Building Websites with Zend Expressive 3

Rob Allen, Nineteen Feet

February 2018 ~ @akrabat

A microframework with full stack components

µFramework core

- Router
- Container
- Template renderer
- Error handler
- Configuration

Ecosystem

- Filtering and validation
- API rendering: HAL & Problem-API
- Database abstraction
- Session handling
- Logging
- Mail
- Pagination
- Caching

Agnostic

Router:

FastRoute, Aura.Router or Zend Router

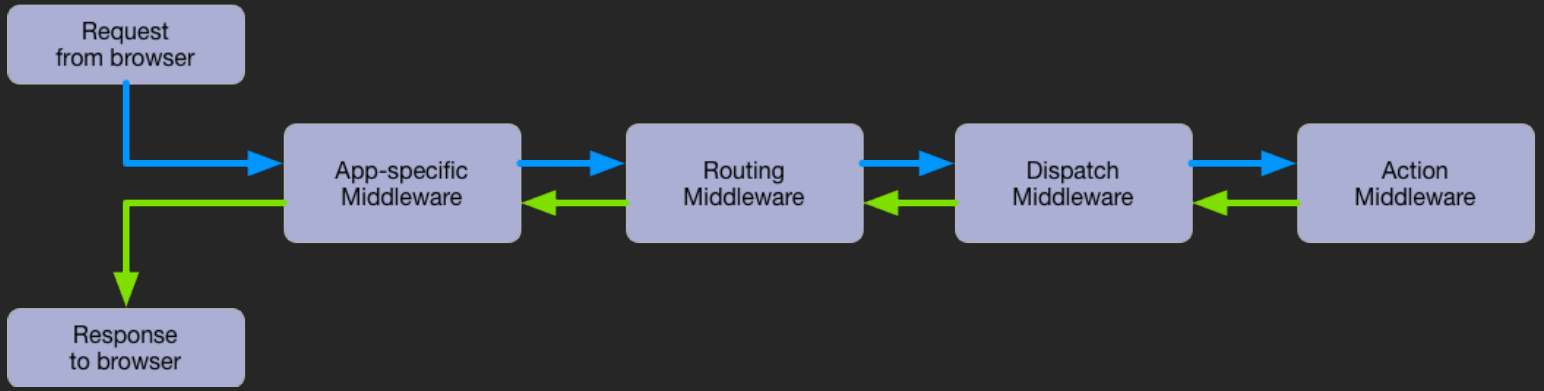
DI Container:

Aura.Di, Auryn, Pimple, Symfony DI Container or Zend-ServiceManager

Template:

Plates, Twig or Zend View

Middleware pipeline



PSR-15 MiddlewareInterface

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ) : ResponseInterface;
}
```

PSR-15 MiddlewareInterface

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ) : ResponseInterface;
}
```

PSR-15 MiddlewareInterface

```php
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ) : ResponseInterface;
}
```

PSR-15 MiddlewareInterface

```
namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;

interface MiddlewareInterface
{
    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ) : ResponseInterface;
}
```

Structure of middleware

```php
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

Structure of middleware

```
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

Structure of middleware

```
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

Structure of middleware

