Testowanie i Jakość Oprogramowania

Akademia Tarnowska

Katedra Informatyki

Tomasz Gądek

Wykład 4: Flask

JSON

JSON (*ang. JavaScript Object Notation*) jest prostym formatem wymiany danych. Jego definicja oparta jest na podzbiorze języka JavaScript. JSON jest formatem tekstowym, całkowicie niezależnym od języków programowania.

JSON, cechy i zastosowanie

Cechy:

- Lekki
- Czytelny

Zastosowanie:

- API
- Bazy danych
- Komunikacja między systemami

JSON, struktura

Podstawowa struktura JSON'a:

```
{
  "klucz" : wartość
}
```

klucz: **string**

wartość: string | number | object | array | bool | null

JSON, przykłady

- Liczba:
  `{ "age": 25 }`
- Ciąg tekstowy:
  `{ "name": "Jan" }`
- Zagnieżdżony obiekt:
  `{ "address": { "city": "Warszawa", "postcode": "00-001" } }`

JSON, przykłady

- Kolekcja:
  `{ "products": ["Jabłko", "Banan", "Pomarańcza"] }`
- Wartość logiczna:
  `{ "active": true }`
- Wartość null:
  `{ "email": null }`

JSON jako słownik

JSON (format tekstowy)

```
{
  "name": "Jan",
  "age": 30,
  "city": "Warsaw"
}
```

JSON (słownik w Pythonie)

```
data = {
  "name": "Jan",
  "age": 30,
  "city": "Warsaw"
}
```

JSON, konwersja TXT > DICT

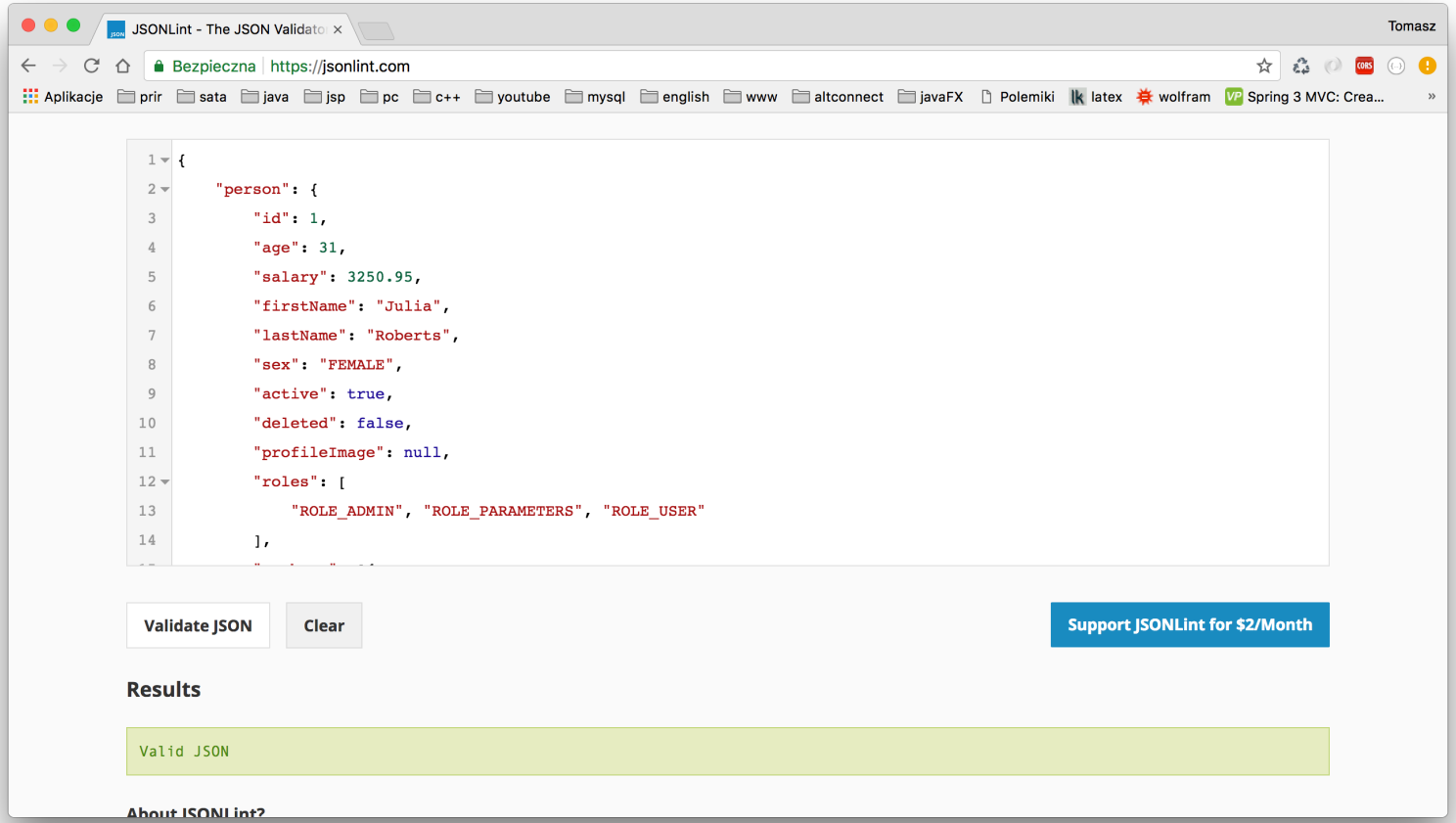
```
import json

json_text = '{"name": "John", "age": 30, "city": "Warsaw"}'

data = json.loads(json_text)

print(data)  # {'name': 'John', 'age': 30, 'city': 'Warsaw'}
```

