Documenting APIs with Swagger™

...

or "*how to be lazy and create solid documentation for your Symfony3 API so you don't waste time communicating with your frontend or mobile developers*"

About me

Ante Crnogorac (ante@q-software.com)

- PHP-ing since 5.3 (about 8 years)
- Symfony fan last 3.5 years (wish it was more!)
- Twitter: **@zwer82**
- Github: **github.com/acrnogor**
- Job: Senior Dev & Team Lead at Q Software

So what’s the idea here?

- Document your API - nobody wants to use undocumented software
- Test your API using this documentation
- Be lazy, lazy is good! Also DRY!
- Avoid "omg we just replaced our mobile/angular developer, can you brief/teach the new guy?"
- Use standards, don’t make stuff up

WHAT’S SWAGGER, OpenAPI AND ALL THAT STUFF?

OpenAPI

- API description format for REST APIs
- Describes endpoints, operations, parameters and other stuff regarding your API
- Uses JSON or YAML

WHAT’S SWAGGER, OpenAPI AND ALL THAT STUFF?

SWAGGER

- set of open source tools for OpenAPI that help you build documentation
- Swagger Editor - write the specs - **NEED THIS**
- Swagger UI - render a nice interactive API documentation - **TO GET THIS**
- Swagger Codegen - generate basic CRUD code

What next - let’s say I know API-fu, where to begin?

How do I “Swagger” it?

- We’ll assume the following:
  - You’ve used Symfony 2.7+ or 3 before (if not, it should be a nice jump-start for any other framework)
  - Your API is already working, but not documented in any way
  - You’ve followed at least some of REST best practices
  - Basic knowledge of annotations

Let’s start!

1. Composer

- like always with **Symfony** - we'll need some bundles, get them on Packagist.org

```
$ composer require timeinc/swagger-bundle --dev

$ composer require harmbandstra/swagger-ui-bundle --dev
```

- Hopefully, with Symfony FLEX, you’ll just be like "*composer req swagger*" - but that’s still in beta, still a few months away
    - even in beta it works great - but no Swagger recipe yet!

2. Add the bundles to your Kernel

AppKernel.php

- Add the bundles to your registerBundles() method:

```
if (in_array($this->getEnvironment(), ['dev', 'test'], true)) {
    $bundles[] = new Symfony\Bundle\DebugBundle\DebugBundle();
    $bundles[] = new Symfony\Bundle\WebProfilerBundle\WebProfilerBundle();
    $bundles[] = new Sensio\Bundle\DistributionBundle\SensioDistributionBundle();
    $bundles[] = new Sensio\Bundle\GeneratorBundle\SensioGeneratorBundle();
    $bundles[] = new TimeInc\SwaggerBundle\SwaggerBundle();
    $bundles[] = new HarmBandstra\SwaggerUiBundle\HBSwaggerUiBundle();
}
```

3. Configure the bundles + add required routes

for timeinc/swagger-bundle:

```
swagger:
    version: '2.0'
    info:
        title: 'Q Example Swagger API'
        version: '1.0.0'
        description: 'Maximum Fluffer.'
    pretty_json: true
    host: ~
    base_path: '/api'
    schemes:
        - http
    produces:
        - application/json
    consumes:
        - application/json
    annotations:
        bundles:
            - QExampleBundle
```

for harmbandstra/swagger-ui-bundle:

```
hb_swagger_ui:
  files:
    - "/_swagger/swagger.json"
```

routing_dev.yml

```
# FOR SWAGGER JSON FILES
_swagger:
    resource:
"@SwaggerBundle/Resources/config/routing.xml"
    prefix:   /

# SWAGGER UI ROUTE
_swagger-ui:
    resource:
'@HBSwaggerUiBundle/Resources/config/routing.yml'
    prefix:   /docs
```

Config done - let’s write some documentation already!

