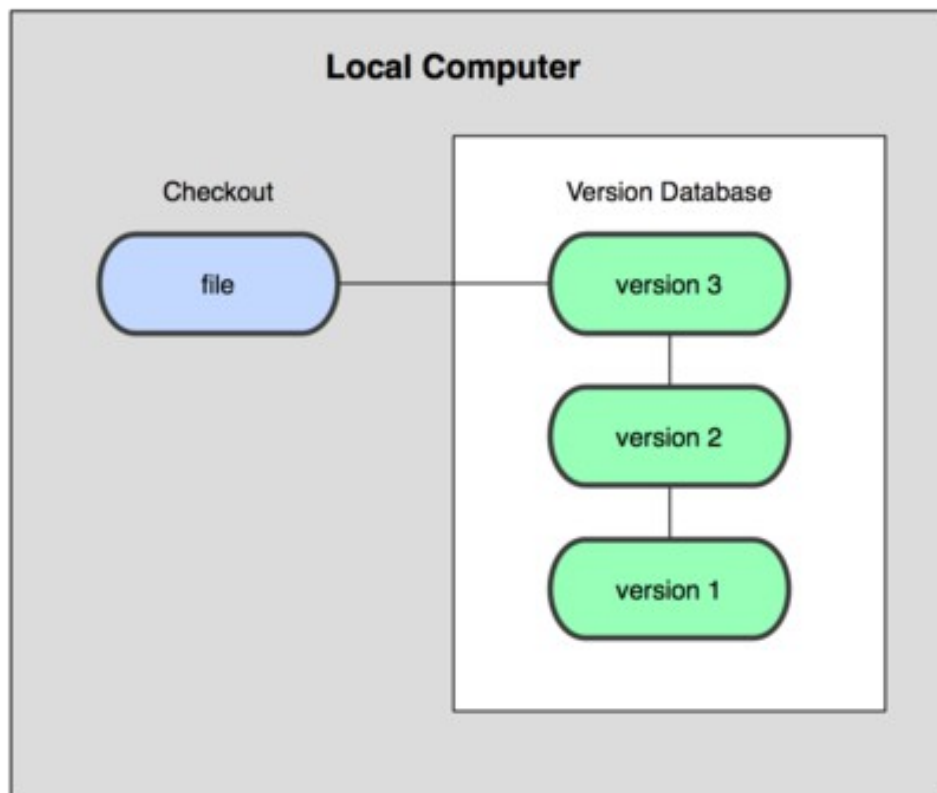
Les logiciels de gestion de versions sont utilisés notamment en développement logiciel pour conserver le code source relatif aux différentes versions d'un logiciel.
(© Wikipedia)

Répond à des besoins

- **Partager** des sources en équipe.
- **Partager** des sources avec tout le monde (open source).
- **Historique** des sources.

