Table of Contents

You are browsing the **Symfony 4 documentation**, which changes significantly from Symfony 3.x. If your app doesn't use Symfony 4 yet, browse the Symfony 3.4 documentation.

Create your First Page in Symfony

Creating a new page - whether it's an HTML page or a JSON endpoint - is a two-step process:

1. **Create a route**: A route is the URL (e.g. `/about`) to your page and points to a controller;
2. **Create a controller**: A controller is the PHP function you write that builds the page. You take the incoming request information and use it to create a Symfony `Response` object, which can hold HTML content, a JSON string or even a binary file like an image or PDF.

Screencast

Do you prefer video tutorials? Check out the Stellar Development with Symfony screencast series.

Symfony *embraces* the HTTP Request-Response lifecycle. To find out more, see *Symfony and HTTP Fundamentals*.

Creating a Page: Route and Controller ¶

> **Tip**
>
> Before continuing, make sure you've read the *Setup* article and can access your new Symfony app in the browser.

Suppose you want to create a page - `/lucky/number` - that generates a lucky (well, random) number and prints it. To do that, create a "Controller class" and a "controller" method inside of it:

```php
// src/Controller/LuckyController.php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;

class LuckyController
{
    public function number()
    {
        $number = random_int(0, 100);

        return new Response(
            '<html><body>Lucky number: '.$number.'</body></html>'
        );
    }
}
```

Now you need to associate this controller function with a public URL (e.g. `/lucky/number`) so that the `number()` method is executed when a user browses to it. This association is defined by creating a **route** in the `config/routes.yaml` file:

```yaml
# config/routes.yaml

# the "app_lucky_number" route name is not important yet
app_lucky_number:
    path: /lucky/number
    controller: App\Controller\LuckyController::number
```

That's it! If you are using Symfony web server, try it out by going to:

> *http://localhost:8000/lucky/number*

If you see a lucky number being printed back to you, congratulations! But before you run off to play the lottery, check out how this works. Remember the two steps to creating a page?

1. *Create a route*: In `config/routes.yaml`, the route defines the URL to your page (`path`) and what `controller` to call. You'll learn more about routing in its own section, including how to make *variable* URLs;

2. *Create a controller*: This is a function where *you* build the page and ultimately return a `Response` object. You'll learn more about controllers in their own section, including how to return JSON responses.

Annotation Routes ¶

Instead of defining your route in YAML, Symfony also allows you to use *annotation* routes. To do this, install the annotations package:

```
$ composer require annotations
```

You can now add your route directly *above* the controller:

```
1   // src/Controller/LuckyController.php
2
3   // ...
4 + use Symfony\Component\Routing\Annotation\Route;
5
6   class LuckyController
7   {
8 +     /**
9 +      * @Route("/lucky/number")
10 +     */
11      public function number()
12      {
13          // this looks exactly the same
14      }
15  }
```

That's it! The page - `http://localhost:8000/lucky/number` will work exactly like before! Annotations are the recommended way to configure routes.

Auto-Installing Recipes with Symfony Flex ¶

You may not have noticed, but when you ran `composer require annotations`, two special things happened, both thanks to a powerful Composer plugin called *Flex*.

First, `annotations` isn't a real package name: it's an *alias* (i.e. shortcut) that Flex resolves to `sensio/framework-extra-bundle`.

Second, after this package was downloaded, Flex executed a *recipe*, which is a set of automated instructions that tell Symfony how to integrate an external package. Flex recipes exist for many packages and have the ability to do a lot, like adding configuration files, creating directories, updating `.gitignore` and adding new config to your `.env` file. Flex *automates* the installation of packages so you can get back to coding.

You can learn more about Flex by reading "*Using Symfony Flex to Manage Symfony Applications*". But that's not necessary: Flex works automatically in the background when you add packages.

The bin/console Command ¶

Your project already has a powerful debugging tool inside: the `bin/console` command. Try running it:

```
```

```
$ php bin/console
```

You should see a list of commands that can give you debugging information, help generate code, generate database migrations and a lot more. As you install more packages, you'll see more commands.

