Spis treści

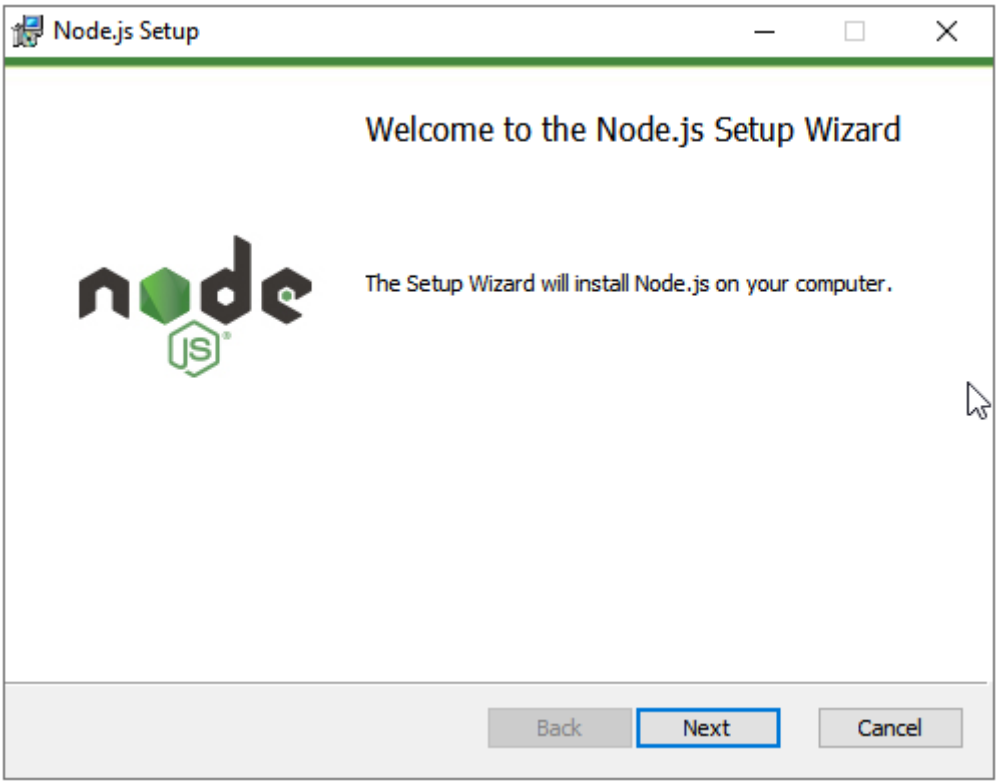
1. Przygotowanie Node.js

1.1. Instalacja Node.js z uprawnieniami administratora

Instalacja Node.js rozpoczyna się od pobeania wersji LTS instalatora Node.js ze strony https://nodejs.org/en/download/.

Następnym krokiem będzie uruchomienie go i przejście standardowej procedury instalacyjnej wybierając domyślne ustawienia.

Konieczne będzie wyrażenie zgody na licencję i wprowadzenie zmian na urządzeniu.



Ilustracja 1. Instalator Node.js.

1.2. Instalacja Node.js bez uprawnień administratora

Node.js może być też zainstalowany bez uprawnień administratora. W tym wypadku konieczne jest pobranie wersji *binary* dla posiadanego systemu ze strony https://nodejs.org/en/download/.

Po rozpakowaniu do folderu można przystąpić do używania go.

Ze względu na brak możliwości ustawienia w sposób permanentny ścieżki do folderu bez uprawnień administratora konieczne będzie przed każdym użyciem Node.js i Npm skonfigurowanie ścieżki na poziomie powłoki. Znacznie ułatwi to wszelkie operacje. W tym celu należy z poziomu powłoki, z której korzystamy skonfigurować zmienną **PATH** wskazując na lokację pliku *node.exe* np.:

```
PATH=%PATH%;C:\node\node-v12.19.0-win-x64
```

1.3. Sprawdzenie instalacji Node.js

Po instalacji możemy przetestować ją. Zaczynamy od stworzenia katalogu projektu testowego (np.: test). Po przejściu do niego wywołujemy polecenia:

```
node -v
npm -v
```

