Templates

Overview

After doing all the Request logic you need to return a response back to the user's browser, and when your response is HTML and CSS you will use a templating system.

Symfony by default relies on a powerful and easy to use and learn templating system called Twig.

Twig basic syntax

`{{ ... }}` - interpolation - output a variable or expression
`{% ... %}` - tag - control structures like `if`, `for`
`{# ... #}` - comment, equivalent of `/* comment */` in PHP

`{% for %}` - loop, renders content specific amount of times

```
{# Numbers #}
{% for i in 1..10 %}
    Number is {{ i }}
{% endfor %}
```

You can use else to render content when iterated array/object contains 0 elements:

```
{# Iterable (array or iterable object) #}
{% for user in users %}
 <li>{{ user.username }}</li>
{% else %}
 <li>No users yet!</li>
{% endfor %}
```

`{% if %}` - conditionally render content

```
{% if enabled %}
 <p>This option is enabled</p>
{% endif %}
```

`if`, `elseif`, `else` example:

```
{% if stock > 10 %}
 Available
{% elseif stock > 0 %}
 Only {{ stock }} left!
{% else %}
 Sold-out!
{% endif %}
```

Key, Value in Twig

Associative array in PHP (with key and value):

```
['name' => 'John', 'age' => 33]
```

In Twig it's called `hash` and defined:

```
{'name': 'John', 'age': 33}
```

Normal arrays are like in PHP:

```
// In PHP
[1, 2, 3]
```

```
{# In Twig #}
[1, 2, 3]
```

IMPORTANT!
This is often use to pass parameters to Twig functions.

Dot notation `.`

`foo.bar` - a simple notation for accessing `objects` and `arrays` in Twig

In case none of above paths is successful return `null`.

PHP arrays can be accessed using `foo['bar']`. If `bar` is a valid key on `foo` array, return it's value, otherwise `null`.

Assigning value to a variable

`%{` `set` variable_name_here `=` expression `%}`

```
{% set name = "Piotr" %}
```

String concatenation

`~` is used for string concatenation (same as `.` in PHP)

```
{% set greeting = 'Hello ' %}
{% set name = 'Fabien' %}

{# Will output hello fabien #}
{{ greeting ~ name|lower }}
```

Tests, operators, comparisons

PHP	Twig	Description
`$a && $b`	`a and b`	AND operator

PHP	Twig	Description
`$a || $b`	`a or b`	OR operator
`!$a`	`not a`	NOT operator
`in_array(1, [1, 2, 3])`	`1 in [1, 2, 3]`	value is in array
`range(1, 5)`	`1..5`	create an array of numbers
`"foo $bar"`	`{{ "foo #{bar}" }}`	String interpolation
`$number % 2 == 1`	`number is odd`	Number is odd
`empty($foo)`	`foo is empty`	Expression is empty
`substr('http://www.googe.pl', 0, 4) === "http"`	`'http://www.googe.pl" starts with "http'`	String starts with a value
`preg_match('/^[0-9]+$/, $phone)`	`number matches '/^[0-9]+$/'`	Expressions matches regular expression

Twig filters

Filters modify content before rendering.

`{{ title|upper }}` - make contents of `title` UPPERCASE

Filter are separated by `|` pipe symbol
and can be chained `{{ text|upper|spaceless }}`

```
{# Given text is "a  bbb c" #}
{{ text|upper|spaceless }}
{# Will output "ABBBC" #}
```

Some filters accept arguments, surrounded by parenthesis:

```
{# Given list is an array ["Thomas", "John", "Terry"] #}
{{ list|join(', ') }}
{# Will output "Thomas, John, Terry" #}
```

HINT!
`php bin/console debug:twig` - will give you a list of all Twig filters, functions, global variables and "tests"

Twig caching

Twig templates are compiled to PHP which makes them very efficient. This is automatically handled by Symfony for you, you don't need to do any extra step.

Template inheritance and layouts

`{% extends 'base.html.twig' %}` - which template to extend
`{% block %}`, `{% endblock %}` - defines a block on parent template, and specifies content to put inside the block on child template

IMPORTANT!
To use inheritance in particular template, `{% extends %}` has to be the very first instruction in that template!

Example parent template:
This templates defines 3 blocks called `title`, `styles` and `body`.

```
{# Parent template: parent.html.twig #}
<head>
 <title>{% block title %}{% endblock %}</title>
 {% block styles %}
 {% endblock %}
</head>

<body>
 {% block body %}
 {% endblock %}
</body>
```

Example child template:

Twig will combine `parent.html.twig` with `about.html.twig`, **replacing** the content between `{% block %}` and `{% endblock %}` with what is provided in the child template.

IMPORTANT!
To keep contents of the `block` already rendered in parent (or one of the parent) templates, call `{{ parent() }}` Twig function.

```
{# Child template: about.html.twig #}
{% extends 'parent.html.twig' %}
{% block title %}About{% endblock %}
{% block styles %}
 {{ parent() }}
 .big-font {
  font-size: 1.5em;
 }
{% endblock %}

{% block body %}
<div class="big-font">
 Welcome to about page!
</div>
{% endblock %}
```

Template naming and locations

`/templates` - default directory for your application templates and templates you override from bundles

Given the following structure:

```
/templates
 base.html.twig
 header.html.twig
 /blog
  index.html.twig
  show.html.twig
  details.html.twig
```

Referencing templates:

```
{# Extending #}
{% extends 'base.html.twig %}
{% extends 'blog/index.html.twig %}
{# Including #}
{{ include('header.html.twig') }}
```

Rendering:

```
class BlogController
{
 public function index()
 {
  return $this->render('blog/index.html.twig');
 }
}
```

Including templates

For re-using the same markup, you can use the Twig `{{ include() }}` function inside any Twig template.

```
{{ include('blog/details.html.twig', { 'post': post }) }}
```

`{{` interpolation start, `include` function call, `'blog/details.html.twig'` template name, `{ 'post': post }` - arguments to pass to the template, `}}` interpolation end

Linking

Generating URLs using routes defined in your application:

`path('route.name', {'id': 1})`

Example:

```
<a href="{{ path('blog_post_show', {'id': 1}) }}">Blog Post</a>
```

Assets

`asset('/images/logo.png')`

Requires to install *asset* package `composer require symfony/asset`

Using `asset` you make sure the URLs to your assets are always rendered correctly according to your application configuration.

IMPORTANT!
It’s supports cache busting (making sure user browsers will fetch updated resources after deployment).

The `framework.assets.version` setting needs to be updated before (or during) each deployment.

```yaml
# config/packages/framework.yaml
framework:
    # ...
    assets:
        version: 'v2'
```

Request, User, Session

Escape output

Twig will do automatic output escaping of everything your render:

```
{{ description }}  {# I &lt;3 this product #}
```

To render the content raw (as it is, including HTML) use the `raw` filter:

```
{# disable output escaping with the raw filter #}
{{ description|raw }}  {# I <3 this product #}
```

Global variables

`app` variable is always available in Twig templates (it’s being set by Symfony).

Variable	What’s inside?	What that is?
`app.user`	`UserInterface`, `string` or `null`	Currently authenticated user
`app.request`	`Request`	Current request
`app.session`	`Session` or `null`	Current session or `null` if none
`app.environment`	`string`	Current environment (`dev`, `test`, `prod`)