Symfony

What is Symfony ?

Symfony is an Open Source distributed PHP framework. Of course, when you know how to develop in PHP, you don't need a framework. However, it is very nice to have one!

A framework helps you work better (structuring developments) and faster (reusing generic modules). A framework facilitates long-term maintenance and scalability by complying with standard development rules. Compliance with development standards also simplifies integrating and interfacing the application with the rest of the information system.

Installing Symfony

Symfony installer

Symfony provides an installer to help you kickstart your projects. It is only available for Symfony 2.x and 3.x. If you want to install Symfony 4.x you'll need to use Composer.

```
$ composer create-project symfony/skeleton my_project_name
```

For more info visit Symfonys offical documentation

MacOS and Linux

```
$ sudo curl -LsS https://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony
```

Windows

```
c:\> php -r "file_put_contents('symfony', file_get_contents('https://symfony.com/installer'
```

Now that you have the installer you may move it into a project directory and use it as follows.

```
c:\> php symfony
```

If your too lazy too move the installer in all of your Symfony project (like me) you can set it as global command.

```
c:\> (echo @ECHO OFF & echo php "%~dp0symfony" %*) > symfony.bat
```

Using the installer

```
# MacOS and linux
$ symfony new my_project

# Windows
c:\> php symfony new my_project
```

If you want to install a specific version of Symfony you can provide the version after your project name.

```
# MacOS and linux
$ symfony new my_project 3.4

# Windows
c:\> php symfony new my_project 3.4
```

File structure

```
my_project/
├─ app      # This folder will contain everything that's related to the application. All bu
├─ bin      # Contains all the symfony cli tools
├─ src      # It's in this folder that you'll put your source code, that will be split into
├─ tests    # In this directory you'll find all the tests for your application.
├─ var      # The files that Symfony creates during its process will go here. e.g. logs, ca
├─ vendor   # All of the projects dependencies will store in this directory. When you do a
└─ web      # This is the frontend of the application. In here you'll find all the files de
```

Bundles

⚠ As of Symfony 4.0 the Bundle system is no longer recommended.

Bundles are a structured set of files. Each Bundle will be an implementation of a single feature. It would like this. BlogBundle, FormBundle and a UserBundle. The great thing about Bundles is that they are similar to plugins, which means you can easily share your code with somebody else and you can install third party Bundles offered by the community.

```
MyBundle/
├─ Controller/          # Contains the controllers of the bundle e.g. StuffController.php
├─ Resources/config     # Houses configuration for the bundle e.g. routing
├─ Resources/views      # Holds templates by controller name. e.g. Stuff/index.html.twig
├─ Resources/public     # Contains the web assets (img, css, js) linked to project web dire
└─ Tests                # Holds all the tests for the bundle
```

You can create a Bundle by hand or you can Symfonys cli to generate if.

```
$ php bin/console generate:bundle --namespace=Foo/BarBundle
```

Actions and Controllers

All of your controllers are stored in the Controller folder within your Bundle e.g. `src/AppBundle/Controller/DefaultController.php`

If you open ths file you'll see the following :

```php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends Controller
{
    /**
     * @Route("/", name="homepage")
     */
    public function indexAction()
    {
        return $this->render('default/index.html.twig', [
            'base_dir' => realpath($this->getParameter('kernel.project_dir')).DIRECTORY_SEP
        ]);
    }
}
```

As you can see controller in Symfony are PHP classes and their name is suffixed with the word `Controller`.

The different methods associated with an URL are called `actions`. In this example the controller has one method called `index`.

Routing

Symfonys routing system can be used in a variety of ways. You can load them from files or have them as anotation of your controllers and actions.

When loading your routes from a file, you can use YAML, XML or PHP files.

```yaml
# app/config/routing.yml
route1:
    path:     /category/{name}
    defaults: { _controller: 'AppBundle:Default:category' }
    options:
        utf8: true
```

```php
// app/config/routing.php
use Symfony\Component\Routing\RouteCollection;
use Symfony\Component\Routing\Route;

$collection = new RouteCollection();
$collection->add('route1', new Route('/category/{name}',
    array(
        '_controller' => 'AppBundle:Default:category',
    ),
    array(),
    array(
        'utf8' => true,
    )
);

// ...

return $collection;
```

```xml
<!-- app/config/routing.xml -->
<?xml version="1.0" encoding="UTF-8" ?>
```

```xml
<routes xmlns="http://symfony.com/schema/routing"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://symfony.com/schema/routing
        http://symfony.com/schema/routing/routing-1.0.xsd">

    <route id="route1" path="/category/{name}">
        <default key="_controller">AppBundle:Default:category</default>
        <option key="utf8">true</option>
    </route>
</routes>
```

```php
// src/AppBundle/Controller/DefaultController.php
namespace AppBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends Controller
{
    /**
     * @Route("/category/{name}", name="route1", options={"utf8": true})
     */
    public function categoryAction()
    {
        // ...
    }
    // ...
}
```

Templating

Symfony uses Twig as its templating engine. By default your applications templates are located in `app/Ressources/views/yyy/zzz.html.twig`. But you can keep them within your controllers so you can keep everything close together .

```twig
{# app/Resources/views/default/index.html.twig #}
{% extends 'base.html.twig' %}

{% block body %}
    <h1>Welcome to Symfony</h1>

    {# ... #}
{% endblock %}
```

To render your view you'll use the handy `render` method provided by Symfonys base

`Controller` class.

```
return $this->render('render/index.html.twig', [
    # Params related to view will go here
    # e.g. 'foo' => $bar,
]);
```