- Demoing with our own ExampleBundle app - you can get it on Github:

http://github.com/acrnogor/q-alliance-example-bundle

- There are two models - **Author** and **Book** - that we want to document:
  - **Author** (can write multiple Books)
  - **Book** (is written by a single Author)
- All files for documentation will be in our bundle, in Swagger/models directory

Create an Author - we'll need a request and response!

`[POST] /api/authors`

- Request JSON
  - Has no id
  - All other fields are posted
- Response JSON
  - Has id
  - All other fields are returned

Let's show some Swagger Annotations. Think of them as bunch of files that are building blocks for your requests and responses.

Create an Author - Request

FILE: Swagger/models/Author/Author-request-create.php

```
/**
 * @SWG\Definition(
 *     definition="Example-Author-request-create",
 *     allOf={
 *         @SWG\Schema(ref="#/definitions/Example-Author-base"),
 *         @SWG\Schema(
 *             title="Example Author request create",
 *             required={"first_name", "lastName", "gender", "place_of_birth"},
 *             @SWG\Property(
 *                 property="biography",
 *                 type="string",
 *                 example="He was OK."
 *             ),
 *         ),
 *     }
 * )
 */
```

Create an Author - Request - base

FILE: Swagger/models/Author/Author-base.php

```
/**
 * @SWG\Definition(
 *     definition="Example-Author-base",
 *     title="Example Author base",
 *     @SWG\Property(
 *         property="first_name",
 *         type="string",
 *         example="John",
 *     ),
 *     @SWG\Property(
 *         property="last_name",
 *         type="string",
 *         example="Doe",
 *     ),
 *     @SWG\Property(
 *         property="birthday",
 *         type="string",
 *         format="date",
 *         example="1980-08-28",
 *     ),
 *     @SWG\Property(
 *         property="gender",
 *         type="string",
 *         example="male",
 *         enum={"male","female"},
 *     ),
 *     @SWG\Property(
 *         property="place_of_birth",
 *         type="string",
 *         example="Zagreb, Croatia",
 *     ),
 * )
 */
```

So why this base?

- Common across most requests and responses
- Reuse it and merge with other properties
- Use serialization groups
- Since biography is usually some text/longtext, serialize it and show only on GET route for **single** item
  - See: *Author-response-light.php*

Create an Author - Response

FILE: Swagger/models/Author/Author-response-default.php

```
/**
 * @SWG\Definition(
 *     definition="Example-Author-response-default",
 *     allOf={
 *         @SWG\Schema(ref="#/definitions/Helper-Id"),
 *         @SWG\Schema(ref="#/definitions/Example-Author-base"),
 *         @SWG\Schema(
 *             title="Example Author response default",
 *             @SWG\Property(
 *                 property="biography",
 *                 type="string",
 *                 example="He was OK."
 *             ),
 *             @SWG\Property(
 *                 property="name",
 *                 type="string",
 *                 example="John Doe",
 *                 description="Virtual property that combines first name and last name."
 *             ),
 *         ),
 *     }
 * )
 */
```

Create an Author - Response - Helper ID?

`FILE: Swagger/models/Helper-Id.php`

```
/**
 * @SWG\Definition(
 *     definition="Helper-Id",
 *     title="Helper ID",
 *     @SWG\Property(
 *         property="id",
 *         type="integer",
 *         format="int64",
 *         example=1,
 *     ),
 * )
 */
```

Helper ID:

- You probably need id/uuid?
- But you will not need it for create request :-)
- So might aswell be flexible and extract this to a separate file

Create an Author - what about Controller annotations?

FILE: Controller/AuthorController.php

```
/**
 * Create a new Author.
 *
 * @SWG\Post(
 *     path="/authors",
 *     tags={"Example Author"},
 *     summary="Create a new Author",
 *     @SWG\Parameter(
 *         name="body",
 *         in="body",
 *         required=true,
 *         @SWG\Schema(ref="#/definitions/Example-Author-request-create")
 *     ),
 *     @SWG\Response(
 *         response="200",
 *         description="Author created",
 *         @SWG\Schema(ref="#/definitions/Example-Author-response-default")
 *     ),
 *     @SWG\Response(
 *         response="422",
 *         description="Validation error",
 *         @SWG\Schema(ref="#/definitions/Validation-error")
 *     )
 * )
 */
```

- Add them above your createAction function
- Mostly self-explanatory
- But hey, what's this Validation-error?!
- Project specific - we wanted nice human readable errors instead of Doctrine exploding on insert :-)
- Feel free to explore this in our example project!

