
	Politechnika Bydgoska im. J. J. Śniadeckich  <b>Wydział Telekomunikacji, Informatyki i Elektrotechniki</b>		
<b>Przedmiot</b>	Skryptowe języki programowania		
<b>Prowadzący</b>	mgr inż. Martyna Tarczewska		
<b>Temat</b>	<i>FastAPI</i>		
<b>Student</b>			
<b>Nr ćw.</b>		<b>Data wykonania</b>	
<b>Ocena</b>		<b>Data oddania spr.</b>	

## 1. Cel ćwiczenia

Celem tego laboratorium jest wprowadzenie w podstawy frameworka FastAPI w języku Python. Studenci zdobędą praktyczną wiedzę na temat tworzenia prostych API, obsługi parametrów wejściowych, oraz korzystania z dokumentacji automatycznej - z FastAPI oraz Swagger. **Funkcje i zmienne stworzone podczas zajęć powinny być odpowiednio otypowane z użyciem biblioteki typing. Kod powinien być napisany w języku angielskim.**

## 2. Informacje podstawowe

### **FastApi**

FastAPI to nowoczesny, szybki (wysokowydajny) framework sieciowy do tworzenia interfejsów API w języku Python 3.8+ w oparciu o standardowe wskazówki typu Python.

- **Szybki** : Bardzo wysoka wydajność, porównywalna z **NodeJS** i **Go** (dzięki Starlette i Pydantic). [Jeden z najszybszych dostępnych frameworków Pythona](#) .
- **Szybkie kodowanie** : zwiększ prędkość tworzenia funkcji o około 200% do 300%. \*
- **Mniej błędów** : Zmniejsz około 40% błędów spowodowanych przez człowieka (programistę). \*
- **Intuicyjny** : Świetna obsługa edytora. Wszędzie dokończenie . Mniej czasu na debugowanie.
- **Łatwy** : Zaprojektowany tak, aby był łatwy w obsłudze i nauce. Mniej czasu na czytanie dokumentów.
- **Krótko** : Zminimalizuj powielanie kodu. Wiele funkcji z każdej deklaracji parametrów. Mniej błędów.
- **Solidny** : Uzyskaj kod gotowy do produkcji. Z automatyczną interaktywną dokumentacją.
- **Oparte na standardach** : Oparte na otwartych standardach API (i w pełni z nimi kompatybilne): [OtwórzAPI](#)(wcześniej znany jako Swagger) i [Schemat JSON-a](#).

Instalacja:

```
$ pip install fastapi uvicorn
```

Utwórz plik **main.py** zawierający kod:

```
from typing import Union

from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}
```

Wystartuj **serwer** z konsoli:

```
$ uvicorn main:app --reload
```

**Testowanie** działania serwera:

Otwórz w przeglądarce adres <http://127.0.0.1:8000/items/5?q=somequery>

Zobaczysz odpowiedź serwera:

```
{"item_id": 5, "q": "somequery"}
```

Stworzony został interfejs API, który:

- Odbiera żądania HTTP w ścieżkach **/i /items/{item\_id}**.
- Obie ścieżki wykorzystują operacje **GET** (znane również jako metody HTTP).
- Ścieżka **/items/{item\_id}** ma parametr ścieżki **item\_id**, który powinien być liczbą całkowitą.
- Ścieżka **/items/{item\_id}** ma opcjonalny parametr zapytania **str q**.

## API — Interfejs Programowania Aplikacji

Interfejs API jest wymagany, aby aplikacje mogły łączyć się między sobą, dzięki czemu mogą wykonywać odpowiednio zaprojektowane funkcje. Interfejsy programowania aplikacji działają niczym pośrednik, pozwalając programistom na budowanie i wymienianie się danymi pomiędzy różnymi aplikacjami, z których korzystamy na co dzień.

Interfejsy API nie są tak nową ideą, jak by się mogło wydawać, jednak w ostatnim czasie stają się coraz to bardziej popularne, głównie dzięki rozwojowi aplikacji mobilnych, ale również dzięki możliwości połączenia systemów wewnętrznych z usługami firm trzecich w celu wymiany danych pomiędzy aplikacjami.

Technicznie rzecz biorąc, API to kod, który zarządza punktami dostępowymi aplikacji lub serwera.

### Jak działa API — Rodzaje Interfejsów Programowania Aplikacji

Wszystkie API wykonują te same czynności, jednak nieco różnią się od siebie:

- REST API — to inaczej Representational State Transfer API. Przeznaczone są do wykonywania zapytań oraz otrzymywania odpowiedzi za pomocą funkcji HTTP. REST opiera się na czterech różnych poleceniach HTTP. Są to: **GET, PUT, POST i DELETE**. Z REST API korzysta np. Facebook.