```
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

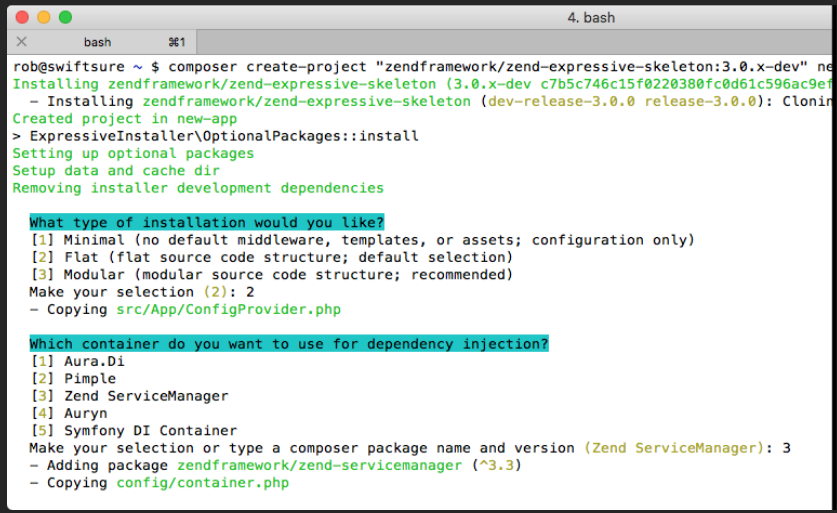
Structure of middleware

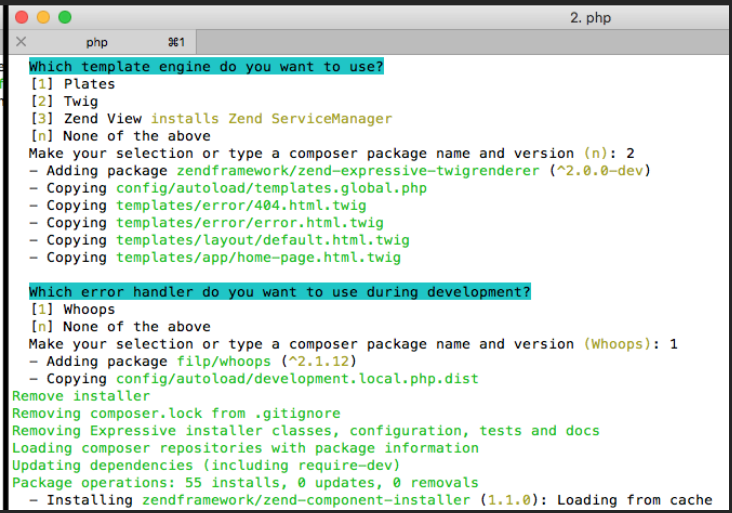
```
class TimerMiddleware implements MiddlewareInterface
{
    public function process($request, $handler)
    {
        $start = microtime(true);

        $response = $handler->handle($request);

        $taken = microtime(true) - $start;
        $response->getBody()->write("<!-- Time: $taken -->");

        return $response;
    }
}
```

Getting started

Skeleton

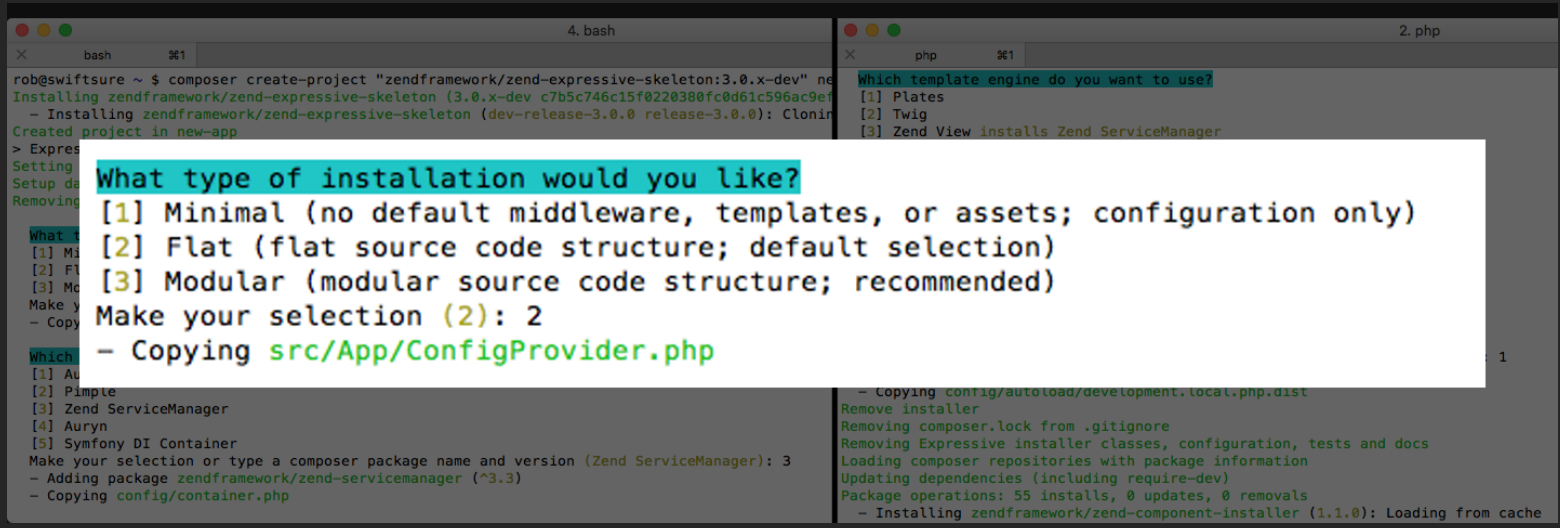
```
$ composer create-project zendframework/zend-expressive-skeleton new-app
```