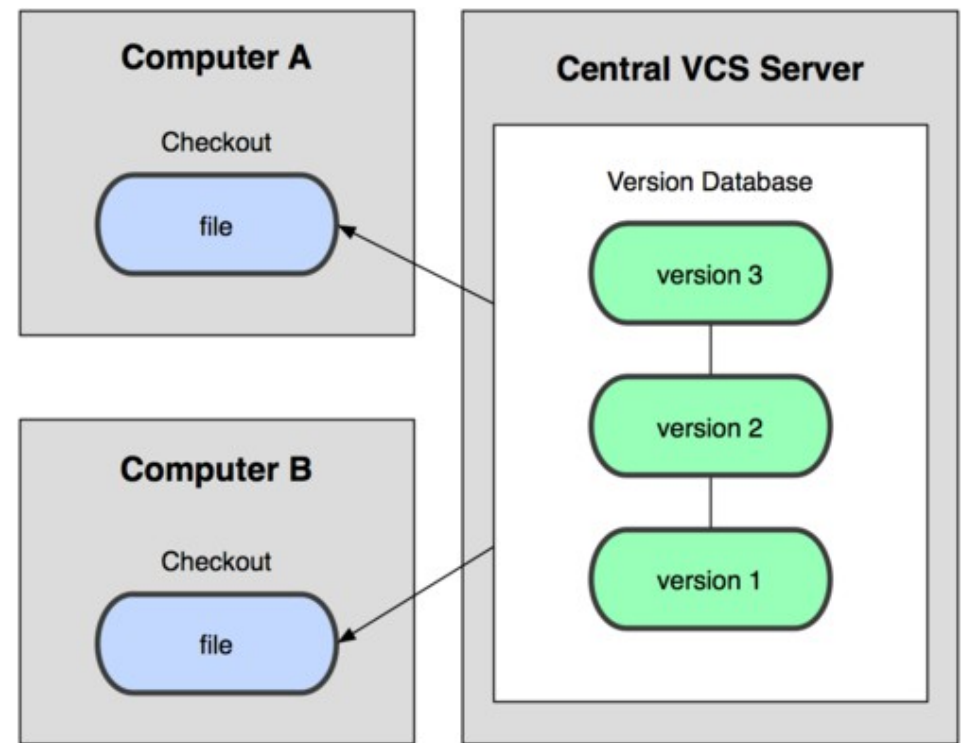
Type #1 : Gestion de versions locale

- Exemple : **RCS**
- Système plus répandu (heureusement) de nos jours.

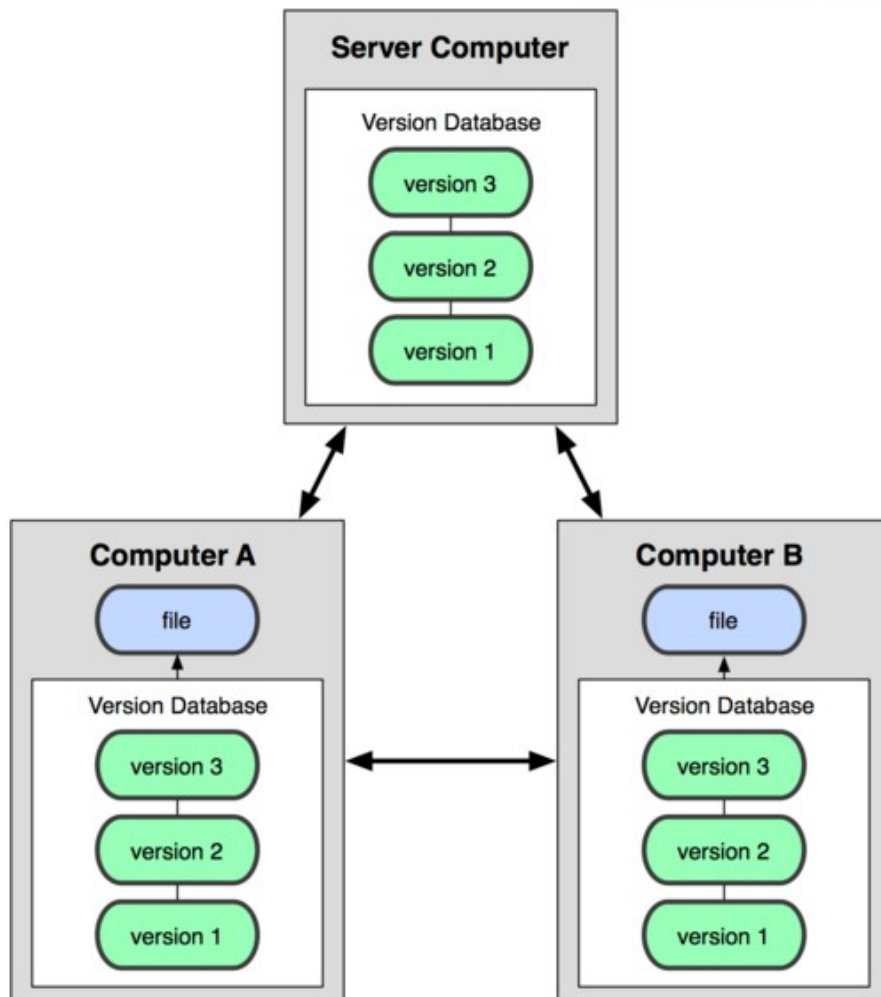


Type #2 : Gestion de versions centralisée

- VSC.
- Exemple **SVN**.
- Tout passe par le serveur central : **SPOF**.
- Les versions locales (et donc poussées) sont anarchiques et incohérentes (possibilité d'*updater* une arborescence partielle).