JSON, konwersja DICT > TXT

```
import json

data = {
    "name": "John",
    "age": 30,
    "city": "Warsaw"
}

json_text = json.dumps(data)

print(json_text)  # {"name": "John", "age": 30, "city": "Warsaw"}
```

JSON, walidacja

Narzędzie jsonlint umożliwia weryfikację, czy nasz JSON zawiera poprawną strukturę.



HTTP

- HyperText Transfer Protocol
- Protokół komunikacyjny dla sieci WWW
- Oparty na modelu żądanie-odpowiedź

Żądanie HTTP

- Zapytanie wysyłane do serwera
- Składa się z metody, nagłówków i treści (body)

Odpowiedź HTTP

- Odpowiedź serwera na zapytanie
- Zawiera kod statusu, nagłówki i treść (body)

Headers / Body

Headers:

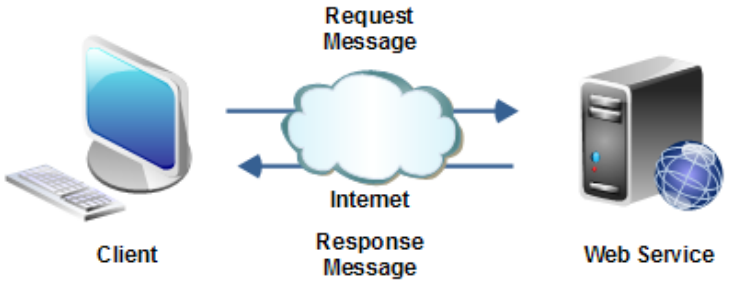
- Dodatkowe informacje w request i response (`Content-Type` , `Authorization` , `User-Agent`).

Body:

- Treść żądania lub odpowiedzi (może zawierać dane w formacie JSON, XML, tekstowym).

Co to jest Web Service?

W uproszczeniu **Web Service** to mechanizm umożliwiający wywołanie pewnej funkcjonalności za pośrednictwem Internetu.



- Usługa sieciowa dostępna przez HTTP.
- Wykorzystuje API do wymiany danych.

REST / API

REST (*ang. Representational State Transfer*) to styl projektowania API, który koncentruje się na tym, jak powinny wyglądać zapytania i odpowiedzi (kolokwialnie: **jak ma wyglądać API?**).

API (*ang. Application Programming Interface*) to zestaw reguł definiujących komunikację między systemami komputerowymi oraz między systemem komputerowym a człowiekiem (kolokwialnie: **jak komunikować się z systemem komputerowym?**).

CRUD

Podstawowe operacje na zasobach

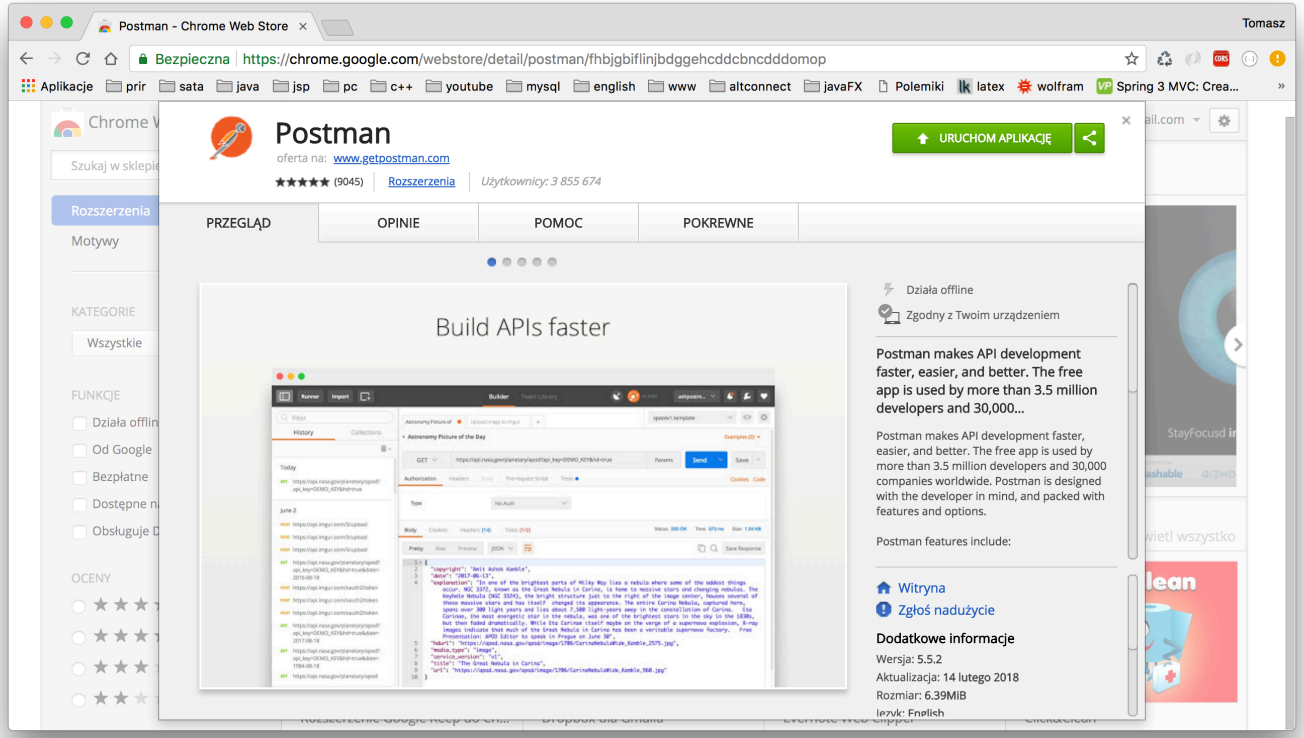
- **C**reate (POST)
- **R**read (GET)
- **U**pdate (PUT)
- **D**elete (DELETE)

