GUSTAVO DELGADO R.

BACKEND FOR VUEDIUM

GUSTAVO DELGADO

Software Engineer

Digital Intent

http://gustav.onecore.cl

TOPICS

- Rest API, un commodity
- PHP7 y estándares básicos PSR
- Endpoints de Vuedium
- Where is the Beer?... oh right.. the questions.. anyone?

REST API, UN COMMODITY

Usemos los métodos HTTP!

PETICIONES QUE NO SON RESTFUL

- GET api/getUsers
- GET api/user/10?new_name=jhon
- POST api/user/10/new
- POST api/user/20/delete

RESPUESTAS QUE NO SON CORRECTAS

- GET api/user/10

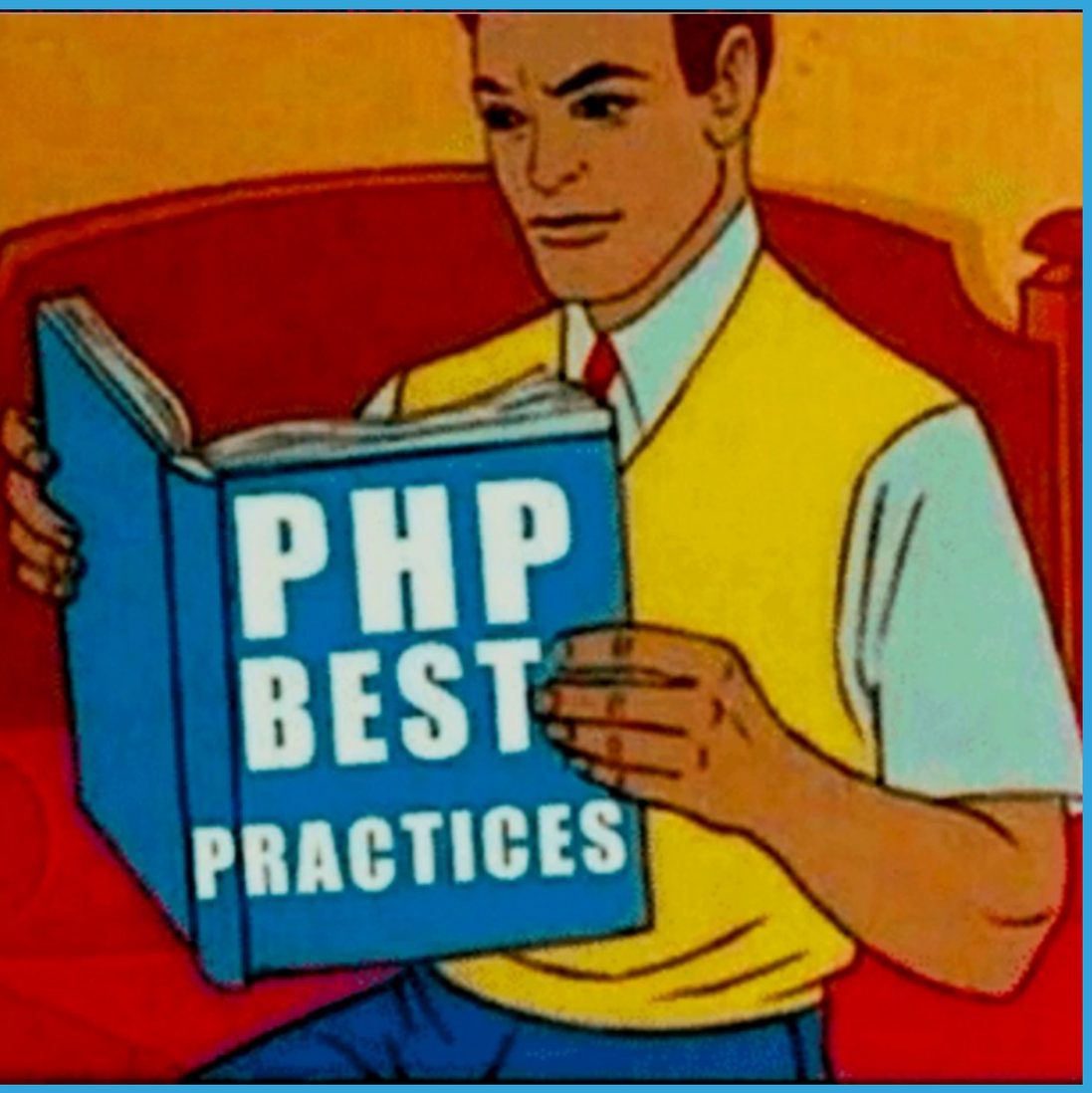
```php
new Response([
    'success' => false,
    'error' => 'The user 10 was not found',
], 200);
```