Type #3 : Gestion de versions décentralisée



- DVSC.
- Exemple [GIT](#), [Hg](#), [Bz](#), etc.
- Serveur local clone du distant (rapidité).
- Les versions locales (et donc poussées) sont cohérentes car on ne peut pousser si le dépôt n'est pas en *phase*.
- Puissant donc complexe à utiliser.
- Vraie gestion de branche.

Focus sur GIT : Historique

- Une histoire liée au noyau **Linux** et à **Linus Torvalds**.
- En **2002**, le projet du noyau Linux commence à utiliser un DVCS propriétaire : BitKeeper.
- En **2005**, clash entre la communauté & BitKeeper qui passe sur un modèle payant.
- Suite à ça, toujours en **2005**, La réponse de la communauté & de Linus fut la **création de GIT**.

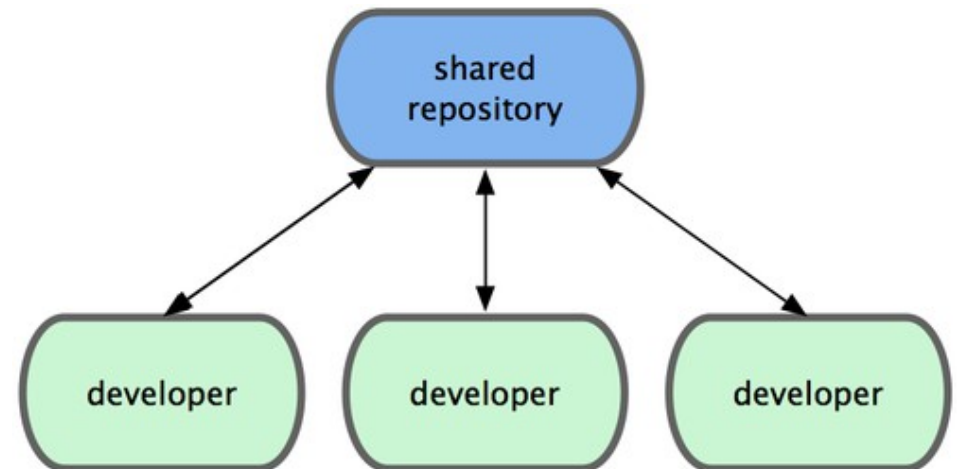
Focus sur GIT : ADN

Le cahier des charges :

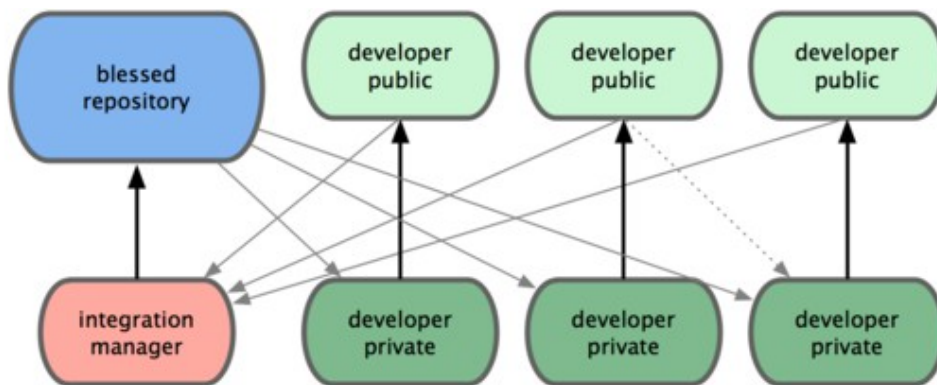
- **Vitesse.**
- Conception **simple.**
- Support pour les développements non linéaires (**milliers de branches** parallèles).
- Complètement **distribué.**
- Capacité à **gérer efficacement des projets d'envergure** tels que le noyau Linux (vitesse et compacité des données).

Workflow #1 : Mode distribué

- Le plus utilisé car le plus simple.
- Proche du VSC.
- Tous les développeurs *push* sur le même dépôt distant.



