Getting Started

Aura.Router is a web router implementation for PSR-7.

You get all the router objects through a library-specific *RouterContainer*, which manages object creation, dependencies, and wiring for you. That means you need to instantiate the container before anything else.

```
<?php
use Aura\Router\RouterContainer;

$routerContainer = new RouterContainer();
?>
```

You can then retrieve a *Map* for adding routes, a *Matcher* for matching the incoming request to a route, and a *Generator* for generating paths from routes.

Let's go step-by-step to add a route, match a request against it, and dispatch it. A full working example is provided at the end of this page.

Adding A Route

To add a route, first retrieve the *Map* from the *RouterContainer*.

```
<?php
$map = $routerContainer->getMap();
?>
```

Then call one of its route-adding methods:

- `$map->get()` adds a GET route
- `$map->post()` adds a POST route
- `$map->patch()` adds a PATCH route
- `$map->delete()` adds a DELETE route
- `$map->options()` adds a OPTIONS route
- `$map->head()` adds a HEAD route

Each route-adding method takes three parameters:

1. A `$name` (for when you need to generate a link from the route)
2. A `$path` (with optional, named token placeholders)
3. An optional `$handler` (a closure, callback, action object, controller class, etc); if you do not pass a handler, the route will use the `$name` parameter as the handler.

For example, this route named `blog.read` will match against a `GET` request on the path `/blog/42` (or any other `{id}` value). It also defines a closure as a handler for the route, using a *ServerRequestInterface* instance and a *ResponseInterface* instance as arguments.

```
<?php
$map->get('blog.read', '/blog/{id}', function ($request, $response) {
    $id = (int) $request->getAttribute('id');
    $response->getBody()->write("You asked for blog entry {$id}.");
    return $response;
});
?>
```

If you want to add a route with a custom HTTP verb, call `$map->route()` and follow with a fluent call to `allows()`:

```
<?php
$map->route('route-name', '/route/path', function () { ... })
    ->allows('CUSTOMVERB');
?>
```

Matching A Request To A Route

First, get the *Matcher* from the *RouterContainer*.

```
<?php
$matcher = $routerContainer->getMatcher();
?>
```

Then call the `match()` method to match a PSR-7 *ServerRequestInterface* instance to a mapped *Route*.

For this you need an implementation of psr-7 .

The most widely used one is zend-diactoros.

```
composer require zendframework/zend-diactoros
```

Create an instance of *ServerRequestInterface* object.

```
$request = Zend\Diactoros\ServerRequestFactory::fromGlobals(
    $_SERVER,
    $_GET,
    $_POST,
    $_COOKIE,
    $_FILES
);
```

and pass `$request` to match method.

```
<?php
/**
 * @var Psr\Http\Message\ServerRequestInterface $request
 */
$route = $matcher->match($request);
?>
```

Dispatching A Route

This is the point at which your application takes over. The route has two properties that you are most likely to be interested in:

- `$route->attributes` is the array of attribute values captured during matching
- `$route->handler` is the handler you added to the route when you mapped it

For example, with the `$route` in hand, you can transfer its attributes to the `$request` ...

```
<?php
foreach ($route->attributes as $key => $val) {
    $request = $request->withAttribute($key, $val);
}
?>
```

... and dispatch to the route handler directly if it was a callable or closure:

```
<?php
$callable = $route->handler;
$response = $callable($request);
?>
```

Alternatively, if you used a class name for the handler, you can create a class from the handler and do what you like with it:

```
<?php
$actionClass = $route->handler;
$action = new $actionClass();
$response = $action($request);
?>
```

Handling Failure To Match

When `$map->match()` returns empty, it means there was no matching route for the request. However, we can still discover the closest-matching, failed route, and which rule it failed to match against.

Your application might do something like the following:

```
<?php
$route = $matcher->match($request);
if (! $route) {
    // get the first of the best-available non-matched routes
    $failedRoute = $matcher->getFailedRoute();

    // which matching rule failed?
    switch ($failedRoute->failedRule) {
        case 'Aura\Router\Rule\Allows':
            // 405 METHOD NOT ALLOWED
            // Send the $failedRoute->allows as 'Allow:'
            break;
        case 'Aura\Router\Rule\Accepts':
            // 406 NOT ACCEPTABLE
            break;
        default:
            // 404 NOT FOUND
            break;
    }
}
?>
```

Working Example

The following is a working example. First, at the command line, require the necessary libraries:

```
$ composer require aura/router zendframework/zend-diactoros
```

Then create the following file as `index.php`:

```
<?php
// set up composer autoloader
require __DIR__ . '/vendor/autoload.php';

// create a server request object
$request = Zend\Diactoros\ServerRequestFactory::fromGlobals(
    $_SERVER,
    $_GET,
    $_POST,
    $_COOKIE,
    $_FILES
);

// create the router container and get the routing map
$routerContainer = new Aura\Router\RouterContainer();
$map = $routerContainer->getMap();

// add a route to the map, and a handler for it
```

```
$map->get('blog.read', '/blog/{id}', function ($request) {
    $id = (int) $request->getAttribute('id');
    $response = new Zend\Diactoros\Response();
    $response->getBody()->write("You asked for blog entry {$id}.");
    return $response;
});

// get the route matcher from the container ...
$matcher = $routerContainer->getMatcher();

// .. and try to match the request to a route.
$route = $matcher->match($request);
if (! $route) {
    echo "No route found for the request.";
    exit;
}

// add route attributes to the request
foreach ($route->attributes as $key => $val) {
    $request = $request->withAttribute($key, $val);
}

// dispatch the request to the route handler.
// (consider using https://github.com/auraphp/Aura.Dispatcher
// in place of the one callable below.)
$callable = $route->handler;
$response = $callable($request, $response);

// emit the response
foreach ($response->getHeaders() as $name => $values) {
    foreach ($values as $value) {
        header(sprintf('%s: %s', $name, $value), false);
    }
}
echo $response->getBody();
```

Now start the built in PHP server ...

```
$ php -S localhost:8000 -t .
```

... and point your browser to `http://localhost:8000/blog/12` .