- GET api/post?tag=nature

```json
{
    "posts":{
        "101": {"title": "Title 101", "description": "Description"},
        "245": {"title": "Title 245", "description": "Description"},
        "33": {"title": "Title 33", "description": "Description"}
    }
}
```

RESTUL API

Description	Endpoint	Response ok	Response nok
Crear un nuevo post	POST api/post	HTTP 200/201	- HTTP 400 Invalid request - HTTP 401 Not authorized - HTTP 403 The account is not allowed to create posts
Editar un post	PATCH api/post/{id}	HTTP 200/202	- HTTP 400 Invalid request - HTTP 401 Not authorized - HTTP 404 Post not found for id {id}
Eliminar un post	DELETE api/post/{id}	HTTP 200/204	- HTTP 401 Not authorized - HTTP 404 Post not found for id {id}
Listar post(s)	GET api/post{/id}	HTTP 200	- HTTP 401 Not authorized



PHP7 Y ESTÁNDARES BÁSICOS PSR

Haters gonna hate

BREVE APOLOGÍA

- OOP
- Mantenible
- Presente en el 80% de toda la web
- Simple

ESTÁNDARES BÁSICOS PSR – HTTP://WWW.PHP-FIG.ORG/PSR/

ACCEPTED

NUM	TITLE	EDITOR
1	Basic Coding Standard	Paul M. Jones
2	Coding Style Guide	Paul M. Jones
3	Logger Interface	Jordi Boggiano
4	Autoloading Standard	Paul M. Jones
6	Caching Interface	Larry Garfield
7	HTTP Message Interface	Matthew Weier O'Phinney
11	Container Interface	Matthieu Napoli, David Négrier
13	Hypermedia Links	Larry Garfield
16	Simple Cache	Paul Dragoonis
15	HTTP Middlewares	Woody Gilk

LOGGER PSR

```php
namespace Psr\Log;
interface LoggerInterface
{
    public function emergency($message, array $context = array());
    public function alert($message, array $context = array());
    public function critical($message, array $context = array());
    public function error($message, array $context = array());
    public function warning($message, array $context = array());
    public function notice($message, array $context = array());
    public function info($message, array $context = array());
    public function debug($message, array $context = array());
    public function log($level, $message, array $context = array());
}
```

```php
class LogLevel
{
    const EMERGENCY = 'emergency';
    const ALERT     = 'alert';
    const CRITICAL  = 'critical';
    const ERROR     = 'error';
    const WARNING   = 'warning';
    const NOTICE    = 'notice';
    const INFO      = 'info';
    const DEBUG     = 'debug';
}
```

HTTP MESSAGE PSR

```php
interface ResponseInterface extends MessageInterface
{
    public function getStatusCode();
    public function withStatus($code, $reasonPhrase = '');
    public function getReasonPhrase();
}
```

```php
interface ServerRequestInterface extends RequestInterface
{
    public function getServerParams();
    public function getCookieParams();
    public function withCookieParams(array $cookies);
    public function getQueryParams();
    public function withQueryParams(array $query);
    public function getUploadedFiles();
    public function withUploadedFiles(array $uploadedFiles);
    public function getParsedBody();
    public function withParsedBody($data);
    public function getAttributes();
    public function getAttribute($name, $default = null);
    public function withAttribute($name, $value);
    public function withoutAttribute($name);
}
```

CONTAINER PSR

```php
interface ContainerInterface
{
    public function get($id);
    public function has($id);
}

interface ContainerExceptionInterface
{
}

interface NotFoundExceptionInterface extends ContainerExceptionInterface
{
}
```

MIDDLEWARE PSR

```php
use Psr\Http\Message\RequestInterface;
use Psr\Http\Message\ResponseInterface;

interface MiddlewareInterface
{
    public function __invoke(
        RequestInterface $request,
        ResponseInterface $response,
        callable $next = null) : ResponseInterface;
}
```

LO BUENO DE USAR UN PSR

- Puedes cambiar una librería por otra fácilmente
- Al seguir un estándar, los issues de implementación son están bien documentados.
- Te obligas a codificar para interfaces en vez de clases concretas.

