testerzy.pl

# Wstęp do testów integracji systemów

Testy oprogramowania w literaturze są najczęściej podzielone na cztery poziomy: jednostkowe, integracyjne, systemowe oraz akceptacyjne. Testy integracyjne dzieli się z kolei na testy małe i duże. Testy integracyjne małe dotyczą na ogół tylko jednego systemu, najczęściej pisze je programista, rzadziej tester. Okazuje się jednak, że testy integracyjne duże (dotyczące integracji systemów) nie są proste i stanowią pewną barierę dla początkujących testerów albo dla testerów, którzy zostali postawieni przed zadaniem przeprowadzenia takich testów pierwszy raz. Artykuł przeznaczony jest nie tylko dla osób testujących, lecz także dla programistów oraz kierowników projektów. Celem jego napisania jest zwrócenie uwagi na problemy, jakie mogą nastąpić podczas projektowania dużych testów integracyjnych w przypadku ich wykonywania przez początkujących testerów.

## 1. Testy integracyjne systemowe

Testy integracyjne systemowe, np. testy API stają się bardzo ważne w dzisiejszych czasach. Istnieje mnóstwo systemów informatycznych uruchamianych poprzez przeglądarkę internetową. Takie systemy są oparte na web serwisach lub zintegrowane z innymi systemami centralnymi w instytucjach, które je udostępniają. Takim przykładem mogą być bankowe konta internetowe, Ebooki, inne konta założone na portalu udostępnionym przez instytucję finansową. [1]

Aby zweryfikować poprawność działania wyżej wspomnianych aplikacji, należy oprócz testów GUI przeprowadzić testy integracyjne, które już wymagają pewnej technicznej wiedzy. Niestety, nie każdy tester taką wiedzę posiada, zwłaszcza początkujący tester. Kierownicy testów lub kierownicy projektów muszą zdawać sobie sprawę z tego, że integracja nie jest wbrew pozorom prosta do zrozumienia.

Powstaje wobec tego pytanie: Jak wprowadzić nową osobę testującą do projektowania i przeprowadzania tego rodzaju testów? Na to pytanie będziemy próbowali znaleźć odpowiedź w następnym rozdziale. Dodatkowym problemem może być środowisko testowe, które nie jest identyczne ze środowiskiem klienta. Wynika to z różnych czynników, np. brak udostępnienia danych (instytucja finansowa nie udostępni danych z systemu centralnego ani nie pozwoli na postawienie takiego systemu w firmie piszącej oprogramowanie), duży koszt utrzymania środowiska, które odzwierciedla rzeczywistość, itd.

## 2. Zrozumienie integracji systemowych

Najprościej mówiąc, integracja pomiędzy dwoma lub więcej systemami ma postać:

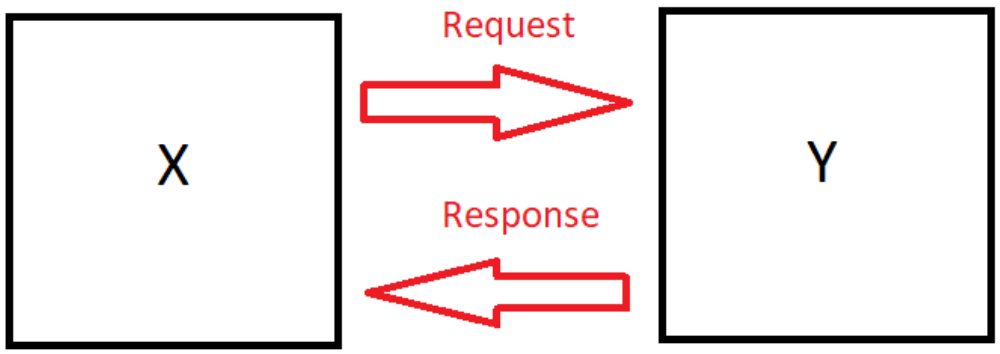
a) przesyłania danych z jednego systemu do drugiego,
b) zbierania danych z kilku systemów i ich zapis w systemie centralnym,
c) przesyłania danych z centralnego systemu do innych systemów,
d) oczekiwania na odpowiedź od innego lub kilku systemów.