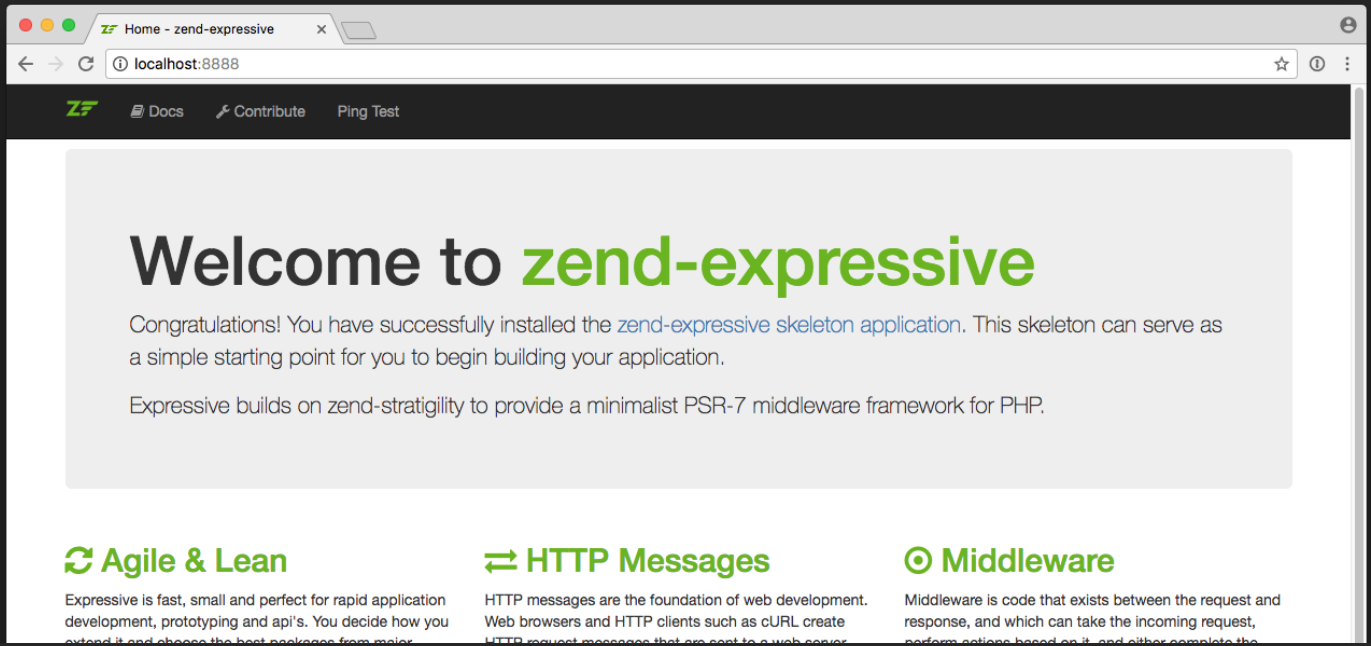


Skeleton

```
$ composer create-project zendframework/zend-expressive-skeleton new-app
```



Skeleton


Home - zend-expressive
localhost:8888
Docs
Contribute
Ping Test
Welcome to zend-expressive
Congratulations! You have successfully installed the zend-expressive skeleton application. This skeleton can serve as a simple starting point for you to begin building your application.
Expressive builds on zend-stratigility to provide a minimalist PSR-7 middleware framework for PHP.
Agile & Lean
Expressive is fast, small and perfect for rapid application development, prototyping and api's. You decide how you
HTTP Messages
HTTP messages are the foundation of web development. Web browsers and HTTP clients such as cURL create
Middleware
Middleware is code that exists between the request and response, and which can take the incoming request,

Directory structure

```
.
├── bin/
├── config/
│   ├── autoload/
│   │   ├── dependencies.global.php
│   │   ├── development.local.php
│   │   ├── development.local.php.dist
│   │   ├── local.php.dist
│   │   ├── router.global.php
│   │   ├── templates.global.php
│   │   └── zend-expressive.global.php
│   ├── config.php
│   ├── container.php
│   ├── development.config.php
│   ├── development.config.php.dist
│   ├── pipeline.php
│   └── routes.php
├── data/
├── public/
│   ├── css/
│   ├── js/
│   └── index.php
├── src/
│   └── App/
├── test/
│   └── AppTest/
├── vendor/
├── composer.json
├── composer.lock
├── phpcs.xml.dist
└── phpunit.xml.dist
```

src/App directory

- Each module lives in its own namespace
- Contains all code for application
- `ConfigProvider` class for initialisation
    - Configuration
    - DI registration

src/App directory structure

```
└── src/
    └── App/
        ├── src/
        │   ├── Handler/
        │   │   ├── HomePageFactory.php
        │   │   ├── HomePageHandler.php
        │   │   └── PingHandler.php
        │   └── ConfigProvider.php
        ├── templates/
        │   ├── app/
        │   │   └── home-page.html.twig
        │   ├── error/
        │   │   ├── 404.html.twig
        │   │   └── error.html.twig
        │   └── layout/
        │       └── default.html.twig
        └── test/
            └── AppTest/
                └── Handler/
```

A handler (AKA: an action)

```php
namespace App\Handler;

use ...;

class HomePageHandler implements RequestHandlerInterface
{
    public function handle(
        ServerRequestInterface $request
    ) : ResponseInterface {

        return new HtmlResponse('<p>Hello World</p>');
    }
}
```

Let's write a web page!