Kody odpowiedzi HTTP

- **1xx** (kody informacyjne)
- **2xx** (kody powodzenia)
- **3xx** (kody przekierowania)
- **4xx** (kody błędu aplikacji klienta)
- **5xx** (kody błędu serwera)

Postman

Postman umożliwia testowanie żądań HTTP (jest to wtyczka dla przeglądarki **Google Chrome**).



Flask

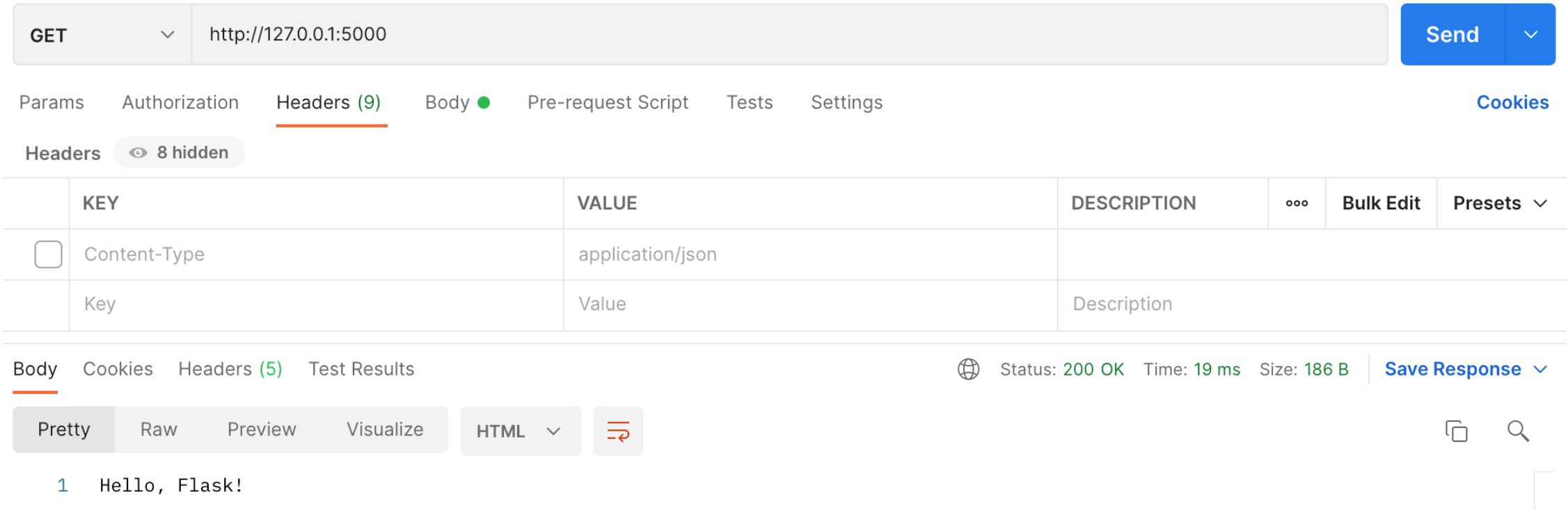
- Mikroframework webowy dla Pythona
- Lekki, elastyczny, rozszerzalny
- Obsługuje routing, request, response, JSON

Flask, pierwsze uruchomienie

Tworzymy plik `app.py`, następnie uruchamiamy `python app.py` (http://127.0.0.1:5000).

```
from flask import Flask
app = Flask(__name__)


@app.route('/')
def home():
    return "Hello, Flask!"


if __name__ == '__main__':
    app.run(debug=True)
```