Rodzajów integracji można wymienić oczywiście więcej, ale nie jest to główny cel artykułu.

Aby wysłać dane z jednego sytemu do drugiego, należy te dane odpowiednio oznaczyć. Przykładowo, jeśli chcemy z systemu X wysłać do systemu Y informacje:

- **Imię: „Marek”,**
- **Nazwisko: „Kowalski”**,

to przesyłamy nie tylko wartości, ale również nazwę atrybutu, do którego przypisana jest wartość. Atrybutami są tutaj **Imię** oraz **Nazwisko**, a ich wartościami są odpowiednio **Marek** oraz **Kowalski**. Nazwa atrybutu może być analogiczna jak nazwa kolumny w bazie danych z systemu Y, ale nie zawsze tak jest w praktyce. Zapis, edycja lub odczyt danych odbywa się za pomocą **API.** Jest to kod umożliwiający komunikację między różnymi procesami, programami lub systemami [3]. Jest to również sposób, rozumiany jako ściśle określony zestaw reguł i ich opisów, w jaki systemy informatyczne komunikują się między sobą. Poza kodem programu, API jest utworzone również z metod (funkcji wielu zmiennych), które są wywoływane w celu modyfikacji lub zapisu danych. Jeśli użytkownik wywołuje metodę API świadomie lub nie, to mówimy, że wywołuje **żądanie (*ang. request*).** Jeżeli natomiast użytkownik dostaje odpowiedź za pomocą metody z API, to mówimy, że otrzymał **odpowiedź (*ang. response*):**




Właściciele portali internetowych np. GUS często udostępniają dokumentację do wybranych metod API po to, aby programiści mogli pisać aplikacje, które odczytują dane z GUS-u, np. wyszukiwanie danych firmy po numerze NIP lub REGON. Przykładowy opis jednej metody API może mieć postać:

| Nazwa atrybutu | Nazwa kolumny w bazie danych systemu Y | Opis |
|---|---|---|
| **Id** | Id (PK, not null) | Id |
| **LastEditDate** | (datetime, null) | Data ostatniej edycji |
| **Borough** | Borough (nvarchar(20), null) | Gmina |
| **City** | City (nvarchar(40), null) | Miasto |
| **District** | District (nvarchar(20), null) | Ulica |
| **HouseNumber** | HouseNumber (nvarchar(10), null) | Numer domu |
| **Post** | Post (nvarchar(20), null) | Poczta |
| **PostCode** | PostCode (nvarchar(3), null) | Kod pocztowy |
| **Street** | Street (nvarchar(80), null) | Ulica |

Załóżmy, że chcemy odczytać adres pewnego kontrahenta. Klikając „edycja” na liście kontrahenta tak naprawdę przekazujemy id tego kontrahenta do pewnej metody z zestawu API, a dzięki temu zostaną odczytane dane tego kontrahenta, którego chcemy podglądnąć. Przykładowa taka metoda może mieć postać:

**Przykład 1.**

http://cms.softwarehouse.pl/Services/CustomerService.svc/ReadAddress?Id=b2aed483-390c-4fa8-8098-5b8b9d06379c
gdzie:

- ReadAddress - nazwa metody odczytującej dane adresowe,
- Id=b2aed483-390c-4fa8-8098-5b8b9d06379c – id kontrahenta z bazy danych z systemu Y,
- CustomerService.svc - nazwa serwisu zawierającego metodę ReadAddress,
- http://cms.softwarehouse.pl/Services - adres, który zawiera wszystkie podzbiory metod API.

Wyżej przedstawiony przykład to metoda REST-owego web serwisu (czyli również API). Metoda *ReadAddress* ma taką własność, że Id z wartością jest przekazywane po znaku „?”. Nie zawsze jednak musi tak być. REST to akronim od *Representational State Transfer*. W przypadku REST-owych web serwisów mamy adresy URL, które są pewnego rodzaju identyfikatorami. Wysyłamy na te adresy zapytanie, które może być zarówno JSON-em, XMLem, ale też zwykłym tekstem czy danymi binarnymi. Istnieją również web serwisy SOAP-owe oparte o dane w postaci XML, ale zasada ich działania jest analogiczna - wysyłane jest żądanie z odpowiednio wprowadzonymi danymi, a następnie przychodzi odpowiedź na żądanie.