Bitcoin conversion

Create a page that displays the current value of 1 Bitcoin in £, $ & €

Create a route

config/routes.php:

```
$app->get(
    '/bitcoin',
    App\Handler\BitcoinPageHandler::class,
    'bitcoin'
);
```

Routes have a method

config/routes.php:

```
$app->get(
    '/bitcoin',
    App\Handler\BitcoinPageHandler::class,
    'bitcoin'
);
```

HTTP Method

```
$app->get()
$app->post()
$app->put()
$app->patch()
$app->delete()
```

Multiple methods:

```
$app->any()
$app->route(…, …, ['GET', 'POST'], …);
```

Routes have a pattern

config/routes.php:

```
$app->get(
    '/bitcoin',
    App\Handler\BitcoinPageHandler::class,
    'bitcoin'
);
```

FastRoute URI pattern

- Literal string match

```
$app->get('/hello', …);
```

FastRoute URI pattern

- Literal string match

```
$app->get('/hello', …);
```

- Placeholders are wrapped in { }

```
$app->get('/hello/{name}', …);
```

FastRoute URI pattern

- Literal string match
```
$app->get('/hello', …);
```
- Placeholders are wrapped in { }
```
$app->get('/hello/{name}', …);
```
- Optional segments are wrapped with [ ]
```
$app->get('/news[/{year}[/{month}]]', …);
```

FastRoute URI pattern

- Literal string match
```
$app->get('/hello', …);
```
- Placeholders are wrapped in { }
```
$app->get('/hello/{name}', …);
```
- Optional segments are wrapped with [ ]
```
$app->get('/news[/{year}[/{month}]]', …);
```
- Constrain placeholders via Regex
```
$app->get('/news/{year:\d{4}}}', …); // exactly 4 digits
```

Routes have a name

config/routes.php:

```
$app->get(
    '/bitcoin',
    App\Handler\BitcoinPageHandler::class,
    'bitcoin'
);
```

Url helper

- Builds URLs from route names
- Can be helpful to use `.` to group related route names

Use the `router`:

```
$router->generateUri('user.profile', ['name' => 'Rob']);
```

or in the template:

```
{{ path('user.profile', {'name': 'Rob'}) }}
```

Routes have a handler

config/routes.php:

```
$app->get(
    '/bitcoin',
    App\Handler\BitcoinPageHandler::class,
    'bitcoin'
);
```

Handlers

- Receive a PSR-7 `Request`
- Manage business logic operations
- Must return a PSR-7 `Response`
- Implemented as PSR-15 `RequestHandler`
- Create using CLI tool:

```
$ composer expressive handler:create \
  App\\Handler\\BitcoinPageHandler
```

Bitcoin handler

```
class BitcoinPageHandler implements RequestHandlerInterface
{
  public function handle(
    ServerRequestInterface $request
  ) : ResponseInterface {

    $data['prices'] = $this->btcService->getCurrentPrices();

    return new HtmlResponse(
      $this->template->render('app::bitcoin-page', $data)
    );
  }
```

Bitcoin handler

```
class BitcoinPageHandler implements RequestHandlerInterface
{
  public function handle(
    ServerRequestInterface $request
  ) : ResponseInterface {

    $data['prices'] = $this->btcService->getCurrentPrices();

    return new HtmlResponse(
      $this->template->render('app::bitcoin-page', $data)
    );
  }
```

Bitcoin handler

```
class BitcoinPageHandler implements RequestHandlerInterface
{
  public function handle(
    ServerRequestInterface $request
  ) : ResponseInterface {

    $data['prices'] = $this->btcService->getCurrentPrices();

    return new HtmlResponse(
      $this->template->render('app::bitcoin-page', $data)
    );
  }
```

Bitcoin handler

```
class BitcoinPageHandler implements RequestHandlerInterface
{
  public function handle(
    ServerRequestInterface $request
  ) : ResponseInterface {

    $data['prices'] = $this->btcService->getCurrentPrices();

    return new HtmlResponse(
      $this->template->render('app::bitcoin-page', $data)
    );
  }
```

Injecting dependencies

```
class BitcoinPageHandler implements RequestHandlerInterface
{
  protected $template;
  protected $btcService;

  public function __construct(
    TemplateRendererInterface $template,
    BitcoinService $btcService
  ) {
      $this->template = $template;
      $this->btcService = $btcService;
  }

  // …
```

Expressive configuration

A mushed-up associative array created from:

- Invoked `ConfigProvider` classes from libs and modules
- Config files in `config/autoload/`

Common top level keys:

- `dependencies`
- `twig`
- `validators`
- `cache`
- `templates`
- `filters`
- `db`

ConfigProvider

```php
namespace App;

class ConfigProvider
{
    public function __invoke() : array
    {
        return [
            'dependencies' => $this->getDependencies(),
            'templates'    => $this->getTemplates(),
        ];
    }
```

ConfigProvider

```php
namespace App;

class ConfigProvider
{
    public function __invoke() : array
    {
        return [
            'dependencies' => $this->getDependencies(),
            'templates'    => $this->getTemplates(),
        ];
    }
```

Dependency configuration

```
public function getDependencies() : array
{
    return [
        'factories'  => [
            Handler\BitcoinPageHandler::class =>
                Handler\BitcoinPageFactory::class,
        ],
    ];
}
```

Dependency configuration

```
public function getDependencies() : array
{
    return [
        'factories'  => [
            Handler\BitcoinPageHandler::class =>
                Handler\BitcoinPageFactory::class,
        ],
    ];
}
```

Dependency configuration

```
public function getDependencies() : array
{
    return [
        'factories'  => [
            Handler\BitcoinPageHandler::class =>
                Handler\BitcoinPageFactory::class,
        ],
    ];
}
```