Create an Author - Response - Validation-error

FILE: Swagger/models/Validation-error.php

```
/**
 * @SWG\Definition(
 *     definition="Validation-error",
 *     @SWG\Property(
 *         property="code",
 *         type="integer",
 *         example=422
 *     ),
 *     @SWG\Property(
 *         property="message",
 *         type="string",
 *         example="Unprocessable Entity"
 *     ),
 *     @SWG\Property(
 *         property="validation",
 *         @SWG\Property(
 *             additionalProperties=true,
 *             property="field_name",
 *             type="string",
 *             example="Please fill FieldName"
 *         )
 *     )
 * )
 */
```

- We added some Asserts to models
- If some assert fails - validation fails - results in 422 Unprocessable entity
- Shows nice JSON response like this:

```
{
    "message": "Validation failed",
    "code": 422,
    "validation": {
        "gender": "Please enter a valid gender"
    },
}
```

Index Authors - Response

FILE: Controller/AuthorController::indexAction

```
/**
 * List all Authors.
 *
 * @SWG\Get(
 *     path="/authors",
 *     tags={"Example Author"},
 *     summary="List all authors",
 *     @SWG\Response(
 *         response="200",
 *         description="List of authors",
 *         @SWG\Schema(
 *             type=" array",
 *
@SWG\Items(ref="#/definitions/ Example-Author-response-light "),
 *         ),
 *     ),
 * )
 *
 */
```

- Listing is just a JSON array of 'Example-Author-response-light'
- Use "light" payload here - we use serialization groups to skip "biography"- keep it small on listings
- Why no error? Only thing that can go wrong is 5xx, we didn't document it.
- There's no 404 Not found here - if there's no Authors, response will be an empty JSON array

Get a single Author - Response

FILE: Controller/AuthorController::getAction

```
/**
 * @SWG\Get(
 *     path="/authors/{id}",
 *     tags={"Example Author"},
 *     summary="Get a single Author",
 *     @SWG\Parameter(
 *         name="id",
 *         in="path",
 *         required=true,
 *         description="ID",
 *         type="integer",
 *         format="int64"
 *     ),
 *     @SWG\Response(
 *         response="200",
 *         description="Author found",
 *
@SWG\Schema(ref="#/definitions/Example-Author-response-default-books")
 *     ),
 *     @SWG\Response(
 *         response="404",
 *         description="Not found"
 *     ),
 * )
 */
```

FILE: Swagger/models/Author/Author-response-default-books.php

```
/**
 * @SWG\Definition(
 *     definition="Example-Author-response-default-books",
 *     allOf={
 *         @SWG\Schema(ref="#/definitions/Example-Author-response-default"),
 *         @SWG\Schema(
 *             title="Example Author response default-books",
 *             type="object",
 *             @SWG\Property(
 *                 property="books",
 *                 type="array",
 *                 @SWG\Items(
 *                     ref="#/definitions/Example-Book-response-default",
 *                 ),
 *             ),
 *         ),
 *     }
 * )
 */
```

- Adding books to serialized Author as a bonus!
- You can do it in a separate route if you like

Cheat sheet: Workflow

1. Model the REQUEST (if needed)
2. Model the RESPONSE
3. Model the Controller action:
   - Is it PUT, POST, GET or DELETE? (or something else?)
   - HTTP codes for each possible response
   - Response body for each possible response
   - Are there required parameters? Add them as well!
     i. Auth token?
     ii. Entity ID?
     iii. Search params?

We use this as a checklist so we don't miss anything when creating new or updating existing API endpoints.

Conclusion

1. Swagger your APIs
2. …
3. Profit :-)

Any questions?