Operacja ta powinna wyświetlić nam informację o wersji Node.js i Npm, co przykładowo, może wyglądać jak następuje:

```
λ node -v
v12.19.0
λ npm -v
6.14.8
```

W ulubionym edytorze można teraz stworzyć w katalogu projektu krótki program:

```
console.log("Hello Node!")
```

Uruchomienie go z użyciem polecenia *node* to przekazanie jego nazwy jako parametru. W wyniku wywołania wyświetlony zostanie komunikat:

```
λ node hello.js
Hello Node!
```

Idąc krok dalej możemy uruchomić obsługę serwera web. Zainstalować możemy w tym celu pakiet *Express.js* wydając polecenie *npm install express*:

```
λ npm install express
--8<--
+ express@4.17.1
updated 1 package and audited 548 packages in 41.229s
--8<--
```

Stwórzona kolejną aplikację testowa w pliku *express.js* zawiera:

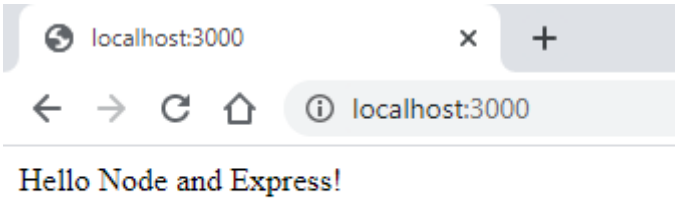
```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => res.send('Hello Node and Express!'))
app.listen(port, () => console.log(`Aplikacja uruchomiona na porcie ${port}!`))
```

Uruchomiamy ją analogicznie jak poprzednio. W trakcie tej operacji może pojawić się pytanie o zezwolenie na dostęp do połączeń. Pracując z lokalnej maszyny nie ma znaczenia, która opcja zostanie wybrana.

W wierszu poleceń poleceń pojawi się po uruchomieniu komunikat:

```
λ node express.js
Aplikacja uruchomiona na porcie 3000!
```

Po uruchomieniu przeglądarki i wpisaniu w niej adresu http://localhost:3000, gdzie 3000 to wskazany wcześniej numer portu powinien być widoczny wynik działania aplikacji webowej:



Ilustracja 2. Aplikacja Node.js korzystająca z Express.

Jeżeli wszystkie korki zakończyły się sukcesem oznacza to, że środowisko jest gotowe do pracy.

2. Użycie Swaggera do przygotowania Restful API w Node.js

2.1. Instalacja Swaggera

W celu instalacji edytora Swagger konieczne jest pobranie go z repozytorium https://github.com/swagger-api/swagger-editor/releases w najnowszej wersji jako pliku *swagger-editor* w postaci archiwum *.zip* lub *.tar.gz*. Archiwum należy wypakować do wybranego folderu.

Po przejsciu do głównego folderu *swagger-editor* rozpoczynamy proces instalacji i wstępnej konfiguraci z wykorzystaniem komendy *npm install*.

