Build RESTful ZF2 Applications

Matthew Weier O'Phinney
@mwop
http://www.mwop.net/

Who I am

Just this guy:

- Project Lead, Zend Framework
- Open Source enthusiast
- Coffee lover
- Chocolate lover
- Beer lover

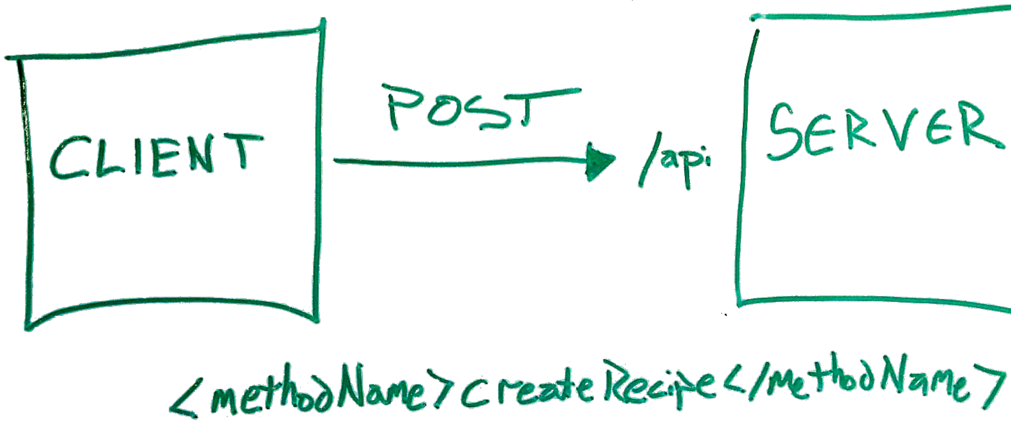
What do I mean by “REST”?

Richardson Maturity Model

http://martinfowler.com/articles/richardsonMaturityModel.html

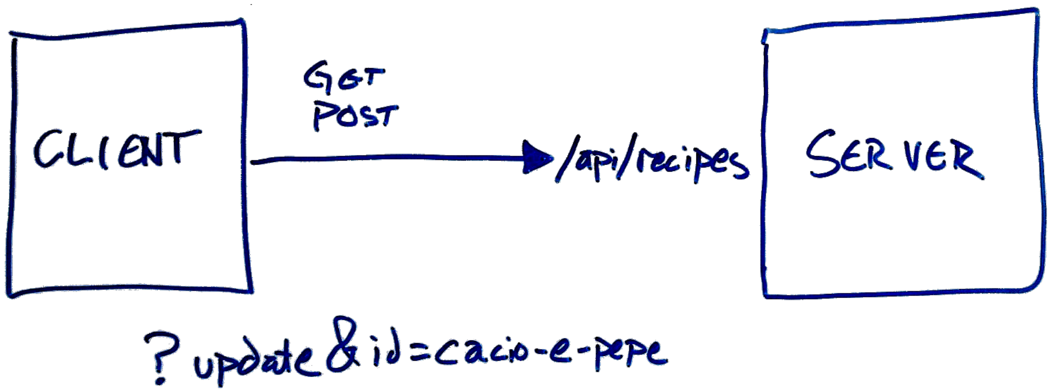
Level 0

HTTP to tunnel RPC



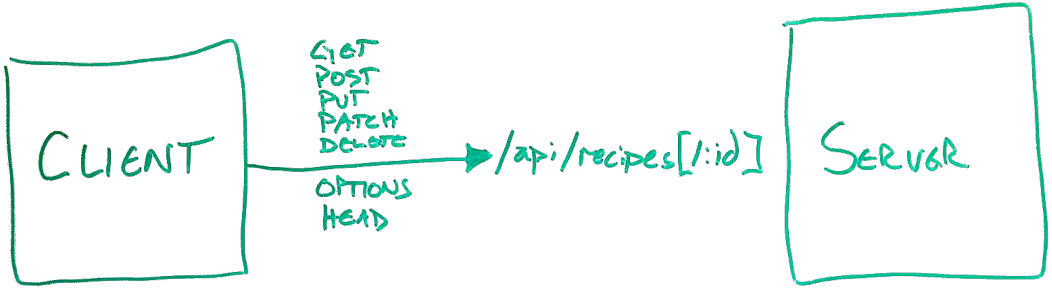
Level 1

Resources (multiple endpoints)



Level 2

HTTP verbs



Level 3

Hypermedia Controls

Level 3: Hypermedia Types: Representations

```
// application/vnd.recipes+json
{
    "id": "identifier",
    "name": "Recipe name",
    "ingredients": [
        // ingredient objects
    ],
    "directions": "Directions for cooking"
}
```

Level 3: Linking

```
{
    "_links": {
        "self": {
            "href": "http://example.com/api/recipes/1234"
        },
        "describedby": {
            "href": "http://example.com/api/resources/recipe"
        }
    }
    // ...
}
```

Level 3: Embedding

```
{
    "_embedded": {
        "addresses": [
            {
                "_links": {"self": {
                    "href": "http://example.com/api/addresses/5678"
                }},
                // a representation
            }
        ]
    }
    // ...
}
```

Aside: Hypermedia Application Language

Which media type should I use?