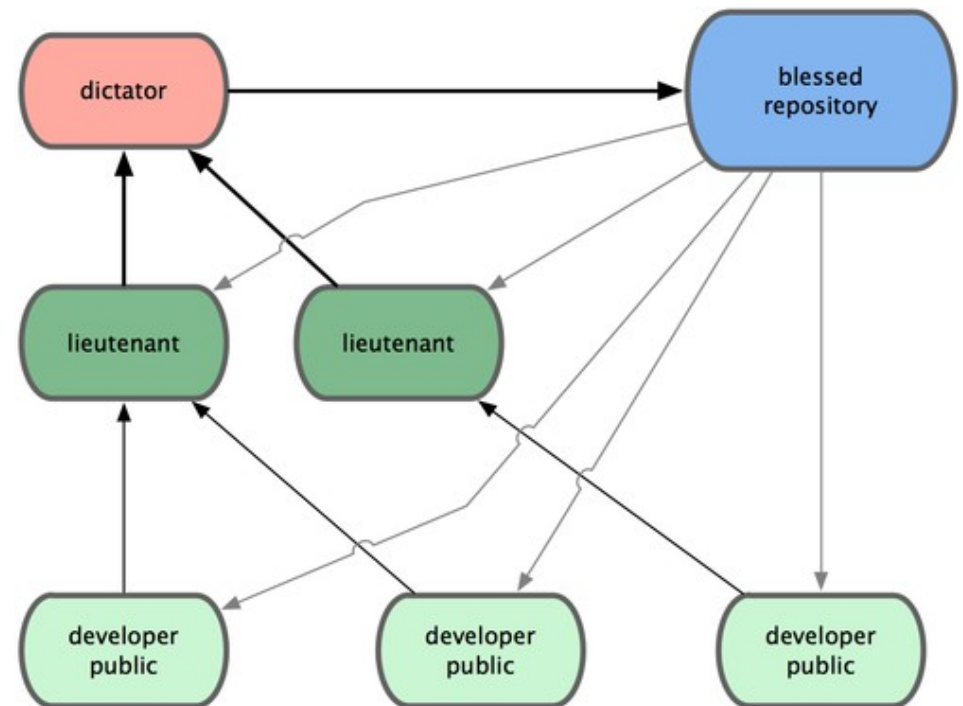
Workflow #2 : Gestionnaire d'intégration



- Approche proche du principe de *Pull Request* de [GitHub](#).
- 1 gardien du temple vérifie les *PR* avant de les *merger* dans le dépôt officiel.

Workflow #3 : Dictateur

- La méthode la plus efficace pour gérer les gros projets subdivisés en plusieurs sous-projets eux même subdivisés en d'autres projets...
- Exemple Linux.



Workflow #4 : GIT-Flow

- Plus une méthode de gestion de branches qu'un workflow.
- Utilisation de branches sémantiques :
 - **master** : code de production. On ne développe rien sur cette branche.
 - **develop** : code prêt pour le *merge* pour la nouvelle release. On peut également corriger, améliorer de petites fonctionnalités.
 - **feature/xxx** : développement des nouvelles fonctionnalités. Une fois terminé, on *merge* les changements dans *develop*.
 - **release/vx.y.z** : Préparation d'une nouvelle release. Un *merge* sera alors effectué dans master.
 - **hotfix** : permet de réparer tout de suite un bug critique en production.



TP Git

- Créer un compte github
- Créer un repository public
- Commit et push son code sur son depot

Métrique : outil de type « Checkstyle »

- Exemple : **Checkstyle** (JAVA), **PHPCS** (PHP), etc.
- Contrôle la cohérence du style d'un code source :
 - Choix des noms de classe et fonction,
 - Indentation,
 - Nom des fichiers,
 - Namespacing,
 - Etc...

Métrique : duplication de code

- Exemple **PHPCPD**, **PMD**
- Permet de détecter les copier/coller et les manques de factorisation au sein d'un projet.

