

# Struktura pliku yaml

Swagger jest narzędziem do projektowania, prototypowania i dokumentowania API w języku yaml lub json. Swagger w projekcie jest naszym wyznacznikiem do budowy frontendu, backendu, oraz tworzenia bazy danych i dzięki takiemu podejściu co do budowy aplikacji może się wypowiedzieć każdy z członków projektu przed jego rozpoczęciem.

Na poprzednich zajęciach było za zadanie przejść przez proces instalacyjny oraz zapoznać się z działaniem swagger-editora i składni yaml. Dzisiaj trochę więcej o swaggerze i dokładniejszym opisem działania.

```
1  swagger: "2.0"
2  info:
3    description: "This is a sample server Petstore server. You can find out more about Swagger at [http://swagger.io](http://swagger.io) or on [irc.freenode.net, #swagger](http://swagger.io/irc/). For this sample, you can use the api key `special-key` to test the authorization filters."
4    version: "1.0.0"
5    title: "Swagger Petstore"
6    termsOfService: "http://swagger.io/terms/"
7    contact:
8      email: "apiteam@swagger.io"
9    license:
10     name: "Apache 2.0"
11     url: "http://www.apache.org/licenses/LICENSE-2.0.html"
12  host: "petstore.swagger.io"
13  basePath: "/v2"
14  tags:
15  - name: "pet"
16    description: "Everything about your Pets"
17    externalDocs:
18      description: "Find out more"
19      url: "http://swagger.io"
20  - name: "store"
21    description: "Access to Petstore orders"
22  - name: "user"
23    description: "Operations about user"
24    externalDocs:
25      description: "Find out more about our store"
26      url: "http://swagger.io"
27  schemes:
28  - "https"
29  - "http"
```

Bazując na przykładzie ze swagger-editora w linijce 1 znajduje się wersja edytora. W drugiej mamy info, gdzie znajdują się kolejno opis projektu, wersja, tytuł, regulamin, adres kontaktowy, licencja. W specyfikacji możemy znaleźć więcej opcji do ustawienia, dla przykładu pole kontakt może mieć również inne pola:

Field Name	Type	Description
name	string	The identifying name of the contact person/organization.
url	string	The URL pointing to the contact information. MUST be in the format of a URL.
email	string	The email address of the contact person/organization. MUST be in the format of an email address.

Link: <https://swagger.io/specification/#contact-object>

Warto zauważyć że prawa strona automatycznie reaguje na wszystkie zmiany i opis z yamla został przedstawiony w eleganckiej formie na górze strony.

## Swagger Petstore 1.0.0

[ Base URL: [petstore.swagger.io/v2](http://petstore.swagger.io/v2) ]

This is a sample server Petstore server. You can find out more about Swagger at <http://swagger.io> or on [irc.freenode.net](http://irc.freenode.net), #swagger. For this sample, you can use the api key `special-key` to test the authorization filters.

[Terms of service](#)

[Contact the developer](#)

[Apache 2.0](#)

[Find out more about Swagger](#)

Kolejnym elementem który przyda się nam są opisy operacji. Możemy wykorzystać do tego tagi. Zaraz po ustawieniach projektu można zdefiniować tagi które będą używane w dalszej części.

```
14 tags:
15   - name: "pet"
16     description: "Everything about your Pets"
17   - externalDocs:
18       description: "Find out more"
19       url: "http://swagger.io"
20   - name: "store"
21     description: "Access to Petstore orders"
22   - name: "user"
23     description: "Operations about user"
24   - externalDocs:
25       description: "Find out more about our store"
26       url: "http://swagger.io"
27 schemes:
28   - "https"
29   - "http"
30 paths:
```

Link: <https://swagger.io/docs/specification/2-0/grouping-operations-with-tags/>

Kolejnym ważnym elementem struktury API są ścieżki:

```
30 paths:
31   /pet:
32     post:
33       tags:
34         - "pet"
35       summary: "Add a new pet to the store"
36       description: ""
37       operationId: "addPet"
38       consumes:
39         - "application/json"
40         - "application/xml"
41       produces:
42         - "application/xml"
43         - "application/json"
44       parameters:
45         - in: "body"
46           name: "body"
47           description: "Pet object that needs to be added to the store"
48           required: true
49           schema:
50             $ref: "#/definitions/Pet"
51       responses:
52         "405":
53           description: "Invalid input"
54       security:
55         - petstore_auth:
56           - "write:pets"
57           - "read:pets"
```

Pierwszą metodą CRUD w przykładzie jest Post służący do dodawania nowych elementów do bazy. Inne dostępne metody to

GET – pobieranie pozycji z bazy

PUT – modyfikacja/aktualizacja

DELETE – usuwanie danych

Oczywiście często metoda PUT czy GET posiada id.