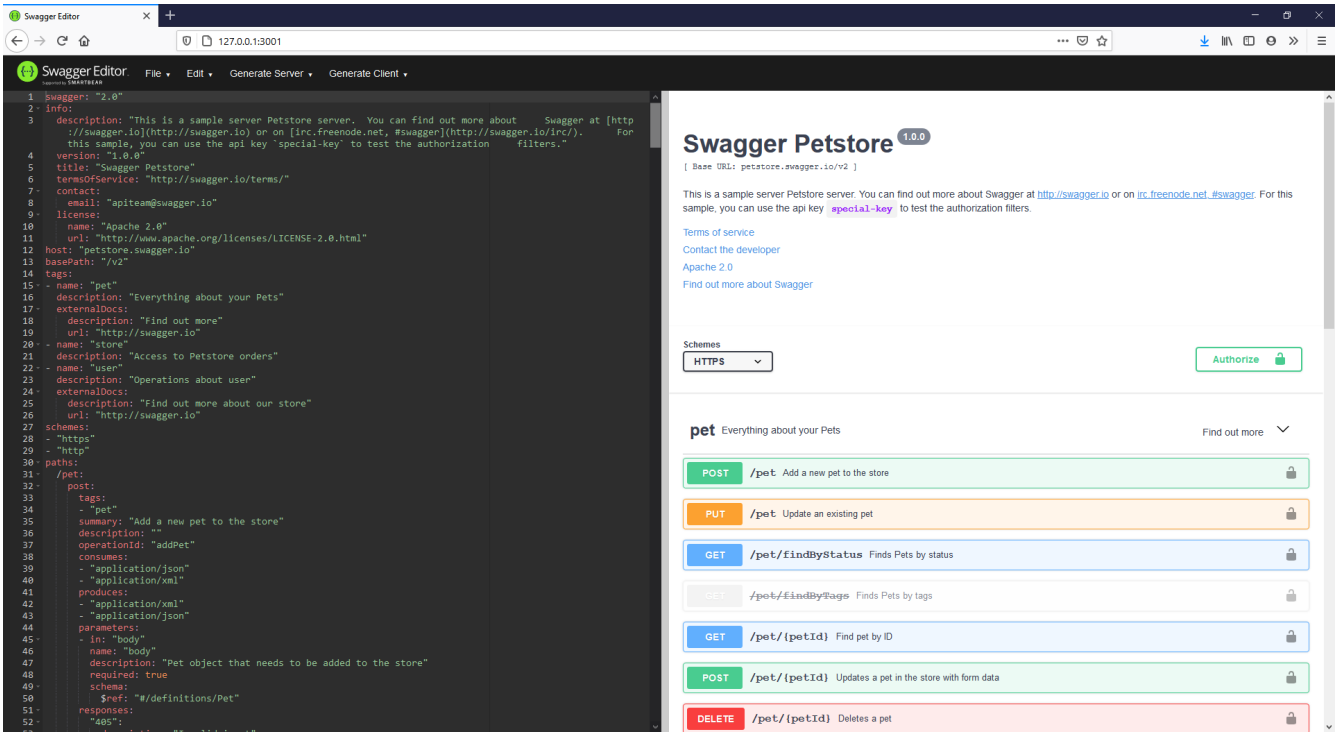
Możliwe jest też uruchomienie bez instalacji poprzez otworzenie w przeglądarce pliku *index.html* jednak wiąże się to z pewnymi ograniczeniami funkcjonalności.

```
λ npm install
--8<--
added 3433 packages from 1391 contributors and audited 3446 packages in 1455.51s
--8<--
```

Uruchomienie edytora to wydanie polecenie *npm run* i otwierzenie w przeglądarce wskazanego adresu:

```
λ npm start
...
Starting up http-server, serving ./
Available on:
  http://192.168.0.189:3001
  http://127.0.0.1:3001
Hit CTRL-C to stop the server
```

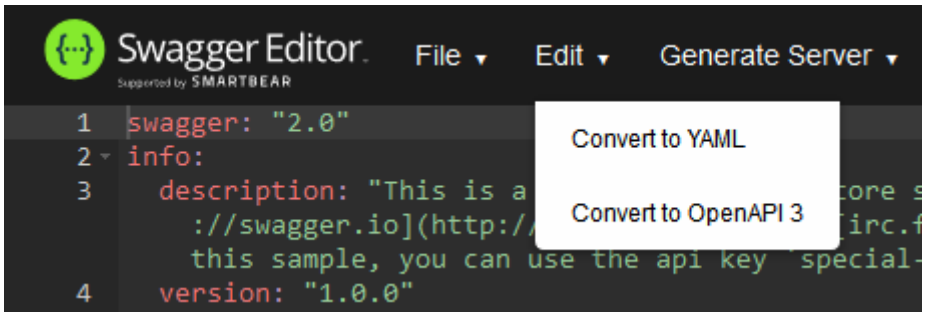
Edytor przy pierwszym uruchomieniu wczyta przykładową definicję API i wyświetli ją.



Ilustracja 3. Uruchomiony edytor Swagger.

API będzie w postaci podstawowej. Możliwe jest dokonanie jego konwersji do najbardziej bieżącego standardu OpenAPI 3 wybierając z menu **Edit** > **Convert to OpenAPI 3**.

Na ten moment wersja Swaggera 3.0.1 generuje niepoprawny projekt dla Node.js więc najlepiej pozostać przy wariancie niekonwertowanym.

