The Front Controller

Up until now, our application is simplistic as there is only one page. To spice things up a little bit, let's go crazy and add another page that says goodbye::

```
<?php

// framework/bye.php

require_once __DIR__.'/vendor/autoload.php';

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

$request = Request::createFromGlobals();

$response = new Response('Goodbye!');
$response->send();
```

As you can see for yourself, much of the code is exactly the same as the one we have written for the first page. Let's extract the common code that we can share between all our pages. Code sharing sounds like a good plan to create our first "real" framework!

The PHP way of doing the refactoring would probably be the creation of an include file::

```
<?php

// framework/init.php

require_once __DIR__.'/vendor/autoload.php';

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

$request = Request::createFromGlobals();
$response = new Response();
```

Let's see it in action::

```
<?php

// framework/index.php

require_once __DIR__.'/init.php';

$input = $request->get('name', 'World');

$response->setContent(sprintf('Hello %s', htmlspecialchars($input, ENT_QUOTES, 'UTF-8')));
$response->send();
```

And for the "Goodbye" page::

```
<?php

// framework/bye.php
```

```
require_once __DIR__.'/init.php';

$response->setContent('Goodbye!');
$response->send();
```

We have indeed moved most of the shared code into a central place, but it does not feel like a good abstraction, does it? We still have the send() method for all pages, our pages do not look like templates, and we are still not able to test this code properly.

Moreover, adding a new page means that we need to create a new PHP script, which name is exposed to the end user via the URL (http://127.0.0.1:4321/bye.php): there is a direct mapping between the PHP script name and the client URL. This is because the dispatching of the request is done by the web server directly. It might be a good idea to move this dispatching to our code for better flexibility. This can be easily achieved by routing all client requests to a single PHP script.

.. tip::

Exposing a single PHP script to the end user is a design pattern called
the "`front controller`_".

Such a script might look like the following::

```
<?php

// framework/front.php

require_once __DIR__.'/vendor/autoload.php';

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

$request = Request::createFromGlobals();
$response = new Response();

$map = array(
    '/hello' => __DIR__.'/hello.php',
    '/bye'   => __DIR__.'/bye.php',
);

$path = $request->getPathInfo();
if (isset($map[$path])) {
    require $map[$path];
} else {
    $response->setStatusCode(404);
    $response->setContent('Not Found');
}

$response->send();
```

And here is for instance the new hello.php script::

```
<?php

// framework/hello.php

$input = $request->get('name', 'World');
```

```
$response->setContent(sprintf('Hello %s', htmlspecialchars($input, ENT_QUOTES, 'UTF-8')));
```

In the `front.php` script, `$map` associates URL paths with their corresponding PHP script paths.

As a bonus, if the client asks for a path that is not defined in the URL map, we return a custom 404 page; you are now in control of your website.

To access a page, you must now use the `front.php` script:

- http://127.0.0.1:4321/front.php/hello?name=Fabien
- http://127.0.0.1:4321/front.php/bye

`/hello` and `/bye` are the page paths.

.. tip::

Most web servers like Apache or nginx are able to rewrite the incoming URLs
and remove the front controller script so that your users will be able to
type ``http://127.0.0.1:4321/hello?name=Fabien``, which looks much better.

The trick is the usage of the `Request::getPathInfo()` method which returns the path of the Request by removing the front controller script name including its sub-directories (only if needed -- see above tip).

.. tip::

You don't even need to setup a web server to test the code. Instead,
replace the ``$request = Request::createFromGlobals();`` call to something
like ``$request = Request::create('/hello?name=Fabien');`` where the
argument is the URL path you want to simulate.

Now that the web server always access the same script (`front.php`) for all pages, we can secure the code further by moving all other PHP files outside the web root directory:

.. code-block:: text

```
example.com
â”œâ”€â”€ composer.json
â”‚   src
â”‚   â”œâ”€â”€ pages
â”‚       â”œâ”€â”€ hello.php
â”‚       â””â”€â”€ bye.php
â”œâ”€â”€ vendor
â””â”€â”€ web
    â””â”€â”€ front.php
```

Now, configure your web server root directory to point to `web/` and all other files won't be accessible from the client anymore.

To test your changes in a browser (`http://localhost:4321/?name=Fabien`), run the PHP built-in server:

.. code-block:: sh

```
$ php -S 127.0.0.1:4321 -t web/ web/front.php
```

.. note::

For this new structure to work, you will have to adjust some paths in
various PHP files; the changes are left as an exercise for the reader.

The last thing that is repeated in each page is the call to `setContent()`. We can convert all pages to "templates" by just echoing the content and calling the `setContent()` directly from the front controller script::

```
<?php

// example.com/web/front.php

// ...

$path = $request->getPathInfo();
if (isset($map[$path])) {
    ob_start();
    include $map[$path];
    $response->setContent(ob_get_clean());
} else {
    $response->setStatusCode(404);
    $response->setContent('Not Found');
}

// ...
```

And the `hello.php` script can now be converted to a template::

```
<!-- example.com/src/pages/hello.php -->

<?php $name = $request->get('name', 'World') ?>

Hello <?php echo htmlspecialchars($name, ENT_QUOTES, 'UTF-8') ?>
```

We have the first version of our framework::

```
<?php

// example.com/web/front.php

require_once __DIR__.'/../vendor/autoload.php';

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

$request = Request::createFromGlobals();
$response = new Response();

$map = array(
    '/hello' => __DIR__.'/../src/pages/hello.php',
    '/bye'   => __DIR__.'/../src/pages/bye.php',
);

$path = $request->getPathInfo();
if (isset($map[$path])) {
    ob_start();
    include $map[$path];
    $response->setContent(ob_get_clean());
```

```
} else {
    $response->setStatusCode(404);
    $response->setContent('Not Found');
}

$response->send();
```

Adding a new page is a two step process: add an entry in the map and create a PHP template in `src/pages/`. From a template, get the Request data via the `$request` variable and tweak the Response headers via the `$response` variable.

.. note::

If you decide to stop here, you can probably enhance your framework by
extracting the URL map to a configuration file.

.. _front controller: http://symfony.com/doc/current/book/from_flat_php_to_symfony2.html#a-front-controller-to-the-rescue