Flask, pierwsze uruchomienie

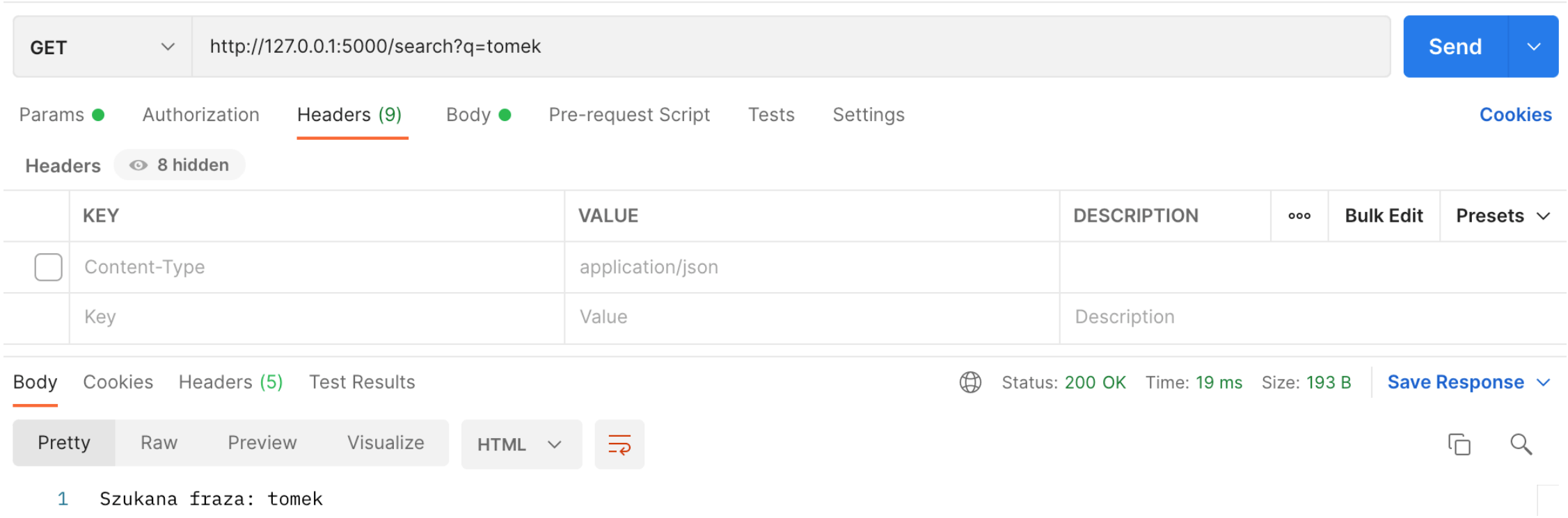


Jak odczytać parametry żądania HTTP?

```
from flask import Flask
from flask import request

app = Flask(__name__)

@app.route('/search')
def search():
    query = request.args.get('q')
    app.logger.info(f'Odczytana fraza {query}.')

    return f'Szukana fraza: {query}'


if __name__ == '__main__':
    app.run(debug=True)
```

Nie stosuj print() do logowania!

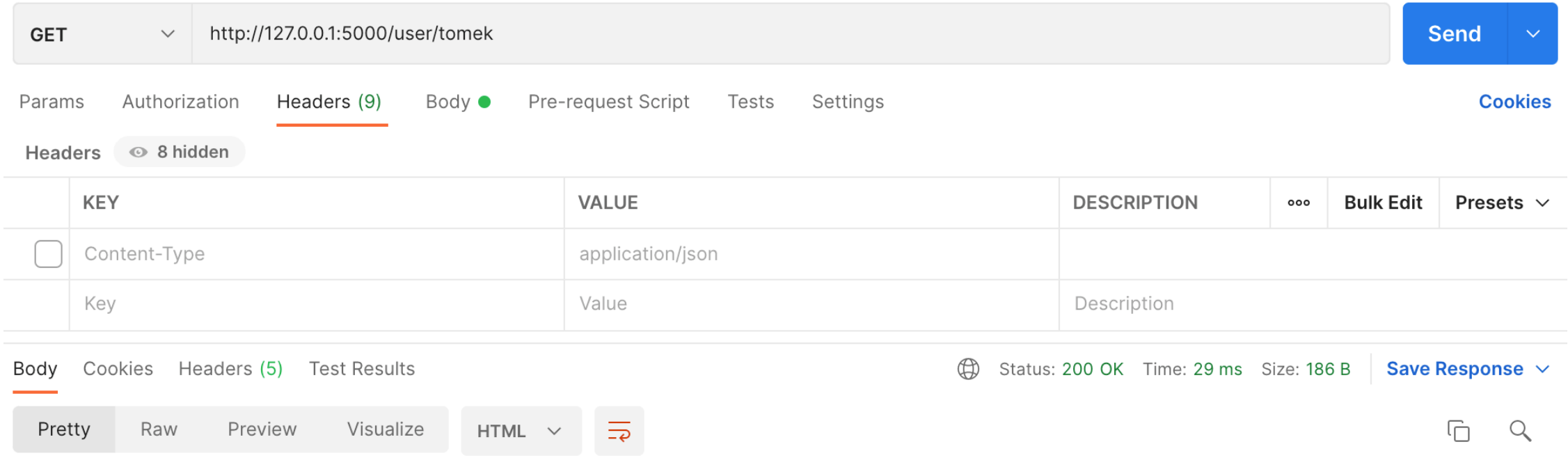
Jak odczytać parametry żądania HTTP? (Body)