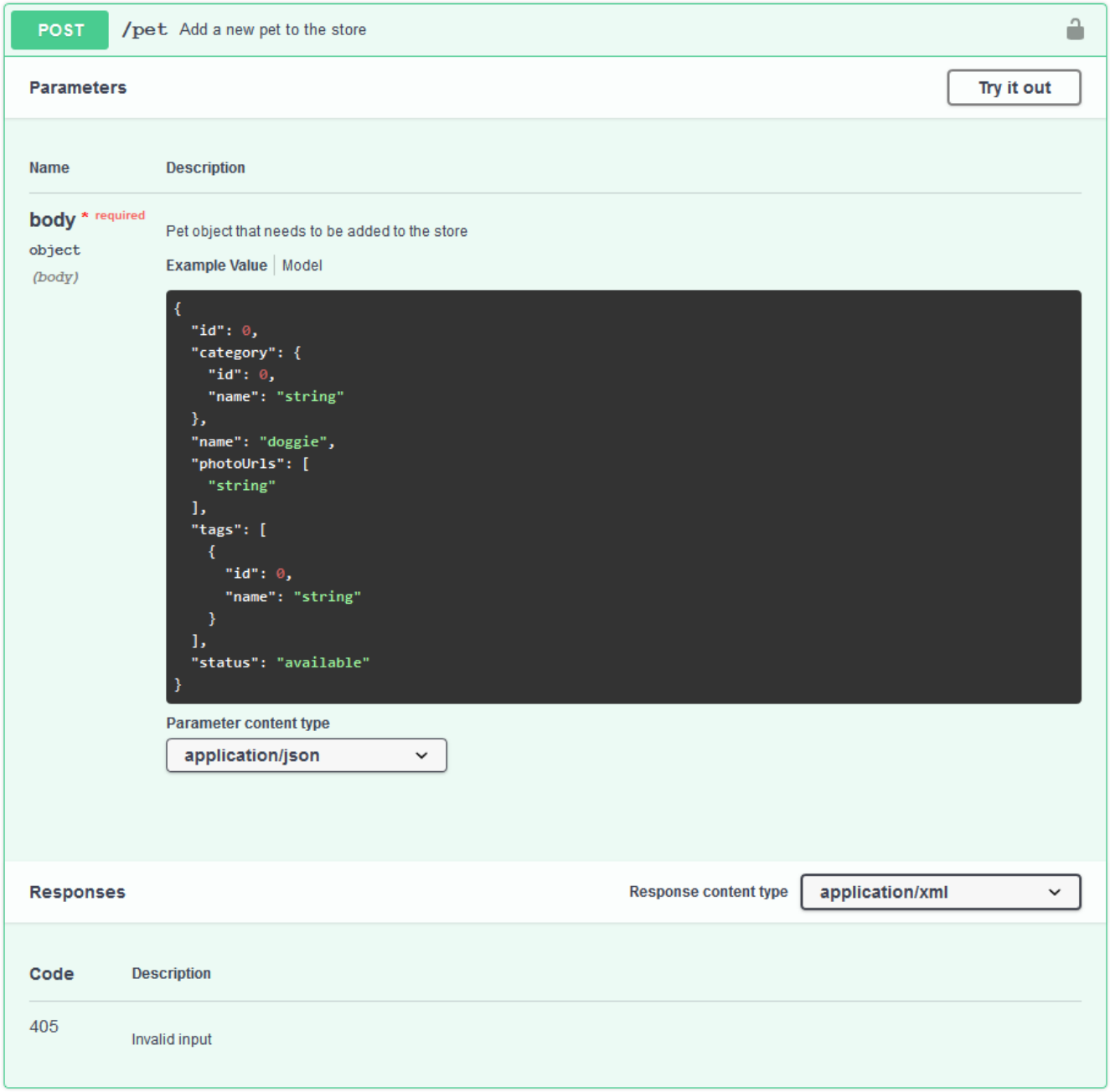


Ilustracja 4. Menu konwersji do OpenAPI Swaggera.