Handler factory

Return an instance of the handler with its injected dependencies

```
class BitcoinPageFactory
{
    public function __invoke($container)
    {
        return new BitcoinPageHandler(
            $container->get(TemplateRendererInterface::class),
            $container->get(BitcoinService::class)
        );
    }
}
```

Handler factory

Return an instance of the handler with its injected dependencies

```
class BitcoinPageFactory
{
    public function __invoke($container)
    {
        return new BitcoinPageHandler(
            $container->get(TemplateRendererInterface::class),
            $container->get(BitcoinService::class)
        );
    }
}
```

Handler factory

Return an instance of the handler with its injected dependencies

```
class BitcoinPageFactory
{
    public function __invoke($container)
    {
        return new BitcoinPageHandler(
            $container->get(TemplateRendererInterface::class),
            $container->get(BitcoinService::class)
        );
    }
}
```

Templates

Expressive's view layer

Templating

- Render method to convert page:

```
$html = $this->template->render('app::bitcoin-page', $data);
```

- Templates are namespaced: `namespace::template`
- Extension is resolved by the adapter:

```
app::bitcoin-page => app/bitcoin-page.html.twig
```

Twig templates

"Twig is a modern template engine for PHP"

- Manual: https://twig.symfony.com
- Script extension: *.twig*
- Variables: `{{ }}`
- Control statements: `{% %}`
- Comments: `{# #}`

Action template

```
{% extends '@layout/default.html.twig' %}

{% block title %}Bitcoin converter{% endblock %}

{% block content %}
    <h1>Bitcoin converter</h1>

    <p>One BTC:</p>
    {% for price in prices %}
        {{ price.symbol }}
        {{ price.rate_float|number_format(2, '.', ',') }}<br>
    {% endfor %}
{% endblock %}
```

Print variables

```
{% extends '@layout/default.html.twig' %}

{% block title %}Bitcoin converter{% endblock %}

{% block content %}
    <h1>Bitcoin converter</h1>

    <p>One BTC:</p>
    {% for price in prices %}
        {{ price.symbol }}
        {{ price.rate_float|number_format(2, '.', ',') }}<br>
    {% endfor %}
{% endblock %}
```

Control statements

```
{% extends '@layout/default.html.twig' %}

{% block title %}Bitcoin converter{% endblock %}

{% block content %}
    <h1>Bitcoin converter</h1>

    <p>One BTC:</p>
    {% for price in prices %}
        {{ price.symbol }}
        {{ price.rate_float|number_format(2, '.', ',') }}<br>
    {% endfor %}
{% endblock %}
```

Template inheritance

```
{% extends '@layout/default.html.twig' %}

{% block title %}Bitcoin converter{% endblock %}

{% block content %}
    <h1>Bitcoin converter</h1>

    <p>One BTC:</p>
    {% for price in prices %}
        {{ price.symbol }}
        {{ price.rate_float|number_format(2, '.', ',') }}<br>
    {% endfor %}
{% endblock %}
```

Template inheritance

- For cohesive look and feel
    - includes default CSS, JS & structural HTML
- Build a base skeleton
- Define *blocks* for children to override
- Each child chooses which template to inherit from

Base skeleton

src/App/templates/layout/default.html.twig:

```
<!DOCTYPE html>
<html>
  <head>
    {% block head %}
        <link rel="stylesheet" href="style.css" />
        <title>{% block title %}{% endblock %} - Akrabat</title>
    {% endblock %}
  </head>
  <body>
    {% block content %}{% endblock %}
    <footer>{% block footer %}&copy; 2017{% endblock %}</footer>
  </body>
</html>
```

Base skeleton

src/App/templates/layout/default.html.twig:

```
<!DOCTYPE html>
<html>
  <head>
    {% block head %}
        <link rel="stylesheet" href="style.css" />
        <title>{% block title %}{% endblock %} - Akrabat</title>
    {% endblock %}
  </head>
  <body>
    {% block content %}{% endblock %}
    <footer>{% block footer %}&copy; 2017{% endblock %}</footer>
  </body>
</html>
```