TEXT

CONSTRUIRÉ MI PROPIO FRAMEWORK CON JUEGOS DE AZAR...

Gustavo Delgado

VUEDIUM ENDPOINTS

LISTAR POSTS – POST CONTROLLER – GET COLLECTION

- GET api/post

```php
/**
 * Get a list of posts
 *
 * @param ServerRequest $request
 * @param Response $response
 * @return Response
 */
public function getCollectionAction(ServerRequest $request, Response $response) : Response
{
    $posts = $this->bus->searchPosts([], ['inflators' => $request->getAttribute('inflators')])['posts'];

    return $this->json([
        'success' => true,
        'data' => [
            'posts' => $posts,
        ],
    ]);
}
```

SEARCH POSTS COMMAND

```php
class SearchPostsCommand implements CommandInterface
{
    public function __construct(array $filters = [], array $options = [])
    {
        $this->filters = $filters;
        $this->options = $options;
    }
}
```

SEARCH POSTS HANDLER

```php
class SearchPostsHandler implements HandlerInterface
{
    public function handle(SearchPostsCommand $command)
    {
        $postQuery = PostQuery::create();
        $filters = $command->getFilters();
        $inflators = isset($command->getOptions()['inflators']) ? $command->getOptions()['inflators'] : [];

        if (isset($filters['slug']))
        {
            $postQuery = $postQuery->filterBySlug($filters['slug']);
        }
        else if (isset($filters['id']))
        {
            $postQuery = $postQuery->filterById($filters['id']);
        }
        else
        {
            if (isset($filters['user_id']))
            {
                $postQuery = $postQuery->filterByUserId($filters['user_id']);
            }
            if (isset($filters['title']))
            {
                $postQuery = $postQuery->filterByTitle($filters['title']);
            }
            if (isset($filters['description']))
            {
                $postQuery = $postQuery->filterByTitle($filters['description']);
            }
        }
        $postQuery = $postQuery->orderByCreatedDt(Criteria::DESC);
        $posts = [];
        foreach ($postQuery->find() as $post)
        {
            $posts[] = $post->map($inflators);
        }
        return [
            'posts' => $posts,
        ];
    }
}
```

DETALLE DE POST – POST CONTROLLER – BEFORE

- GET api/post/{id}

```php
/**
 * {@inheritDoc}
 */
public function before(ServerRequest $request, Response $response) : Response
{
    if ($request->getAttribute('post_id'))
    {
        $posts = $this->bus->searchPosts(
            [$identifier => $postId],
            ['inflators' => $request->getAttribute('inflators')]
        )['posts'];

        if ($posts === [])
        {
            throw new ResourceNotFoundException('Post with ' . $identifier . ' ' . $postId);
        }

        $this->post = $posts[0];
    }

    return parent::before($request, $response);
}
```

DETALLE DE POST – POST CONTROLLER – GET RESOURCE

- GET api/post/{id}

```php
/**
 * Get an specific post by slug
 *
 * @param ServerRequest $request
 * @param Response $response
 * @return Response
 */
public function getResourceAction(ServerRequest $request, Response $response) : Response
{
    return $this->json([
        'success' => true,
        'data' => [
            'post' => $this->post,
        ],
    ]);
}
```

CREAR POST – POST CONTROLLER – POST COLLECTION

- POST api/post

```
{
    "title":"Hello vuedium",
    "description":"This is a description",
    "is_published": true
}
```

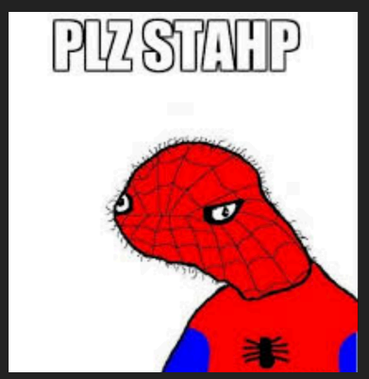
CREATE POST - POST API/POST

```php
/**
 * Create a new post
 *
 * @param ServerRequest $request
 * @param Response $response
 * @return Response
 */
public function postCollectionAction(ServerRequest $request, Response $response) : Response
{
    if ($this->authenticatedUser === [])
    {
        throw new AuthorizedException;
    }

    $sanitizer = $this->getSanitizer([
        'title' => 'string',
        'description' => 'string',
        'is_published' => 'bool',
    ]);

    $input = $sanitizer->sanitize($request->getParsedBody());

    $this->requireFields(['title', 'description']);

    $post = $this->bus->createPost(
        $this->authenticatedUser['id'],
        $input['title'],
        $input['description'],
        (bool)$input['is_published']
    )['post'];

    return $this->json([
        'success' => true,
        'data' => [
            'post' => $post,
        ],
    ]);
}
```