Tabela 1. Metody HTTP

Metoda HTTP	Opis
POST	Tworzy nowy zasób.
GET	Pobiera zasób.
PUT	Aktualizuje istniejący zasób.
DELETE	Usuwa zasób.

- SOAP API — (ang. Simple Object Access Protocol) SOAP API w odróżnieniu od REST nie narzuca architektury, a trzyma się określonych standardów. Jest zależny także od systemów programowania opartych na XML.
- RPC API — (ang. Remote Procedure Call), czyli zdalne wywołanie procedury. RPC powstało najwcześniej. Zostało zaprojektowane w taki sposób, aby móc wykonywać kod na innym serwerze. Jeśli użyjemy RPC API za pośrednictwem HTTP, wtedy może być Web API.

## API docs - jak się dostać do dokumentacji API (REST API) - Swagger UI

Otwórz w przeglądarce adres <http://127.0.0.1:8000/docs>

Fast API 0.1.0 OAS3  
/openapi.json

default

GET /items/{item\_id} Read Item Get

Parameters

Try it out

Name	Description
item_id * required integer (path)	
q string (query)	

Responses

Code	Description	Links
200	Successful Response application/json Controls Accept header.	No links
422	Validation Error application/json	No links

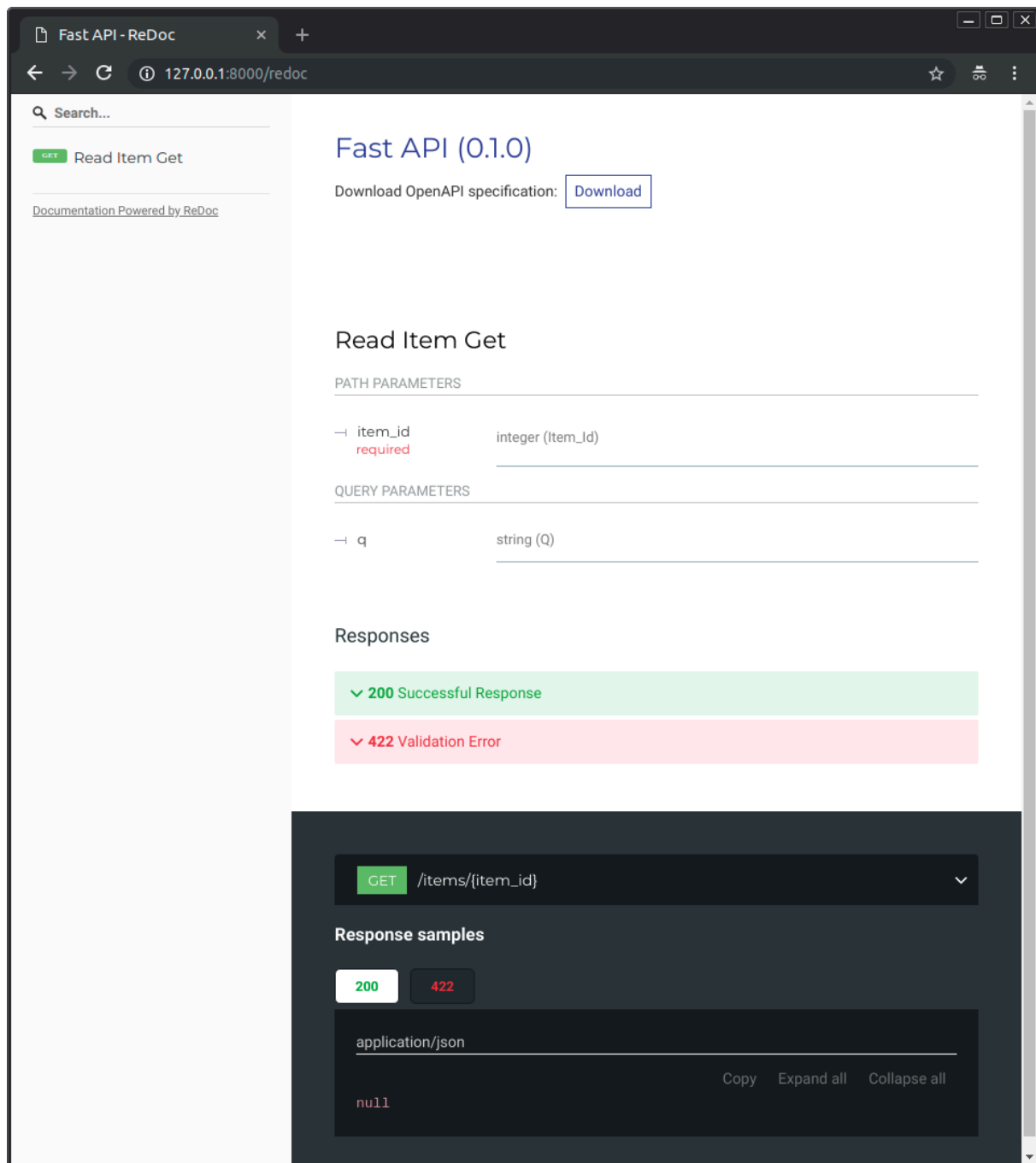
Example Value | Schema

```
{  "detail": [    {      "loc": [        "string"      ]    }  ]}
```