2.2. Korzystanie z edytora Swagger

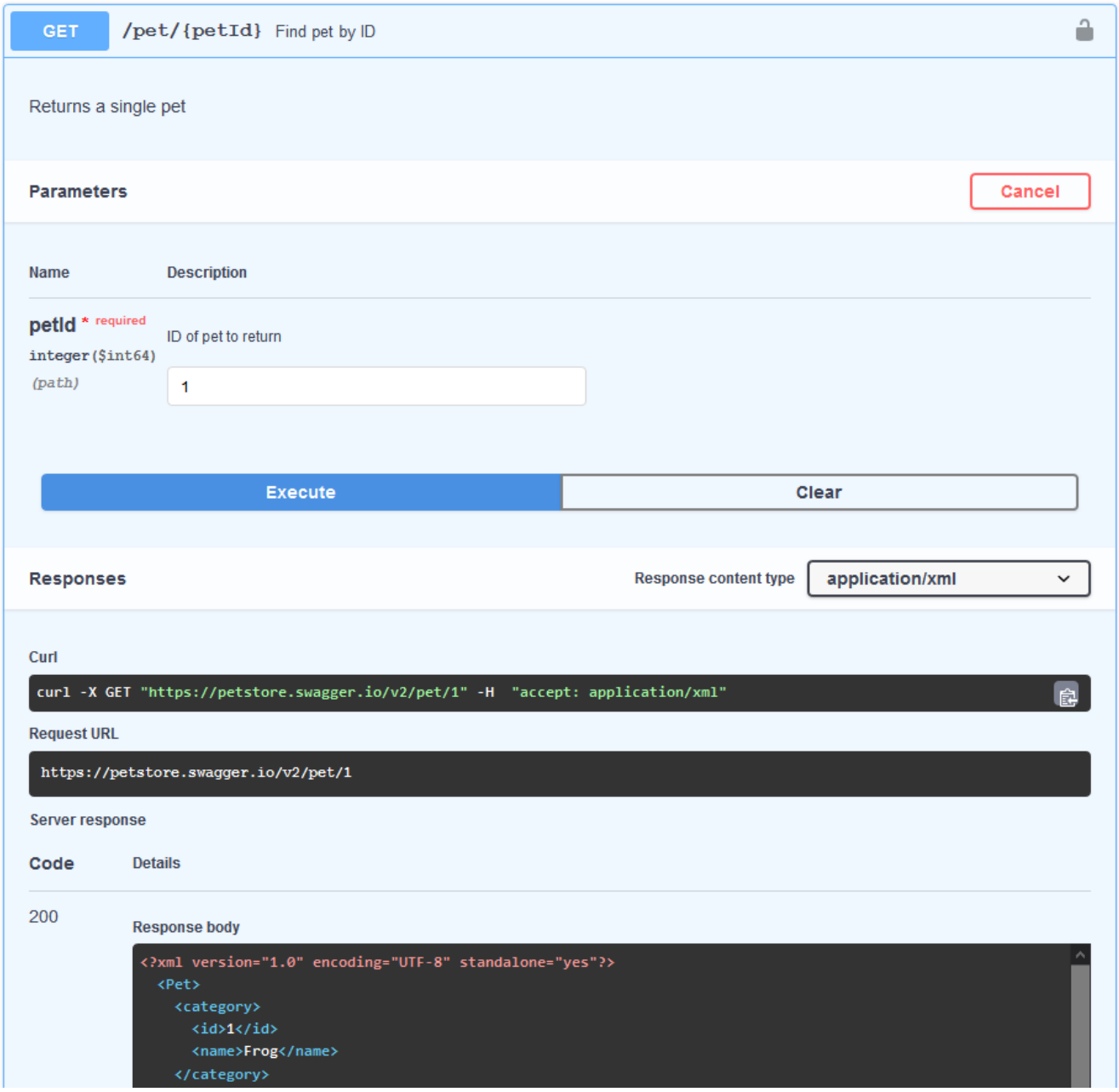
Definicja API w postaci pliku w formacie YAML prezentowana jest w lewym panelu edytora. W nim możemy dokonywać wszelkich modyfikacji. Na bieżąco poprawność zarówno formatowania jak i składni OpenAPI jest sprawdzana, a o ewentualnych błędach jesteśmy informowani.

Prawa część edytora przedstawia zwizualizowany jej wariant co ułatwia śledzenie poszczególnych ścieżek, skojarzonych z nimi metod i formatów komunikatów zarówno na poziomie żądań, jak i odpowiedzi.