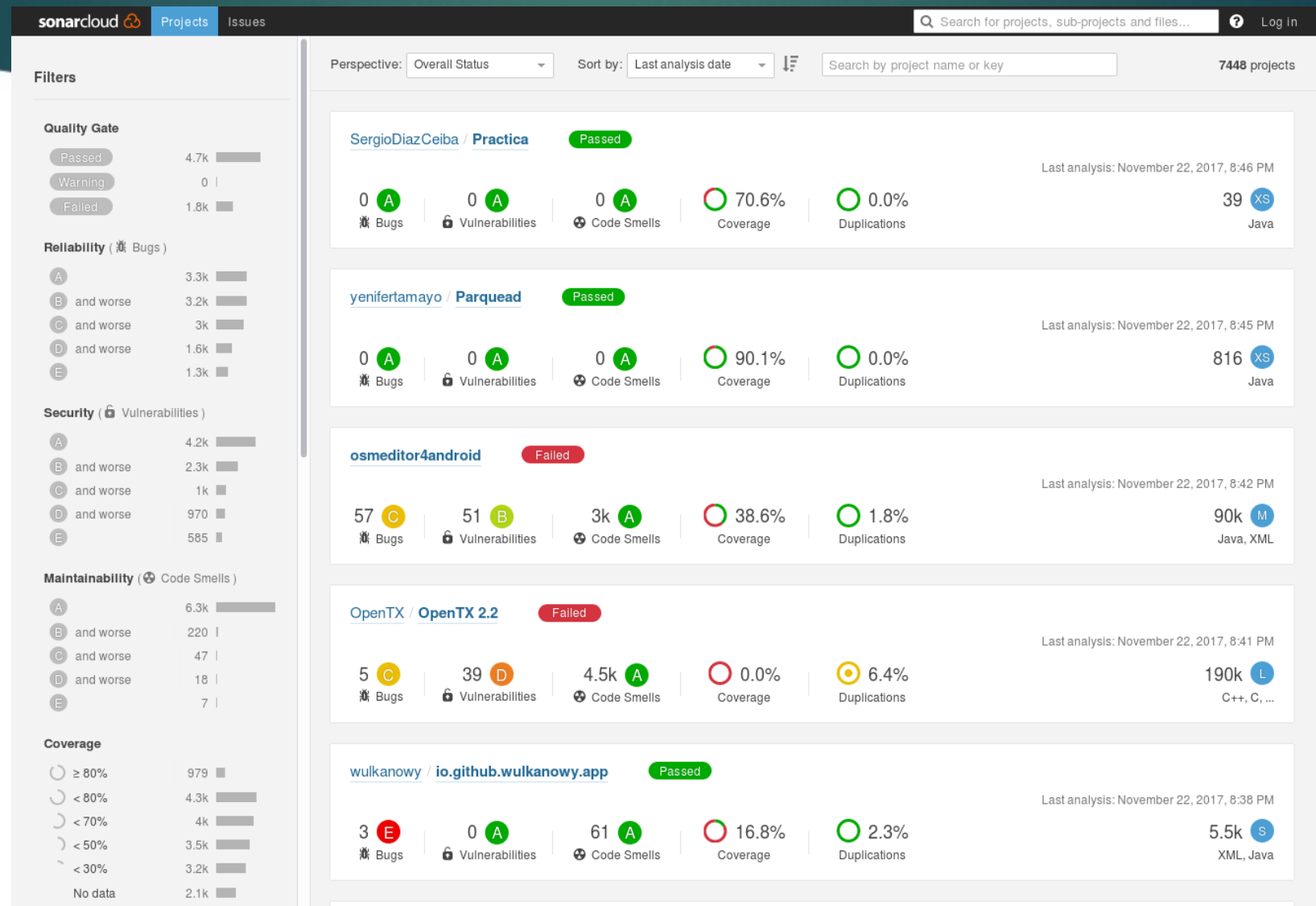
Métrique : complexité

- Exemple : [PMD](#), [Pdepend](#), etc.
- Détection de code mort, fonctions et/ou variable inutilisé.
- Notion de complexité cyclomatique
- Etc...

Métrique : divers

- Accessibilité (ex : [Asqatasun](#)).
- Ligne de code ([phploc](#)).
- Performance Web ([YSlow](#), [GTmetrix](#)).
- Bien sûr tous les outils de TU / TI / TNR ([CasperJS](#), [Selenium](#)).

Exemple d'outil de qualimétrie : Sonar



L'expert technique / Lead developer

- Son rôle est primordiale,
- Il doit suivre tous ces indicateurs,
- Il doit faire des revues de code.

Annexes

Guide de survit de bin/console

- Vider le cache :

`bin/console cache:clear`

Webographie

- Site officiel de Symfony : <http://symfony.com/>
- Le manuel PHP : <http://php.net/manual/fr/index.php>
- La doc : <http://symfony.com/doc/current/book/index.html>

Bibliographie

- Toutes la documentation et les « books » officiels sont librement téléchargeables
<http://symfony.com/doc/current/index.html>

Licence

La documentation Symfony5 étant bien faite et sous Licence libre (MIT), des illustrations tout comme des bouts de textes de ce cours en sont extraits.