Templating

The astute reader has noticed that our framework hardcodes the way specific "code" (the templates) is run. For simple pages like the ones we have created so far, that's not a problem, but if you want to add more logic, you would be forced to put the logic into the template itself, which is probably not a good idea, especially if you still have the separation of concerns principle in mind.

Let's separate the template code from the logic by adding a new layer: the controller:*The controller's mission is to generate a Response based on the information conveyed by the client's Request.*

Change the template rendering part of the framework to read as follows::

```
<?php

// example.com/web/front.php

// ...

try {
    $request->attributes->add($matcher->match($request->getPathInfo()));
    $response = call_user_func('render_template', $request);
} catch (Routing\Exception\ResourceNotFoundException $e) {
    $response = new Response('Not Found', 404);
} catch (Exception $e) {
    $response = new Response('An error occurred', 500);
}
```

As the rendering is now done by an external function (`render_template()` here), we need to pass to it the attributes extracted from the URL. We could have passed them as an additional argument to `render_template()`, but instead, let's use another feature of the`Request` class called *attributes*: Request attributes is a way to attach additional information about the Request that is not directly related to the HTTP Request data.

You can now create the`render_template()` function, a generic controller that renders a template when there is no specific logic. To keep the same template as before, request attributes are extracted before the template is rendered::

```
function render_template($request)
{
    extract($request->attributes->all(), EXTR_SKIP);
    ob_start();
    include sprintf(__DIR__.'/../src/pages/%s.php', $_route);

    return new Response(ob_get_clean());
}
```

As `render_template` is used as an argument to the PHP`call_user_func()` function, we can replace it with any valid PHP `callbacks`_. This allows us to use a function, an anonymous function, or a method of a class as a controller... your choice.

As a convention, for each route, the associated controller is configured via the`_controller` route

attribute::

```
$routes->add('hello', new Routing\Route('/hello/{name}', array(
    'name' => 'World',
    '_controller' => 'render_template',
)));

try {
    $request->attributes->add($matcher->match($request->getPathInfo()));
    $response = call_user_func($request->attributes->get('_controller'), $request);
} catch (Routing\Exception\ResourceNotFoundException $e) {
    $response = new Response('Not Found', 404);
} catch (Exception $e) {
    $response = new Response('An error occurred', 500);
}
```

A route can now be associated with any controller and of course, within a controller, you can still use the render_template() to render a template::

```
$routes->add('hello', new Routing\Route('/hello/{name}', array(
    'name' => 'World',
    '_controller' => function ($request) {
        return render_template($request);
    }
)));
```

This is rather flexible as you can change the Response object afterwards and you can even pass additional arguments to the template::

```
$routes->add('hello', new Routing\Route('/hello/{name}', array(
    'name' => 'World',
    '_controller' => function ($request) {
        // $foo will be available in the template
        $request->attributes->set('foo', 'bar');

        $response = render_template($request);

        // change some header
        $response->headers->set('Content-Type', 'text/plain');

        return $response;
    }
)));
```

Here is the updated and improved 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;
use Symfony\Component\Routing;

function render_template($request)
{
    extract($request->attributes->all(), EXTR_SKIP);
```

```
    ob_start();
    include sprintf(__DIR__.'/../src/pages/%s.php', $_route);

    return new Response(ob_get_clean());
}

$request = Request::createFromGlobals();
$routes = include __DIR__.'/../src/app.php';

$context = new Routing\RequestContext();
$context->fromRequest($request);
$matcher = new Routing\Matcher\UrlMatcher($routes, $context);

try {
    $request->attributes->add($matcher->match($request->getPathInfo()));
    $response = call_user_func($request->attributes->get('_controller'), $request);
} catch (Routing\Exception\ResourceNotFoundException $e) {
    $response = new Response('Not Found', 404);
} catch (Exception $e) {
    $response = new Response('An error occurred', 500);
}

$response->send();
```

To celebrate the birth of our new framework, let's create a brand new application that needs some simple logic. Our application has one page that says whether a given year is a leap year or not. When calling `/is_leap_year`, you get the answer for the current year, but you can also specify a year like in `/is_leap_year/2009`. Being generic, the framework does not need to be modified in any way, just create a new `app.php` file::

```
<?php

// example.com/src/app.php

use Symfony\Component\Routing;
use Symfony\Component\HttpFoundation\Response;

function is_leap_year($year = null) {
    if (null === $year) {
        $year = date('Y');
    }

    return 0 == $year % 400 || (0 == $year % 4 && 0 != $year % 100);
}

$routes = new Routing\RouteCollection();
$routes->add('leap_year', new Routing\Route('/is_leap_year/{year}', array(
    'year' => null,
    '_controller' => function ($request) {
        if (is_leap_year($request->attributes->get('year'))) {
            return new Response('Yep, this is a leap year!');
        }

        return new Response('Nope, this is not a leap year.');
    }
)));

return $routes;
```

The is_leap_year() function returns true when the given year is a leap year,false otherwise. If the year is null, the current year is tested. The controller is simple: it gets the year from the request attributes, pass it to the is_leap_year() function, and according to the return value it creates a new Response object.

As always, you can decide to stop here and use the framework as is; it's probably all you need to create simple websites like those fancy one-page websites_ and hopefully a few others.

.. _callbacks: http://php.net/callback#language.types.callback .. _websites: http://kottke.org/08/02/single-serving-sites