Ilustracja 5. Wizualizacja OpenAPI Swaggera.

Każdy tak uwidoczniony rekord można przetestować w oparciu o mockup serwer korzystający z definicji API.



Ilustracja 6. Test żądania w oparciu o mockup.

Warto w tym momencie zapoznać się ze standardem OpenAPI na przykład z użyciem dokumentacji dostępnej pod adresem https://swagger.io/docs/specification/about/.

2.3. Podstawowe składowe definicji ścieżki API

Konfiguracja poszczególnych wywołań ma miejsce w oparciu o format YAML (ważne są wcięcia!):

Kolejno w sekcji paths: podajemy definicję ścieżki wraz ze wskazaniem metody (*get:*, *post:*, *delete:*, *put:*), funkcji kontrolera na poziomie logiki businessowej (*operationId:*), przekazywanymi parametrami (*parameters:*) oraz możliwymi odpowiedziami.

```yaml
paths:
  /pet:
    post:
      tags:
      - "pet"
      summary: "Add a new pet to the store"
      description: ""
      operationId: "addPet"
      consumes:
      - "application/json"
      - "application/xml"
      produces:
      - "application/xml"
      - "application/json"
      parameters:
      - in: "body"
        name: "body"
        description: "Pet object that needs to be added to the store"
        required: true
        schema:
          $ref: "#/definitions/Pet"
      responses:
        "405":
          description: "Invalid input"
      security:
      - petstore_auth:
        - "write:pets"
        - "read:pets"
```

Jeżeli chcemy aby parametr był przekazywany pozycyjnie w ścieżce zamiast w zapytaniu zmieniamy pole in: z *query* na *path*, a w definicji ścieżki dodajemy stosowny placehoder (w powyższym przypadku byłoby to zastąpienie */pet* przez */pet/{name}*).

W parametrach jest również zawarte wskazanie $ref do zdefiniowanych struktur danych umieszczonych w końcowej sekcji dokumentu i wyglądających następująco:

```
definitions:
  Pet:
    type: "object"
    required:
    - "name"
    - "photoUrls"
    properties:
      id:
        type: "integer"
        format: "int64"
      category:
        $ref: "#/definitions/Category"
      name:
        type: "string"
        example: "doggie"
      photoUrls:
        type: "array"
        xml:
          name: "photoUrl"
          wrapped: true
        items:
          type: "string"
      tags:
        type: "array"
        xml:
          name: "tag"
          wrapped: true
        items:
          $ref: "#/definitions/Tag"
      status:
        type: "string"
        description: "pet status in the store"
        enum:
        - "available"
        - "pending"
        - "sold"
    xml:
      name: "Pet"
```