Jak odczytać dane z ścieżki żądania HTTP?

```
from flask import Flask

app = Flask(__name__)

@app.route('/user/<username>', methods=['GET'])
def get_user(username):
    return f'Witaj, {username}!'


if __name__ == '__main__':
    app.run(debug=True)
```

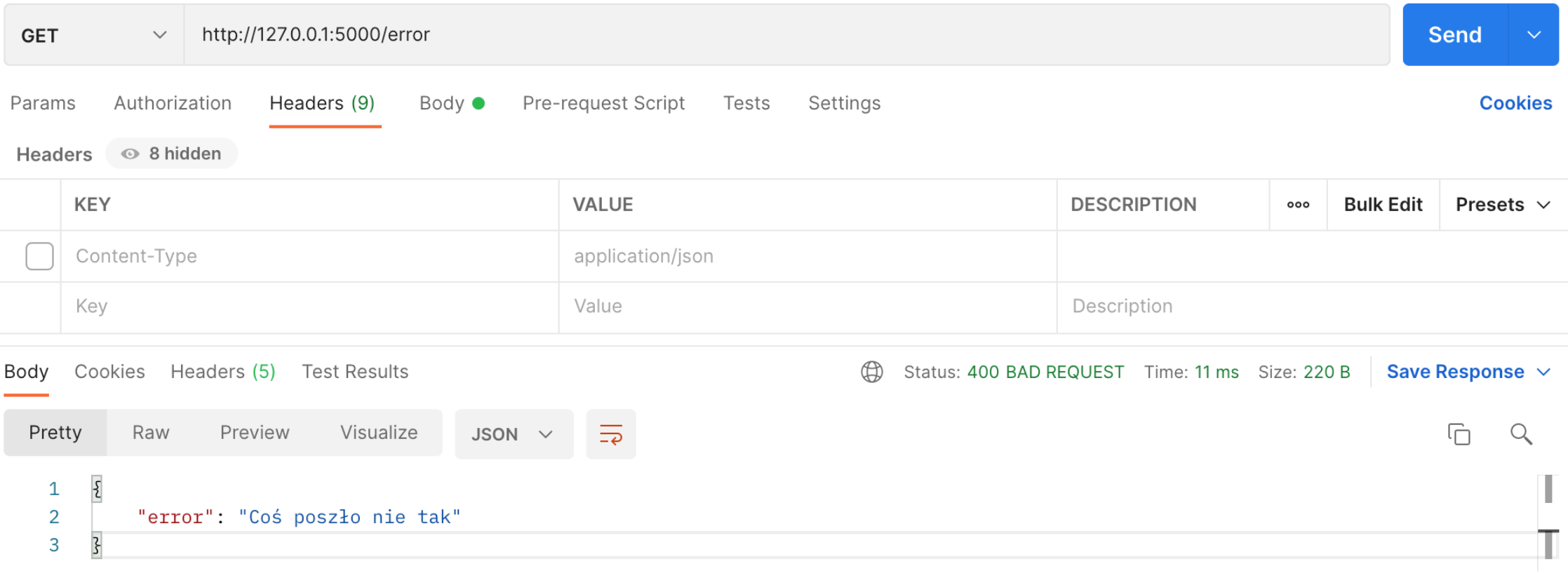
Jak odczytać dane z ścieżki żądania HTTP? (Body)



Jak zwrócić JSON z kodem błędu?

```
from flask import Flask
from flask import jsonify

app = Flask(__name__)

@app.route('/error')
def error():
    return jsonify({"error": "Coś poszło nie tak"}), 400


if __name__ == '__main__':
    app.run(debug=True)
```