CREATE POST
POST API/POST

```php
class CreatePostHandler implements HandlerInterface
{
    /**…
    public function handle(CreatePostCommand $command)
    {
        $slug = (new Slugify())->slugify($command->getTitle());

        $existingSlug = PostQuery::create()->findOneBySlug($slug);

        if ($existingSlug)
        {
            throw new SlugAlreadyExistsException($command->getTitle());
        }

        $publishedDt = $command->isPublished() ? gmdate('Y-m-d H:i:s') : null;

        $post = new Post();

        $rows = $post->
            setUserId($command->getUserId())->
            setTitle($command->getTitle())->
            setDescription($command->getDescription())->
            setSlug($slug)->
            setPublishedDt($publishedDt)->
            setCreatedDt(gmdate('Y-m-d H:i:s'))->
            save();

        if ($rows <= 0)
        {
            throw new \Exception('not valid data');
        }

        return [
            'post' => $post->map(),
        ];
    }
}
```

PREGUNTAS?

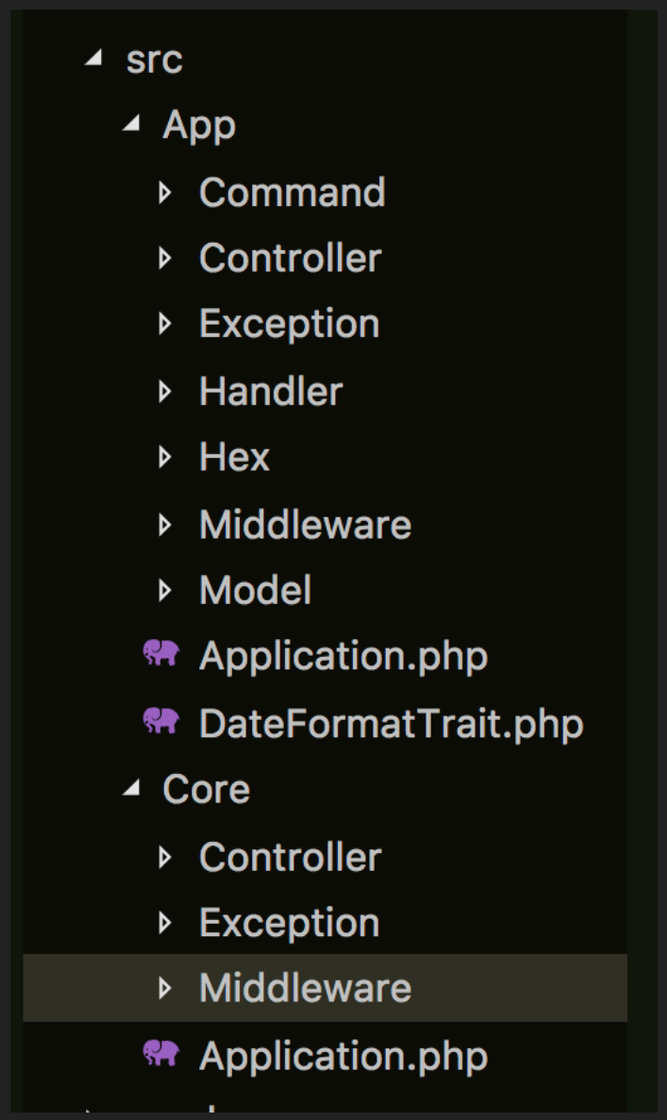


TEXT

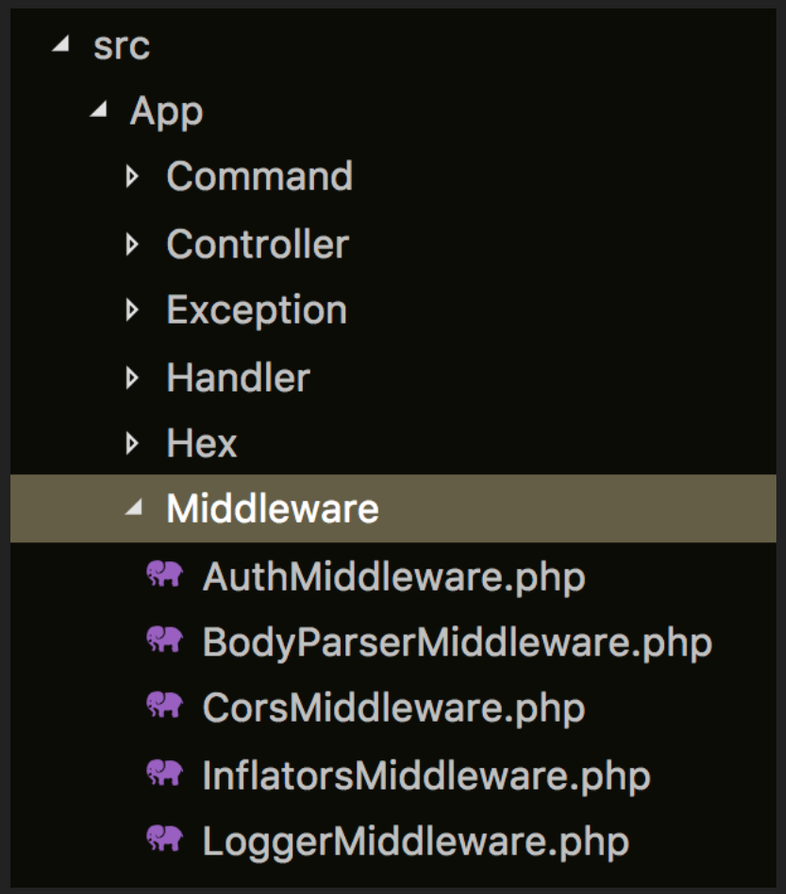
RECURSOS

- Vuedium-api: https://github.com/gustavonecore/vuedium-api
- Vuedium SPA: https://github.com/raulghm/vuedium
- Leftaro framework (alpha): https://github.com/gustavonecore/leftaro
- PHP PSR: http://www.php-fig.org/psr/

ANEXOS – ESTRUCTURA



ANEXOS - MIDDLEWARES




```php
class AuthMiddleware implements MiddlewareInterface
{
    /**…
    public function __invoke(RequestInterface $request, ResponseInterface $response, callable $next = null) :
    {
        $accessToken = null;

        if (isset($request->getQueryParams()['access_token']) === true)