W linii 33 przypisujemy tag który czytelnie pokaże nasze metody z opisem po prawej stronie oraz kolejne pole summary opisze daną metodę. Następnie możemy ustawić opis metody. Kolejne pola consumes i produces są to typy mime: <https://swagger.io/docs/specification/2-0/mime-types/>.

Kolejnym ważnym elementem jest określenie parametrów, tutaj możemy zdefiniować typ prosty, albo skorzystać ze schematu poprzez \$ref: ścieżka do definicji. Tutaj możemy również dodać przykłady <https://swagger.io/docs/specification/describing-responses/>. Wszystkie schematy należy umieszczać na dole pliku yaml. W linii 51 możemy dodać odpowiedzi http. Najbardziej popularne to: 200 – wszystko ok, 404 – nie znaleziono strony, 503 – usługa niedostępna, 500 – wewnętrzny błąd serwera. Od linii 55 można definiować ustawienia bezpieczeństwa, w tym wypadku dostępny jest odczyt i zapis.

Definicje schematów:

```
579 definitions:
580   Order:
581     type: "object"
582     properties:
583       id:
584         type: "integer"
585         format: "int64"
586       petId:
587         type: "integer"
588         format: "int64"
589       quantity:
590         type: "integer"
591         format: "int32"
592       shipDate:
593         type: "string"
594         format: "date-time"
595       status:
596         type: "string"
597         description: "Order Status"
598         enum:
599           - "placed"
600           - "approved"
601           - "delivered"
602       complete:
603         type: "boolean"
604         default: false
605     xml:
606       name: "Order"
607   Category:
608     type: "object"
609     properties:
```

### Models

**Order** ▾ {  
 id integer(\$int64)  
 petId integer(\$int64)  
 quantity integer(\$int32)  
 shipDate string(\$date-time)  
 status string  
  
 Order Status  
 Enum:  
 ▾ [ placed, approved, delivered ]  
 complete boolean  
 default: false  
}

**Category** >

**User** >

**Tag** >

W przypadku przykładu PetStore definicje zaczynają się od 579 linijki. Obiekt Order zawiera pola potrzebne do skonstruowania poprawnego schematu czyli id własne, id zwierzęcia i inne potrzebne w zamówieniu pola. Taka struktura będzie wzorem do stworzenia bazy danych.

Dzięki swagger-editor możemy później wygenerować szkielet wraz z kontrolerami.

**Swagger Editor** File Edit Generate Server Generate Client  
1 swagger: "2.0"  
2 info:  
3 description: "This is a sample s  
4 title: "Swagger Petstore"  
5 termsOfService: "http://swagger.  
6 contact:  
7 email: "apiteam@swagger.io"  
8 license:  
9 name: "Apache 2.0"  
10 url: "http://www.apache.org/li  
11 host: "petstore.swagger.io"  
12 basePath: "/v2"  
13 tags:  
14 - name: "pet"  
15 description: "Everything about y  
16 externalDocs:  
17 description: "Find out more"  
18 url: "http://swagger.io"  
19 - name: "store"  
20 description: "Access to Petstore  
21 name: "user"  
22 description: "Operations about u  
23 externalDocs:  
24 description: "Find out more ab  
25 url: "http://swagger.io"  
26 schemes:

ada-server	jaxrs-resteasy	restbed
aspnetcore	jaxrs-resteasy-eap	rust-server
erlang-server	jaxrs-spec	scala-lagom-server
finch	kotlin-server	scalatra
go-server	lumen	sinatra
haskell	msf4j	slim
inflector	nancyfx	spring
java-pkms	<b>nodejs-server</b>	undertow
java-play-framework	php-silex	ze-ph
java-vertx	php-symfony	
jaxrs	piastache-server	
jaxrs-cxf	python-flask	
jaxrs-cxf-cdi	rails5	

W utworzonym w ten sposób szkielecie musimy doinstalować paczki za pomocą menagera npm, czyli komendą npm install, a następnie npm start. W tak utworzonym szkielecie mamy dostęp do wygenerowanych automatycznie kontrolerów w folderze „controllers” oraz mamy dostęp do pliku yaml w folderze api. Wygodnie jest na tym etapie korzystać ze Swagger-editora ponieważ interfejs od razu waliduje i generuje interfejs z prawej strony. Szkielet aplikacji przyda się później.