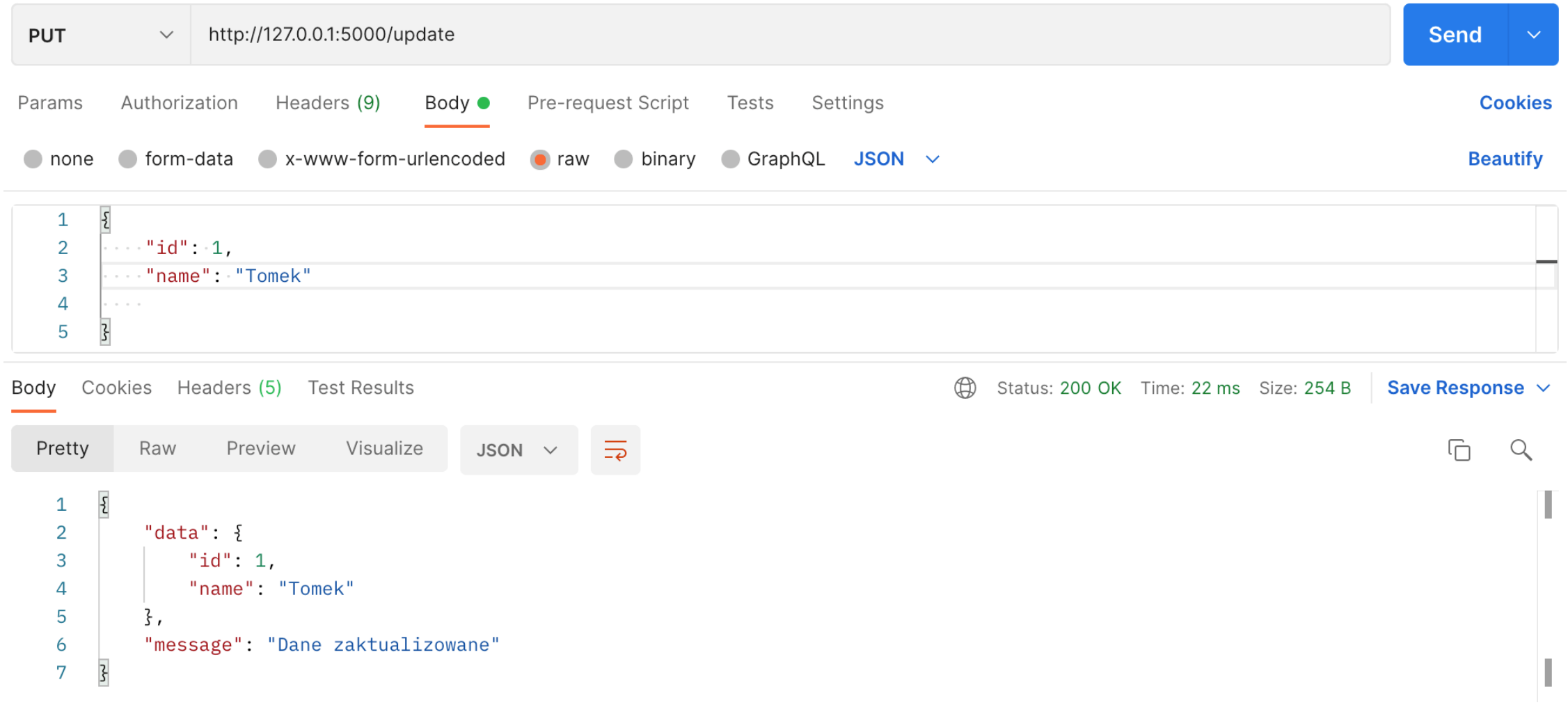
Jak zwrócić JSON z kodem błędu? (Body)