        {
            $accessToken = $request->getQueryParams()['access_token'];
        }
        else if ($request->hasHeader('x-access-token') === true)
        {
            $accessToken = $request->getHeader('x-access-token')[0];
        }

        $request = $request->withAttribute('access_token', $accessToken);

        return $next($request, $response);
    }
}
```

ANEXOS
UPDATE POST HANDLER

```php
class UpdatePostHandler implements HandlerInterface
{
    /**…
    public function handle(UpdatePostCommand $command)
    {
        $slug = (new Slugify())->slugify($command->getTitle());

        $post = PostQuery::create()->findOneById($command->getPostId());

        if (!$post)
        {
            throw new ResourceNotFoundException('Post with id ' . $command->getPostId());
        }

        $existingSlug = PostQuery::create()->findOneBySlug($slug);

        if ($existingSlug && $existingSlug->getId() !== $command->getPostId())
        {
            throw new SlugAlreadyExistsException($command->getTitle());
        }

        if ($command->isPublished() && $post->getPublishedDt() === null)
        {
            $post->setPublishedDt(gmdate('Y-m-d H:i:s'));
        }

        if ($command->isDeleted() && $post->getDeletedDt() === null)
        {
            $post->setDeletedDt(gmdate('Y-m-d H:i:s'));
        }

        $post->setTitle($command->getTitle());
        $post->setSlug($slug);
        $post->setDescription($command->getDescription());
        $post->setUpdatedDt(gmdate('Y-m-d H:i:s'));
        $post->save();

        return [
            'post' => $post->map(),
        ];
    }
}
```