Base skeleton

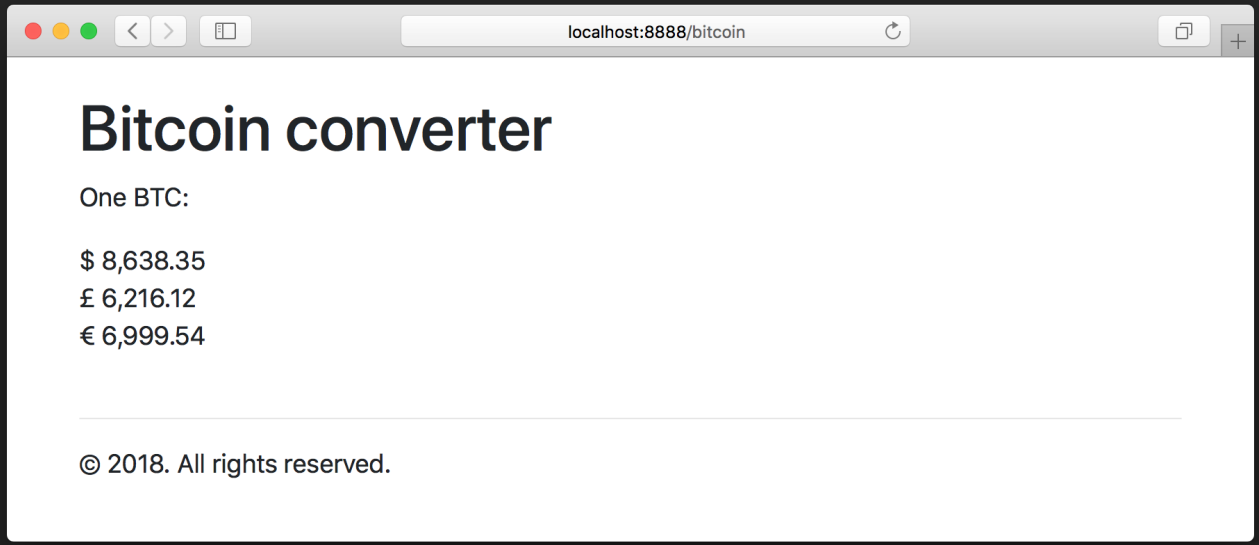
src/App/templates/layout/default.html.twig:

```
<!DOCTYPE html>
<html>
  <head>
    {% block head %}
        <link rel="stylesheet" href="style.css" />
        <title>{% block title %}{% endblock %} - Akrabat</title>
    {% endblock %}
  </head>
  <body>
    {% block content %}{% endblock %}
    <footer>{% block footer %}&copy; 2017{% endblock %}</footer>
  </body>
</html>
```

Base skeleton

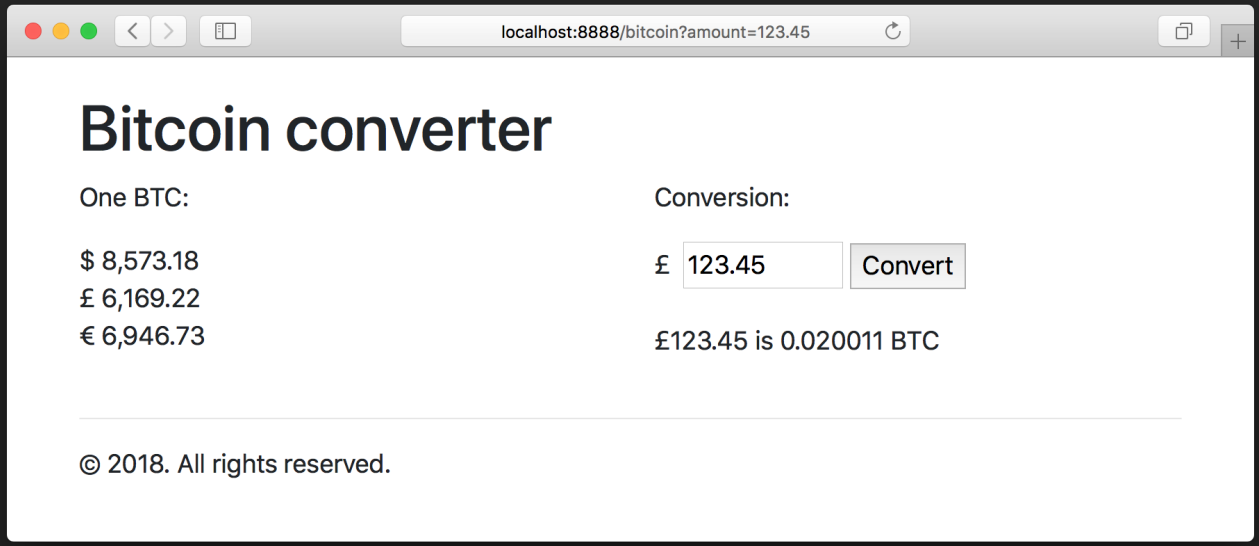
src/App/templates/layout/default.html.twig:

```
<!DOCTYPE html>
<html>
  <head>
    {% block head %}
        <link rel="stylesheet" href="style.css" />
        <title>{% block title %}{% endblock %} - Akrabat</title>
    {% endblock %}
  </head>
  <body>
    {% block content %}{% endblock %}
    <footer>{% block footer %}&copy; 2017{% endblock %}</footer>
  </body>
</html>
```


localhost:8888/bitcoin
Bitcoin converter
One BTC:
$ 8,638.35
£ 6,216.12
€ 6,999.54
© 2018. All rights reserved.

Adding components

Add a form


localhost:8888/bitcoin?amount=123.45
Bitcoin converter
One BTC:
$ 8,573.18
£ 6,169.22
€ 6,946.73
Conversion:
£ 123.45
Convert
£123.45 is 0.020011 BTC
© 2018. All rights reserved.