- Vendor-specific? *(e.g., application/vnd.myorg.recipe+json)*
- Fully generic? *(e.g., application/json)*

A happy medium: HAL

application/hal+json

- Describes hypermedia links
- Describes how to embed resources, either as parts of other resources or parts of collections
- Otherwise retains your object structure

HAL: Resource

```
{
    "_links": {
        "self": {
            "href": "http://example.com/api/recipes/cacio-e-pepe"
        }
    },
    "id": "cacio-e-pepe",
    "name": "Cacio e Pepe Pasta"
}
```

HAL: Embedded resource

```
{
    "_links": {"self": {"href": "..."}},
    "_embedded": {
        "author": {
            "_links": {"self": {"href": "..."}},
            "id": "mario",
            "name": "Mario Mario"
        }
    }
    // ...
}
```

HAL: Collections

```
{
    "_links": {
        "self": {"href": "..."},
        "next": {"href": "..."},
        "prev": {"href": "..."},
        "first": {"href": "..."},
        "last": {"href": "..."}
    },
    // ...
}
```

HAL: Collections

```
{
    // ...
    "_embedded": {
        "recipes": [
            {
                "_links": { "self": { "href": "..." } },
                "id": "cacio-e-pepe",
                "name": "Cacio e Pepe Pasta"
            },
            // ...
        ]
    },
    "and-other-properties": "if desired"
}
```

Translating the Richardson Maturity Model to ZF2

Areas of Concern

- Routing (unique URLs per resource, link generation)
- `AbstractRestfulController` (HTTP method negotiation)
- View Models and Renderers (media-type negotiation and resource representations)

The easy bit: routing

Segment routes

Segment routes with an `:id` segment:

```
'recipe' => array(
    'type' => 'Segment',
    'options' => array(
        'route' => '/api/recipes[/:id]',
        'defaults' => array(
            'controller' => 'Recipe\ApiController',
        ),
    ),
)
```

Controllers

AbstractRestfulController

Extend `Zend\Mvc\Controller\AbstractRestfulController`

- Provides a method per HTTP method, and calls them accordingly.
- Extracts the identifier from the route matches and passes it to the method, when available.
- Marshals data from the request and passes it to the method, when available.

Controller methods

```php
public function create($data); // POST to collection
public function delete($id); // DELETE to resource
public function deleteList(); // DELETE to collection
public function get($id); // GET to resource
public function getList(); // GET to collection
public function head($id = null); // HEAD to either
public function options(); // OPTIONS to either
public function patch($id, $data); // PATCH to resource
public function replaceList($data); // PUT to collection
public function update($id, $data); // PUT to resource
```

Options

- You should tell the consumer what HTTP methods are available for a resource.
- You should *restrict* the consumer to those HTTP methods.
- Use the `options()` method for the first, and write an event listener for the second.

Example: options

```
protected $collectionOptions = array('GET', 'POST');
protected $resourceOptions = array('DELETE', 'GET', 'PATCH', 'PUT');

public function options()
{
    if ($this->params->fromRoute('id', false)) {
        $options = $this->resourceOptions;
    } else {
        $options = $this->collectionOptions;
    }
    $response = $this->getResponse()
    $response->getHeaders()
        ->addHeaderLine('Allow', implode(',', $options));
    return $response;
}
```

Example: listener (1)

```php
public function setEventManager(EventManagerInterface $events)
{
    $this->events = $events;
    // Register a listener at high priority
    $events->attach('dispatch', array($this, 'checkOptions'), 10);
}
```

Example: listener (2)

```php
public function checkOptions($e)
{
    if ($this->params->fromRoute('id', false)) {
        $options = $this->resourceOptions;
    } else {
        $options = $this->collectionOptions;
    }
    if (!in_array($e->getRequest()->getMethod(), $options)) {
        return;
    }
    $response = $this->getResponse()
    $response->setStatusCode(405); // Method Not Allowed
    return $response;
}
```

Example: create

```php
public function create($data)
{
    // if JSON Content-Type, returns decoded data; for
    // application/x-www-form-urlencoded, returns array
    $resource = $this->myComposedService->create($data);
    $response = $this->getResponse();
    $response->setStatusCode(201) // Created
    $response->getHeaders()->addHeaderLine(
        'Location',
        $this->url('recipe', array('id', $resource->id))
    );
    return $resource; // More on this later
}
```

Media-type negotiation

Media-type negotiation

- Choose view model based on `Accept` header. (Potentially write custom view models for custom media types.)
- Potentially restrict access to specific media types.
- Return the appropriate `Content-Type` in the response.