Jest to automatycznie wygenerowana dokumentacja dla utworzonego API dostarczona przez Swagger UI. Dokumentacja umożliwia przegląd możliwości API oraz testowanie endpointów.

## API docs - jak się dostać do dokumentacji API (REST API) - ReDoc

Otwórz w przeglądarce adres <http://127.0.0.1:8000/redoc>



Jest to automatycznie wygenerowana dokumentacja dla utworzonego API dostarczona przez [ReDoc](#). Dokumentacja umożliwia przegląd możliwości API oraz testowanie endpointów.

## Modyfikacja przykładu w pliku *main.py*

W tym przykładzie dodamy **POST** request, który doda nowy obiekt do listy *database*.

```
from typing import Union, List

from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import uvicorn

app = FastAPI()

class Item(BaseModel):
    name: str
    price: float
    item_id: int | None = None
    is_offer: bool | None = False

# Utworzenie prowizorycznej "bazy danych" elementów Item
database: List[Item] = []

# Zwraca wszystkie elementy z databsse
@app.get("/items")
def get_all_items() -> List[Item]:
    return database

# Zwraca item o wybranym id
@app.get("/items/{item_id}")
def read_item(item_id: int) -> Item:
    items = [x for x in database if x.item_id == item_id]
    if not items:
        # Wyrzucamy błąd jeśli nie ma w naszej bazie elementu o podanym
        id
        raise HTTPException(status_code=404, detail="Item not found")
    return items[0]

# Dodaje element do bazy database i zwraca utworzony obiekt - użycie path
parametr i body
@app.post("/items/{item_id}")
def post_item(item_id: int, item: Item) -> Item:
    item = Item(name=item.name, price=item.price, item_id=item_id,
    is_offer=item.is_offer)
```

```

    database.append(item)
    return item

# Dodaje element do bazy database i zwraca utworzony obiekt - użycie
parametru query, path i body
@app.post("/items/q/{item_id}")
async def post_item_q(item_id: int, item: Item, q_name: Union[str, None]
= None) -> Item:
    item = Item(name=item.name, price=item.price, item_id=item_id,
is_offer=item.is_offer)
    if q_name:
        item.name = q_name
    database.append(item)
    return item

if __name__ == '__main__':

```

Po wejściu w dokumentację - <http://127.0.0.1:8000/docs> widzimy aktualizację:

**FastAPI** 0.1.0 OAS 3.1  
/openapi.json

default ^

GET	/items	Get All Items	▼
GET	/items/{item_id}	Read Item	▼
POST	/items/{item_id}	Post Item	▼
POST	/items/q/{item_id}	Post Item Q	▼

Schemas ^

HTTPValidationError > Expand all object

Item > Expand all object

ValidationError > Expand all object

POST /items/{item\_id} Post Item

Parameters

Try it outReset

Name	Description
item_id * required integer (path)	<input type="text" value="123"/>

Request body required

application/json

Example Value | Schema

```
{  "name": "string",  "price": 0,  "is_offer": true}
```

Responses

Code	Description	Links
200	Successful Response <div>Media type<div>application/json</div>Controls Accept header.<div>Example Value   Schema<div><pre>{  "name": "string",  "price": 0,  "item_id": 0,  "is_offer": true}</pre></div></div></div>	No links
422	Validation Error	No links

Spróbujemy dostać się do tego endpointa poprzez naciśnięcie przycisku **Try it out**:

GET /items/{item\_id} Read Item

POST /items/{item\_id} Post Item

CancelReset

Name	Description
item_id * required integer (path)	<input type="text" value="123"/>

Request body required

application/json

```
{  "name": "book",  "price": 110}
```

Execute

Clear

Responses

Curl



Następnie wciskamy przycisk **Execute**, interfejs użytkownika skomunikuje się z utworzonym API, wyśle parametry, pobierze wyniki i pokaże je na ekranie:

The screenshot displays a REST client interface with the following sections:

- Curl:** A text area containing a curl command for a POST request to `http://127.0.0.1:8000/items/123` with headers `-H 'accept: application/json'` and `-H 'content-type: application/json'`, and a JSON body `{ "name": "book", "price": 110 }`.
- Request URL:** A text field showing `http://127.0.0.1:8000/items/123`.
- Server response:** A section with a table showing the response code `200` and details.
- Response body:** A text area showing the JSON response: `{ "name": "book", "price": 110, "item_id": 123, "is_offer": false }`. A `Download` button is visible.
- Response headers:** A text area showing headers: `content-length: 60`, `content-type: application/json`, `date: Mon, 16 Dec 2024 08:10:57 GMT`, and `server: uvicorn`.
- Responses:** A table with columns `Code` and `Description`. It shows a `200 Successful Response` with a `Media type` dropdown set to `application/json`. Below the table, there is an `Example Value` section showing a JSON schema: `{ "name": "string", "price": 0, "item_id": 0, "is_offer": true }`.

## Podsumowując - FastAPI będzie:

1. Sprawdzać, czy istnieje `item_id` w ścieżce dla żądań typu GET i PUT.
2. Sprawdzać, czy `item_id` ma typ `int` dla żądań typu GET i PUT.
  - a. Jeśli nie, klient zobaczy użyteczny, czytelny błąd.
3. Sprawdzać, czy istnieje opcjonalny parametr zapytania o nazwie `q` (jak w `http://127.0.0.1:8000/items/foo?q=somequery`) dla żądań typu GET.
  - a. Parametr `q` został zadeklarowany z `z = None`, co oznacza, że jest opcjonalny.
  - b. Bez wartości `None` byłby wymagany (tak jak ciało w przypadku żądań typu PUT).
4. Dla żądań typu PUT do `/items/{item_id}` odczytywać ciało jako JSON:
  - a. Sprawdzać, czy ma wymagany atrybut `name`, który powinien być typu `str`.
  - b. Sprawdzać, czy ma wymagany atrybut `price`, który musi być typu `float`.
  - c. Sprawdzać, czy ma opcjonalny atrybut `is_offer`, który powinien być typu `bool`, jeśli jest obecny.
  - d. Wszystko to będzie działać również dla zagnieżdżonych obiektów JSON.
5. Automatycznie przekształcać dane z `i` do formatu JSON.
6. Dokumentować wszystko przy użyciu OpenAPI, co może być wykorzystane przez:
  - a. Systemy interaktywnej dokumentacji.
  - b. Systemy automatycznego generowania kodu klienta dla wielu języków.
7. Dostarczać dwa interaktywne interfejsy internetowe do dokumentacji.

### 3. Przebieg ćwiczenia

Przygotowanie: na podstawie kodu z instrukcji utwórz własny obiekt przy pomocy **BaseModel** z biblioteki pydantic (powinien posiadać co najmniej 3 pola).

#### 3.1. Zadanie 1.

Dodaj 2 endpointy typu GET do utworzonego programu, pierwszy zwróci wszystkie elementy z listy, drugi zwraca element o wybranym id (użyj path parametru).

#### 3.2. Zadanie 2.

Dodaj 1 endpointy typu POST do utworzonego programu, ma on dodawać element do list **database**. Użyj parametrów body, path lub query.

#### 3.3. Zadanie 3.

Dodaj 1 endpoint typu PUT do utworzonego programu, ma on aktualizować element o podanym id. Użyj parametrów body (dane obiektu do aktualizacji) i path (id).

#### 3.4. Zadanie 4.

Dodaj 1 endpoint typu DELETE do utworzonego programu, ma on usuwać element o podanym id. Użyj parametru query.

#### 3.5. Zadanie 5.

Zobrazuj oraz przetestuj działanie utworzonych endpointów za pomocą wybranej dokumentacji automatycznej - Swagger UI, ReDoc.

#### 3.6. Zadanie 6.

Na podstawie artykułu: <https://fastapi.tiangolo.com/async/#in-a-hurry> wyjaśnij dlaczego stosowanie funkcji async jest dobrą praktyką.

### Zadanie dodatkowe:

<https://fastapi.tiangolo.com/tutorial/body-fields/> - korzystając z dokumentacji dodaj opis dla każdego parametru utworzonego obiektu.

### 4. Sprawozdanie

Sprawozdanie z laboratorium powinno zawierać:

- wypełnioną tabelę z początku instrukcji,
- kody programów będących rozwiązaniami wszystkich zadań wraz z komentarzami,
- demonstrację działania programu,
- odpowiedzi na pytania zawarte w zadaniach,
- wnioski.