Add a form: HTML

```
<form method="GET" action="/bitcoin">
    <label>&pound;</label>
    <input name="amount" value="{{ amount }}">
    <button>Convert</button>
</form>

<p>&pound;{{amount}} is {{ bitcoins|number_format(6) }} BTC
```

Add a form: HTML

```
<form method="GET" action="/bitcoin">
    <label>&pound;</label>
    <input name="amount" value="{{ amount }}">
    <button>Convert</button>
</form>

<p>&pound;{{amount}} is {{ bitcoins|number_format(6) }} BTC
```

Add a form: HTML

```
<form method="GET" action="/bitcoin">
    <label>&pound;</label>
    <input name="amount" value="{{ amount }}">
    <button>Convert</button>
</form>

<p>&pound;{{amount}} is {{ bitcoins|number_format(6) }} BTC
```

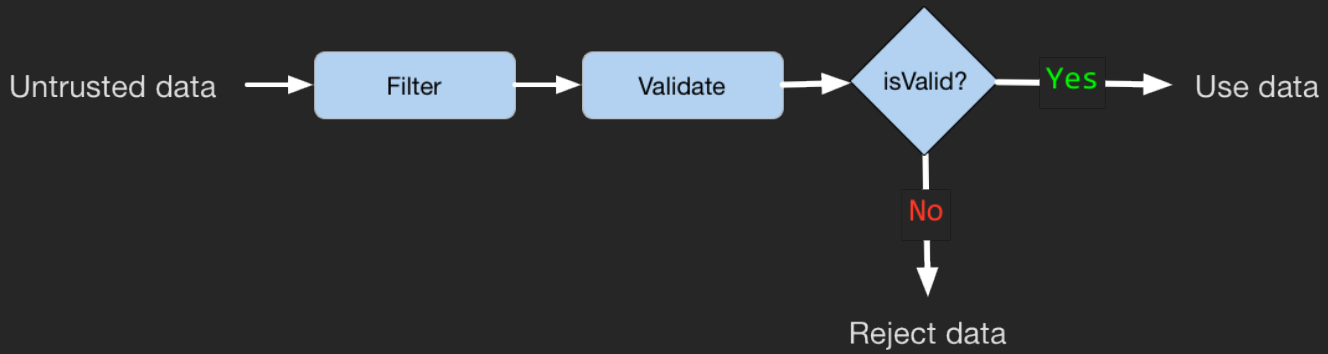
Add a form: HTML

```
<form method="GET" action="/bitcoin">
    <label>&pound;</label>
    <input name="amount" value="{{ amount }}">
    <button>Convert</button>
</form>

<p>&pound;{{amount}} is {{ bitcoins|number_format(6) }} BTC
```

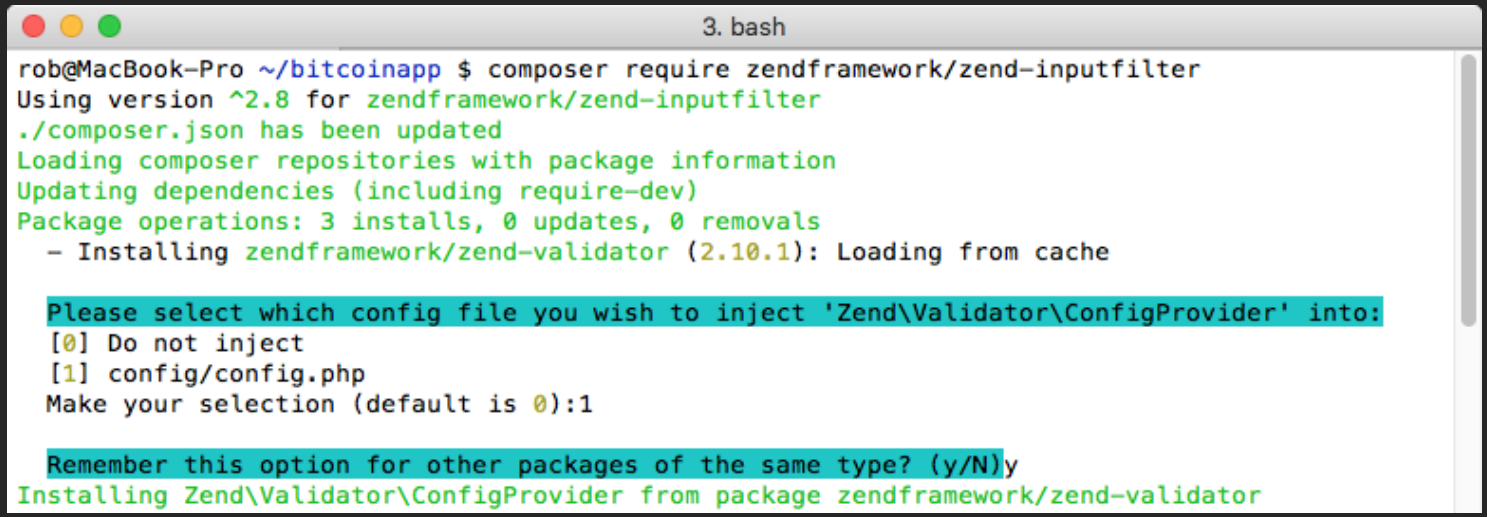
Add a form: HTML

```
<form method="GET" action="/bitcoin">
    <label>&pound;</label>
    <input name="amount" value="{{ amount }}">
    <button>Convert</button>
</form>

<p>&pound;{{amount}} is {{ bitcoins|number_format(6) }} BTC</p>
```

