According to the ideal 12 Factor App you want to store your configuration in the specific environment. This includes debugging, logging, and everything else your application does. When building an API, these days you want your response data to be JSON, unless you're an XML holdout which case you're still using XML. Regardless, while in production you want clear and concise error message responses that do not reveal too much information. However, outside of production you probably want a quick way to see at least the source file throwing the exception so you know where to begin without crawling logs for every iterative issue while in development. For this you'll need your API response data to include some contextual debugging and/or exception information you normally wouldn't want in your response. This is a perfect scenario for environment configuration, and Symfony has a great way we can handle this with an `ExceptionSubscriber` for "Dev" and for "Non-Dev" environments (in this example).

Here's a brief breakdown of what we're building:

1. Service declaration for subscribing to exceptions in `dev` and `default` environments
2. `ExceptionSubscriber` class that processes exceptions being thrown, and cleans up the response `JSON` (...and `XML` because we aren't jerks).
3. `DevExceptionSubscriber` that extends `ExceptionSubscriber` and adds some debug "stuff" to the response.

Declare the Default Service

```
//app/config/config.yml
services:
    api.subscriber.exception:
        class:
Litwicki\ApiBundle\EventSubscriber\Exception\ExceptionSubscriber
        tags:
            - { name: kernel.event_subscriber }
        arguments: ["@serializer"]
```

```
//app/config/config_dev.yml
services:
    api.subscriber.exception:
        class:
Litwicki\ApiBundle\EventSubscriber\Exception\Dev\ExceptionSubscriber
        tags:
            - { name: kernel.event_subscriber }
        arguments: ["@serializer"]
```

Build ExceptionSubscriber

```
<?php namespace Litwicki\ApiBundle\EventSubscriber\Exception;

use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpKernel\Event\GetResponseForExceptionEvent;
use JMS\Serializer\Serializer;

class ExceptionSubscriber implements EventSubscriberInterface
{
    protected $serializer;

    public function __construct(Serializer $serializer)
    {
        $this->serializer = $serializer;
    }

    public static function getSubscribedEvents()
    {
        // return the subscribed events, their methods and priorities
        return array(
```

```
            'kernel.exception' => array(
                array('processException', 10)
            )
        );
    }

    /**
     * Process an Exception message.
     *
     * @param
\Symfony\Component\HttpKernel\Event\GetResponseForExceptionEvent
$event
     */
    public function processException(GetResponseForExceptionEvent
$event)
    {
        $exception = $event->getException();
        $code = $exception->getCode() == 0 ?
Response::HTTP_BAD_REQUEST : $exception->getCode();
        $format = preg_match('/\.xml$/',
$event->getRequest()->getUri()) ? 'xml' : 'json';

        $message = $exception->getMessage();

        /**
         * For some particular exceptions, we want to give a generic
message response.
         */

        if($exception instanceof
AuthenticationCredentialsNotFoundException) {
```

```
            $message = 'You must be authorized to access this
resource.';
            $code = Response::HTTP_UNAUTHORIZED;
        }

        if($exception instanceof AccessDeniedHttpException) {
            $message = 'You do not have permission to access this
resource.';
            $code = Response::HTTP_FORBIDDEN;
        }

        if($code === Response::HTTP_NOT_FOUND) {
            $message = 'The resource you were looking for could not be
found.';
        }

        if($code === Response::HTTP_INTERNAL_SERVER_ERROR) {
            $message = 'There was an error completing your request.';
        }

        $data = [
            'code' => $code,
            'message' => $exception->getMessage()
        ];

        $content = $this->serializer->serialize($data, $format);

        $response = new Response();
        $response->headers->set('Content-Type',
sprintf('application/%s', $format));
        $response->setStatusCode($code);
```

```
        $response->setContent($content);
        $event->setResponse($response);

    }
}
```

Extend ExceptionSubscriber for DevExceptionSubscriber

Here, we don't filter any of the message content, so we can get the literal exception text thrown for exception classes we didn't specifically write. This is particularly useful if we want to mask exceptions for invalid passwords so users cannot guess which usernames exist or not, and other business rule exceptions for Unauthorized or Access Denied messages where we don't want to reveal too much.

```
<?php namespace Litwicki\ApiBundle\EventSubscriber\Exception\Dev;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpKernel\Event\GetResponseForExceptionEvent;
use Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException;
use
Symfony\Component\Security\Core\Exception\AuthenticationCredentialsNot
FoundException;

class ExceptionSubscriber extends
\Litwicki\ApiBundle\EventSubscriber\Exception\ExceptionSubscriber
{
    /**
     * Process an Exception message.
     *
     * @param
```

```
\Symfony\Component\HttpKernel\Event\GetResponseForExceptionEvent
$event
     */
    public function processException(GetResponseForExceptionEvent
$event)
    {
        $exception = $event->getException();
        $code = $exception->getCode() == 0 ?
Response::HTTP_BAD_REQUEST : $exception->getCode();
        $format = preg_match('/\.xml$/',
$event->getRequest()->getUri()) ? 'xml' : 'json';

        $message = $exception->getMessage();

        $data = [
            'code' => $code,
            'message' => $exception->getMessage(),
            'error' => $exception->getLine(),
            'debug' => $exception->getTraceAsString(),
        ];

        $content = $this->serializer->serialize($data, $format);

        $response = new Response();
        $response->headers->set('Content-Type',
sprintf('application/%s', $format));
        $response->setStatusCode($code);
        $response->setContent($content);
        $event->setResponse($response);

    }
```

```
}
```

Finally..

In use, you'll know see responses like this:

```
{
    "code": 401,
    "message": "You must be authorized to access this resource."
    "line": "Litwicki\ApiBundle\Controller\UserController.php:101",
    "trace": "A really long JSON blog of the exception trace.."
}

{
    "code": 401,
    "message": "You must be authorized to access this resource."
}
```

Share this:

- LinkedIn
- Twitter
- Email
- Print