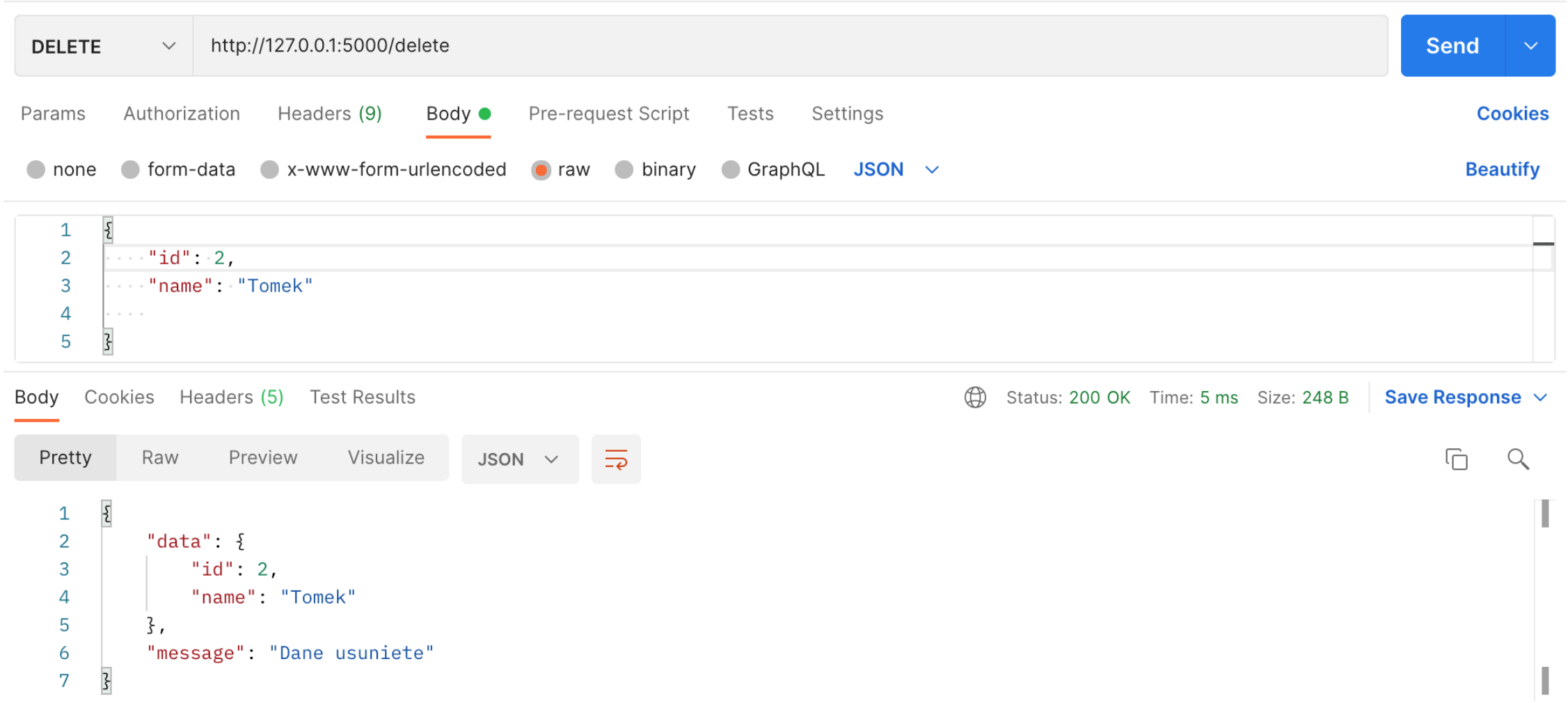
Przykład usługi PUT i DELETE

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/update', methods=['PUT'])
def update():
    data = request.json
    return jsonify({"message": "Dane zaktualizowane", "data": data})

@app.route('/delete', methods=['DELETE'])
def delete():
    data = request.json
    return jsonify({"message": "Dane usuniete", "data": data})