Zend-InputFilter



Integration via ConfigProvider



Create an input filter

```
$factory = new Zend\InputFilter\Factory();
$inputFilter = $factory->createInputFilter([
    'amount' => [
        'filters' => [
            ['name' => 'NumberParse'],
        ],
        'validators' => [
            [
                'name' => 'GreaterThan',
                'options' => ['min' => 0],
            ],
        ]
    ],
]);
```

Create an input filter

```
$factory = new Zend\InputFilter\Factory();
$inputFilter = $factory->createInputFilter([
    'amount' => [
        'filters' => [
            ['name' => 'ToInt'],
        ],
        'validators' => [
            [
                'name' => 'GreaterThan',
                'options' => ['min' => 0],
            ],
        ]
    ],
]);
```

Create an input filter

```
$factory = new Zend\InputFilter\Factory();
$inputFilter = $factory->createInputFilter([
    'amount' => [
        'filters' => [
            ['name' => 'ToInt'],
        ],
        'validators' => [
            [
                'name' => 'GreaterThan',
                'options' => ['min' => 0],
            ],
        ]
    ],
]);
```

Create an input filter

```
$factory = new Zend\InputFilter\Factory();
$inputFilter = $factory->createInputFilter([
    'amount' => [
        'filters' => [
            ['name' => 'ToInt'],
        ],
        'validators' => [
            [
                'name' => 'GreaterThan',
                'options' => ['min' => 0],
            ],
        ]
    ],
]);
```

Create an input filter

```
$factory = new Zend\InputFilter\Factory();
$inputFilter = $factory->createInputFilter([
    'amount' => [
        'filters' => [
            ['name' => 'ToInt'],
        ],
        'validators' => [
            [
                'name' => 'GreaterThan',
                'options' => ['min' => 0],
            ],
        ]
    ],
]);
```

Validating request data

1. Retrieve data from `Request` object
2. Pass to `InputFilter`
3. Call `isValid()`
4. Retrieve sanitized, valid data using `getValues()`
5. Use `getMessages()` to find out what failed

Validating request data

```
public function handle(ServerRequestInterface $request)
{
    $requestData = $request->getQueryParams();
    $this->inputFilter->setData($requestData);

    if ($this->inputFilter->isValid()) {
        /* request data is valid */
        $values = $this->inputFilter->getValues();
        $btc = $this->btcService->convert($values['amount']);
    } else {
        /* request data is invalid */
        $errors = $this->inputFilter->getMessages();
    }
```

Validating request data

```
public function handle(ServerRequestInterface $request)
{
    $requestData = $request->getQueryParams();
    $this->inputFilter->setData($requestData);

    if ($this->inputFilter->isValid()) {
        /* request data is valid */
        $values = $this->inputFilter->getValues();
        $btc = $this->btcService->convert($values['amount']);
    } else {
        /* request data is invalid */
        $errors = $this->inputFilter->getMessages();
    }
```

Validating request data

```
public function handle(ServerRequestInterface $request)
{
    $requestData = $request->getQueryParams();
    $this->inputFilter->setData($requestData);

    if ($this->inputFilter->isValid()) {
        /* request data is valid */
        $values = $this->inputFilter->getValues();
        $btc = $this->btcService->convert($values['amount']);
    } else {
        /* request data is invalid */
        $errors = $this->inputFilter->getMessages();
    }
```

Validating request data

```
public function handle(ServerRequestInterface $request)
{
    $requestData = $request->getQueryParams();
    $this->inputFilter->setData($requestData);

    if ($this->inputFilter->isValid()) {
        /* request data is valid */
        $values = $this->inputFilter->getValues();
        $btc = $this->btcService->convert($values['amount']);
    } else {
        /* request data is invalid */
        $errors = $this->inputFilter->getMessages();
    }
```

Validating request data

```
public function handle(ServerRequestInterface $request)
{
    $requestData = $request->getQueryParams();
    $this->inputFilter->setData($requestData);

    if ($this->inputFilter->isValid()) {
        /* request data is valid */
        $values = $this->inputFilter->getValues();
        $btc = $this->btcService->convert($values['amount']);
    } else {
        /* request data is invalid */
        $errors = $this->inputFilter->getMessages();
    }
```

Summary

Resources

- https://docs.zendframework.com/zend-expressive/
- https://github.com/zendframework/
- https://akrabat.com/category/zend-expressive/
- https://framework.zend.com/blog
- *Zend Expressive Essentials* by Matt Setter

Thank you!

https://joind.in/talk/52ee1

Rob Allen ~ @akrabat