To get a list of *all* of the routes in your system, use the `debug:router` command:

```
$ php bin/console debug:router
```

You should see your `app_lucky_number` route at the very top:

Name	Method	Scheme	Host	Path
app_lucky_number	ANY	ANY	ANY	/lucky/number

You will also see debugging routes below `app_lucky_number` -- more on the debugging routes in the next section.

You'll learn about many more commands as you continue!

The Web Debug Toolbar: Debugging Dream ¶

One of Symfony's *killer* features is the Web Debug Toolbar: a bar that displays a *huge* amount of debugging information along the bottom of your page while developing. This is all included out of the box using a package called `symfony/profiler-pack`.

You will see a black bar along the bottom of the page. You'll learn more about all the information it holds along the way, but feel free to experiment: hover over and click the different icons to get information about routing, performance, logging and more.

Rendering a Template ¶

If you're returning HTML from your controller, you'll probably want to render a template. Fortunately, Symfony comes with Twig: a templating language that's easy, powerful and actually quite fun.

Make sure that `LuckyController` extends Symfony's base `AbstractController` class:

```
1   // src/Controller/LuckyController.php
2
3   // ...
4 + use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
5
6 - class LuckyController
7 + class LuckyController extends AbstractController
8   {
9       // ...
10  }
```

Now, use the handy `render()` function to render a template. Pass it a `number` variable so you can use it in Twig:

```
// src/Controller/LuckyController.php

// ...
class LuckyController extends AbstractController
{
    /**
     * @Route("/lucky/number")
     */
    public function number()
    {
        $number = random_int(0, 100);

        return $this->render('lucky/number.html.twig', [
            'number' => $number,
        ]);
    }
}
```

Template files live in the `templates/` directory, which was created for you automatically when you installed Twig. Create a new `templates/lucky` directory with a new `number.html.twig` file inside:

```
{# templates/lucky/number.html.twig #}

<h1>Your lucky number is {{ number }}</h1>
```

The `{{ number }}` syntax is used to *print* variables in Twig. Refresh your browser to get your *new* lucky number!

> *http://localhost:8000/lucky/number*

Now you may wonder where the Web Debug Toolbar has gone: that's because there is no `</body>` tag in the current template. You can add the body element yourself, or extend `base.html.twig`, which contains all default HTML elements.

In the *Creating and Using Templates* article, you'll learn all about Twig: how to loop, render other templates and leverage its powerful layout inheritance system.

Checking out the Project Structure ¶

Great news! You've already worked inside the most important directories in your project:

`config/`
: Contains... configuration of course!. You will configure routes, *services* and packages.

`src/`
: All your PHP code lives here.

`templates/`
: All your Twig templates live here.

Most of the time, you'll be working in `src/`, `templates/` or `config/`. As you keep reading, you'll learn what can be done inside each of these.

So what about the other directories in the project?

`bin/`
: The famous `bin/console` file lives here (and other, less important executable files).

`var/`
: This is where automatically-created files are stored, like cache files (`var/cache/`) and logs (`var/log/`).

`vendor/`
: Third-party (i.e. "vendor") libraries live here! These are downloaded via the Composer package manager.

`public/`
: This is the document root for your project: you put any publicly accessible files here.

And when you install new packages, new directories will be created automatically when needed.

What's Next? ¶

Congrats! You're already starting to master Symfony and learn a whole new way of building beautiful, functional, fast and maintainable apps.

Ok, time to finish mastering the fundamentals by reading these articles:

- Routing
- Controller
- Creating and Using Templates
- Configuring Symfony (and Environments)

Then, learn about other important topics like the *service container*, the *form system*, using *Doctrine* (if you need to query a database) and more!

Have fun!

Go Deeper with HTTP & Framework Fundamentals ¶

- Symfony versus Flat PHP
- Symfony and HTTP Fundamentals

Symfony 3.4.23 released
March 3, 2019

They Help Us Make Symfony

Thanks **Jordan Hoff** for being a Symfony contributor.

1 commit · **2** lines

Get Involved in the Community

A passionate group of over 600,000 developers from more than 120 countries, all committed to helping PHP surpass the impossible.

Getting involved →

SymfonyInsight

Follow Symfony

Switch to dark theme