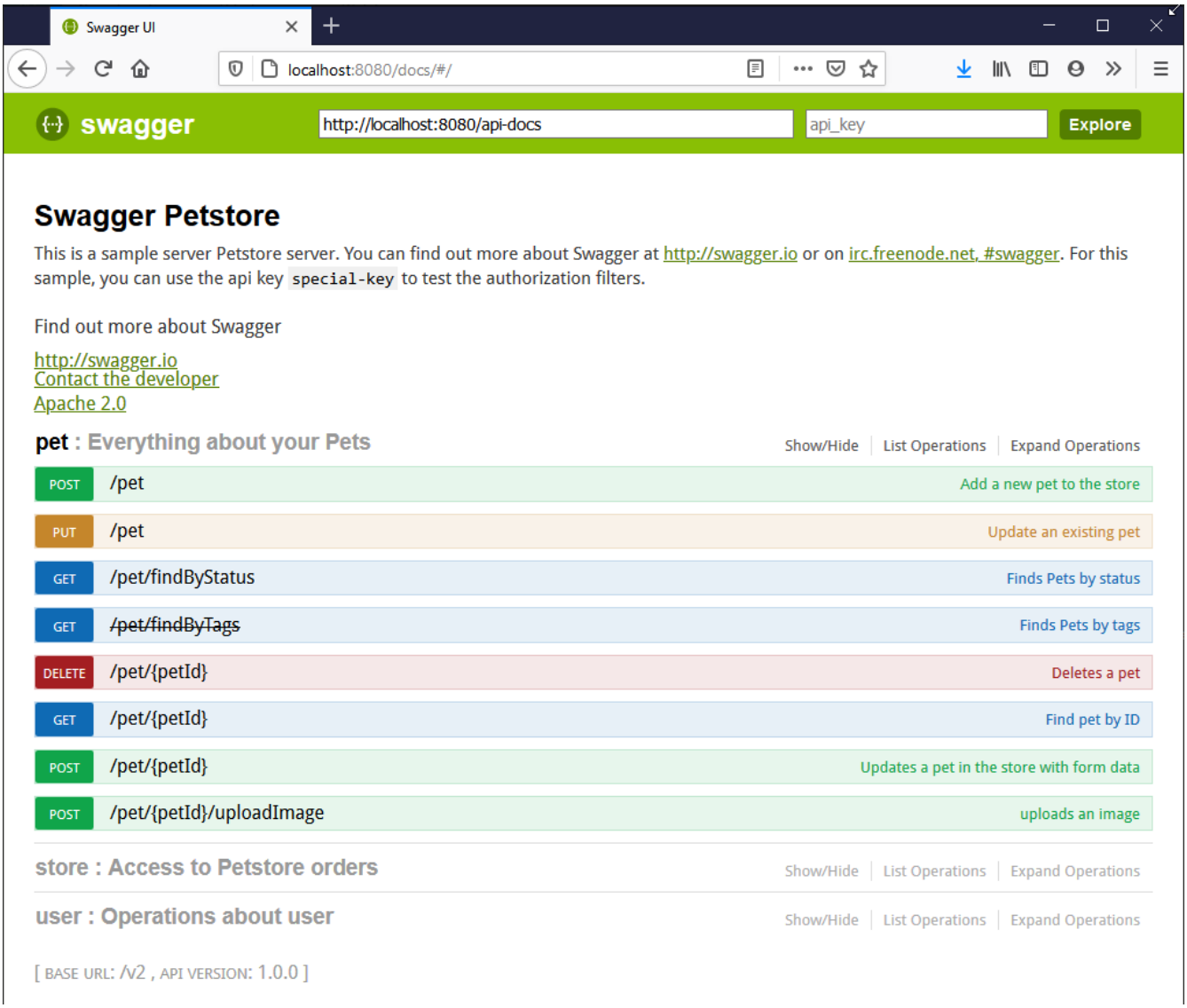
Wskazujemy też przyjmowane i zwracane formaty danych (*consumes:* i *produces:*).

2.4. Generacja szkieletu aplikacji

W oparciu o bieżące API możliwe jest wygenerowanie szkielet aplikacji wybierając z menu **Generate server** > **nodejs-server**. Po pobraniu pliku wypakuj go do głównego folderu projektu. Konieczne będzie lokalnie doinstalowanie koniecznych pakietów korzystając, ale całość operacji wykonane zostanie po tym jak wywołamy *npm* wraz z parametrem *start*

```
λ npm start
--8<--
Your server is listening on port 8080 (http://localhost:8080)
Swagger-ui is available on http://localhost:8080/docs
```

Serwer dostępny jest na wskazanym porcie. Dodatkowo do dyspozycji mamy skróconą postać zwizualizowanego API znaną ze edytora.



Ilustracja 7. Wygląd Swagger-ui.

2.5. Zadanie

Przygotuj prostą definicję API z wykorzystaniem edytora *Swagger*, która dla ścieżki */event/{id}* zdefiniuje **CRUD** w oparciu o metody *GET*, *PUT*, *DELETE* korzystając z *JSON*, dla obiektu zawierającego pola *name* i *date* oraz **C**RUD dla metody *POST* i ścieżki */event*, dla której zwrócony zostaje wygenrowany identyfikator.

wersja: 2020-10-21 23:08:47 +0200

Wszelkie znalezione błędy, nieścisłości czy sugestie odnośnie poszerzenia zawartości można przekazywać poprzez adres: rafal@stegierski.com.

All trademarks are the property of their respective owners.

Copyright © 2020 Rafał Stęgierski

Creative Commons Attribution-NoDerivatives 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-NoDerivatives 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.