AcceptableViewModelSelector

Select view model type based on Accept header.

```
$criteria = array(
    'Zend\View\Model\JsonModel' => array(
        'application/json',
        'text/json',
    ),
);

$viewModel = $this->acceptableViewModelSelector($criteria);
$viewModel->setVariable('resource', $resource);
```

Use your own view model

```
$criteria = array(
    'Recipe\View\RecipeJsonModel' => array(
        'application/json',
        'text/json',
    ),
);
```

Raise a 406

```
if (!$viewModel instanceof RecipeJsonModel) {
    $response = $this->getResponse();
    $response->setStatusCode(406); // Not Acceptable
    return $response;
}
```

Set the Content-Type

```
// in a controller
$response = $this->getResponse();
$response->getHeaders()
    ->addHeaderLine('Content-Type', 'application/hal+json');
```

Set the Content-Type (2)

```php
// In a "render" listener
function ($e) {
    $viewModel = $e->getViewModel();
    if (!$viewModel instanceof RecipeJsonModel) {
        return;
    }
    $response = $e->getResponse();
    $response->getHeaders()
        ->addHeaderLine('Content-Type', 'application/hal+json');
}
```

Set the Content-Type (3)

```
// In a "response" listener on the View object
function ($e) {
    $viewModel = $e->getModel();
    if (!$viewModel instanceof RecipeJsonModel) {
        return;
    }
    $response = $e->getResponse();
    $response->getHeaders()
        ->addHeaderLine('Content-Type', 'application/hal+json');
}
```

Linking

Helpers and plugins

- `url()` controller plugin and view helper
- `serverUrl()` view helper

Url helper

```
// These examples are true of both controllers
// and view scripts.

// collection:
$this->url('recipe');

// collection with query string:
$this->url('recipe', array(), array('query' => true));

// resource:
$this->url('recipe', array('id' => $id));
```

ServerUrl helper

```
// Generates fully qualified URL (vs. just path)
$this->serverUrl($urlGeneratedViaHelper);
```

Renderers

Resource representations

- The “R” in REST is for “Representational”
- The root of “representational” is “presentation”
- The View layer is where presentation is achieved

Approaches

- Custom View Models
- Custom Renderers

Extending JsonModel

- Provides a `serialize()` method, which is called by the `JsonRenderer`
- Allows you to marshal what you want into the structure you want for the representation

Example: view model

```
class RecipeJsonModel extends JsonModel
{
    public function serialize()
    {
        $resource = $this->getVariable('resource');
        $representation = array(
            'id'   => $resource->getId(),
            'name' => $resource->getName(),
        );
        return Json::encode($representation);
    }
}
```

Custom renderer

- Can provide helper capabilities (e.g., for links!).
- Usually managed by the `ServiceManager`, allowing for dependencies.
- Can alter workflow based on view models detected, or contents of view model.

Example: renderer

```php
class RecipeJsonRenderer implements RendererInterface
{
    public function render($nameOrModel, $values = null)
    {
        if ($nameOrModel instanceof RecipeJsonModel) {
            $helper = $this->helpers->get('RenderRecipe');
        } elseif ($nameOrModel instanceof RecipesJsonModel) {
            $helper = $this->helpers->get('RenderRecipes');
        } else {
            throw new Exception('Cannot handle this!');
        }
        // delegate to the selected helper!
        return $helper($nameOrModel);
    }
}
```

Recap

Understand the Richardson Maturity Model

http://martinfowler.com/articles/richardsonMaturityModel.html

- Level 0: RPC, POX
- Level 1: Resources
- Level 2: HTTP verbs
- Level 3: Hypermedia controls

Get to know emerging REST standards

- Hypermedia Application Language (http://tools.ietf.org/html/draft-kelly-json-hal-05)
- Collection JSON (http://amundsen.com/media-types/collection/)

Understand the HTTP specification

http://www.w3.org/Protocols/rfc2616/rfc2616.html

- HTTP methods, which are idempotent, and expected response structure
- `Accept` and `Content-Type` headers, and how they relate
- HTTP response status codes

Understand ZF2's HTTP capabilities

- `Request`, `Response`, and `Headers` objects from `Zend\Http`
- `AcceptableViewModelSelector` MVC controller helper and the `Accept` HTTP header

Utilize ZF2’s event system

- Use event listeners to check for `Content-Type`, HTTP method used, `Accept` header, etc., and return early for bad requests
- Use event listeners to shape the rendering cycle; use a combination of application and view events

Utilize ZF2’s view layer

- Use custom view models to “type” your responses
- Use custom view renderers to ensure you return appropriate representations
- Use existing helpers such as `url()` and `serverUrl()` to generate links
- Create new helpers for implementing link relations

Topics not covered

- API versioning (hint: use custom media types and/or headers)
- Authentication/Authorization (hint: use OAuth tokens)
- XML and XML formats (hint: PHP has lots of tools for this)
- Probably tons more…

Thank You

@mwop
http://www.mwop.net/
http://framework.zend.com/