if __name__ == '__main__':
    app.run(debug=True)
```

Przykład usługi PUT (Body)



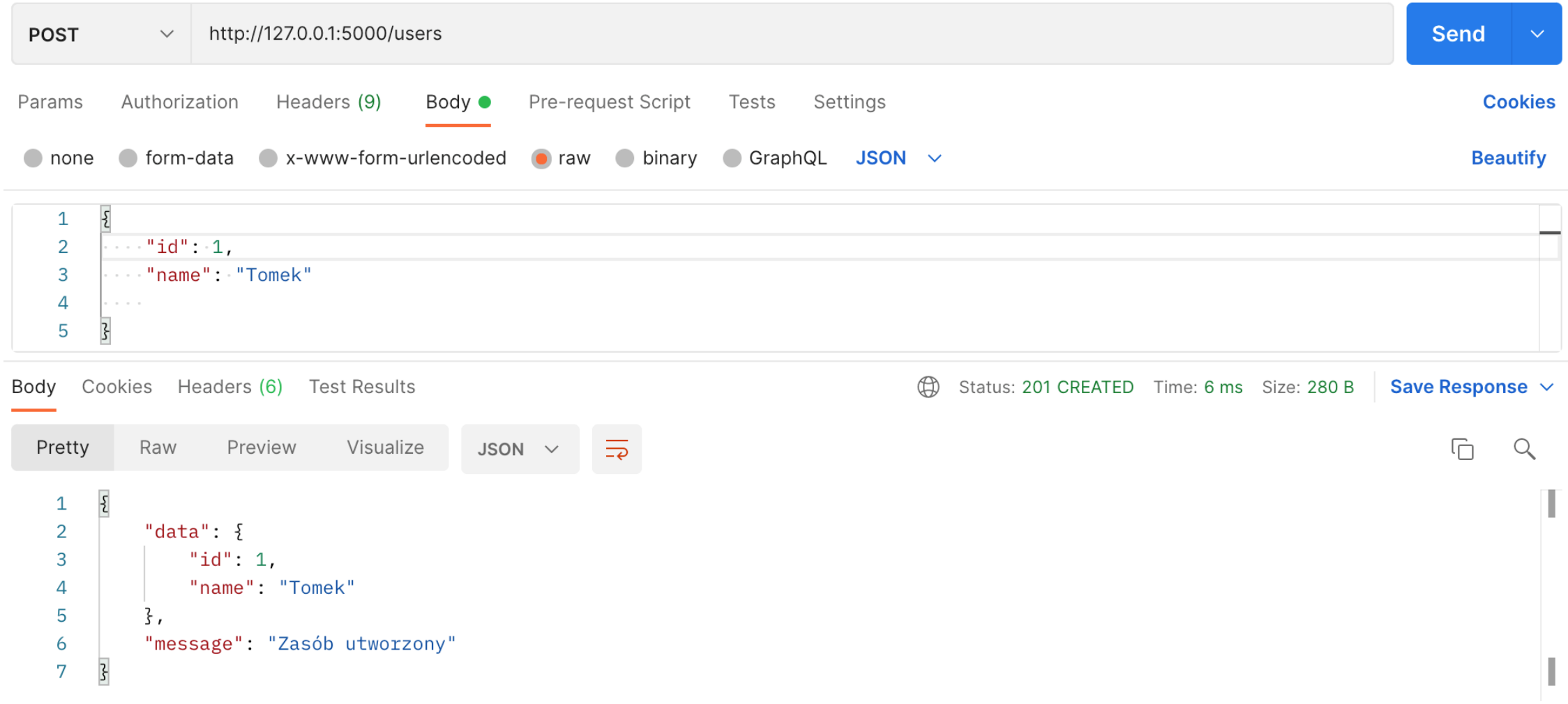
Przykład usługi DELETE (Body)



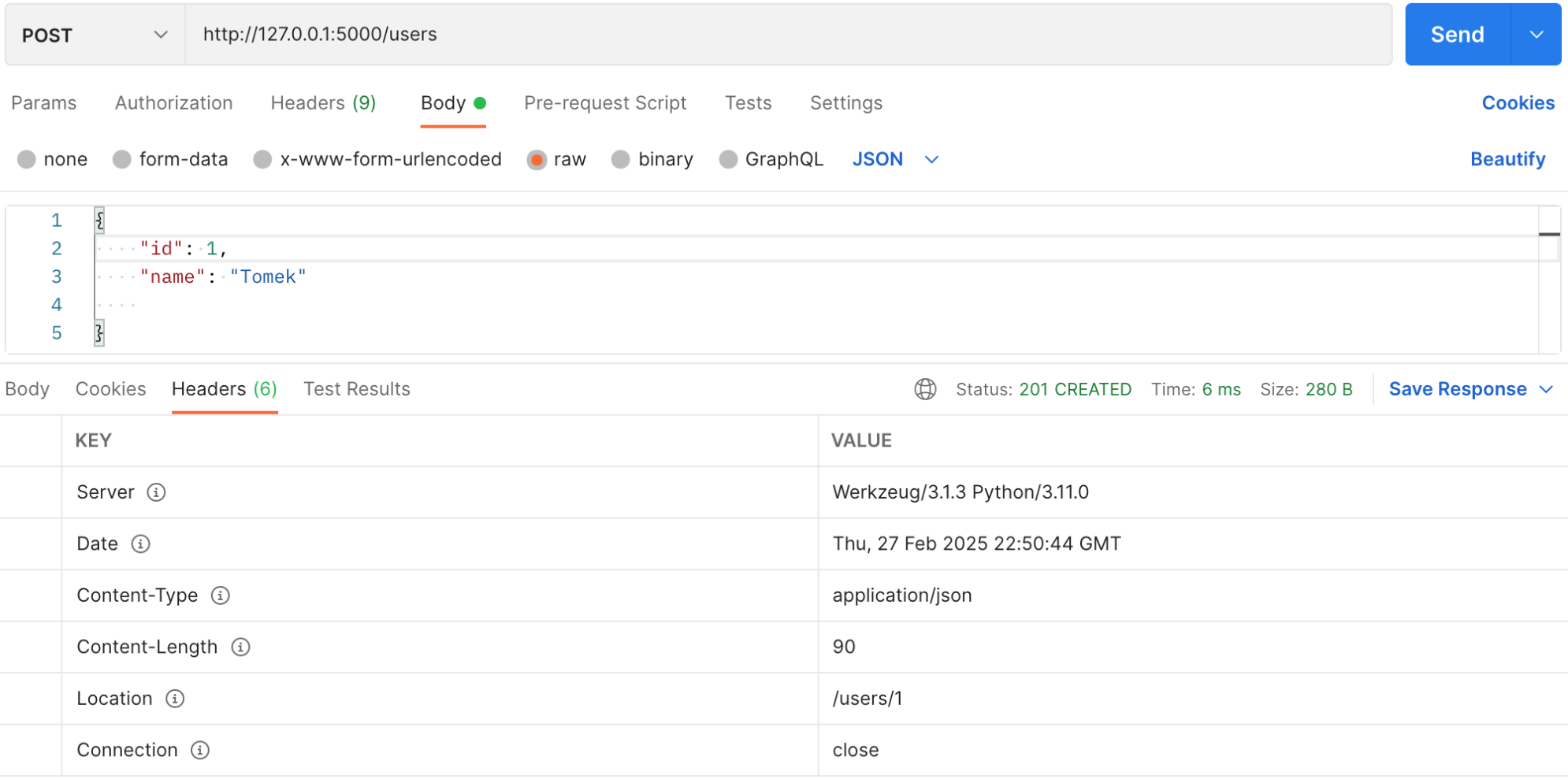
Przykład usługi POST

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/users', methods=['POST'])
def create_user():
    data = request.get_json()
    response = {"message": "Zasób utworzony", "data": data}
    return jsonify(response), 201, {"Location": f"/users/{data.get('id')}"}

if __name__ == '__main__':
    app.run(debug=True)
```

Przykład usługi POST (Body)



Przykład usługi POST (Headers)



Koniec