Każda odpowiedź zawiera atrybut o nazwie ***status***, który przyjmuje wartości liczbowe. Wspomniane wartości informują o poprawności albo o rodzaju błędu podczas próby wysłania żądania. Istnieją

cztery grupy takich statusów, 2xx, 3xx, 4xx, 5xx [4]. Status 200 oznacza, że zapytanie zostało przetworzone i zwróciło wynik. Status 400 oznacza, że zapytanie zostało źle skonstruowane. Status 404 oznacza, że żądany zasób nie istnieje lub ktoś mógł usunąć metodę. Z kolei status 500 oznacza błąd dostępu do serwera. Podczas testów, w momencie tworzenia API, pojawia się status 502 *(Bad gateway*), co oznacza, że pewien węzeł pośredniczący między klientem a serwerem dostał niepoprawną odpowiedź od poprzedniego węzła.

W przypadku metod REST-owych wyróżniamy cztery rodzaje metod:

- *GET — pobieranie (kolekcji, jak i pojedynczego elementu),*
- *POST — tworzenie (kolekcji jak i pojedynczego elementu),*
- *PUT — aktualizacja (tylko pojedynczego elementu),*
- *DELETE — usuwanie (tylko pojedynczego elementu).*

W praktyce (w przypadku web serwisów REST-owych) dodawanie, odczyt, usuwanie oraz edycję metod można obsłużyć POST-em.

Podczas projektowania testów API należy pamiętać o tym, że testy powinny zawierać przypadki negatywne. Testy negatywne są bardzo ważne w przypadku testowania web serwisów. Programiści wykorzystują API do uzyskiwania dostępu do usług zewnętrznych. Wobec tego w wyniku defektu może zaistnieć taka sytuacja, że będzie możliwe użycie API w sposób niezgodny z jego przeznaczeniem, a to z kolei oznacza konieczność wprowadzenia odpornych mechanizmów obsługi błędów. Typowymi testami negatywnymi w przypadku web serwisów (jak i również API) mogą być następujące przypadki:

a) podanie nieprawidłowego formatu danych dla pewnego atrybutu,
b) brak podania wymaganych wartości,
c) nieautoryzowany dostęp do metod – podania błędnej wartości klucza (*Keyidentifier*) lub brak klucza,
d) brak certyfikatu, jeśli jest wymagany,
e) wywołanie tego samego żądania kilka razy pod rząd – sprawdza się w ten sposób, czy nie występują zdublowane transakcje lub obiekty w bazie danych (w naszym przypadku w systemie Y),
f) obsługa wyjątków,
g) poprawność obsługi komunikatów błędów,
h) długość znaków zmiennych i stałych.

**Przykład 2.**

Załóżmy, że chcemy dokonać modyfikacji danych adresowych pewnego kontrahenta. W takim przypadku należy w tym samym czasie:

a) wywołać metodę, która taką czynność umożliwi,
b) zidentyfikować kontrahenta (np. poprzez ID z bazy danych z systemu Y),
c) wpisać dane, które chcemy modyfikować do ciała metody:

http://cms.softwarehouse.pl/Services/CustomerService.svc/UpdateAdres

```
{
        "PostCode":"12-512",
        "Borough":"Tyczyn",
        "Street":"Polna",
        "Post":"Tyczyn",
        "ID":"955f7835-a8f7-408f-8aad-dc40856c31e0",
}
```

Metoda w tym przykładzie jest tak zdefiniowana, że nie podaje się parametrów w adresie URL, natomiast w ciele metody w formacie JSON. Testowanie takiej metody musi obejmować przypadki pozytywne i negatywne. Z pewnością należy sprawdzić, czy brak konkretnego Id kontrahenta wywoła odpowiednie ostrzeżenie.

**Przykład 3.**

Chcąc przykładowo usunąć adres należy wywołać metodę, która ten adres usuwa, czyli:

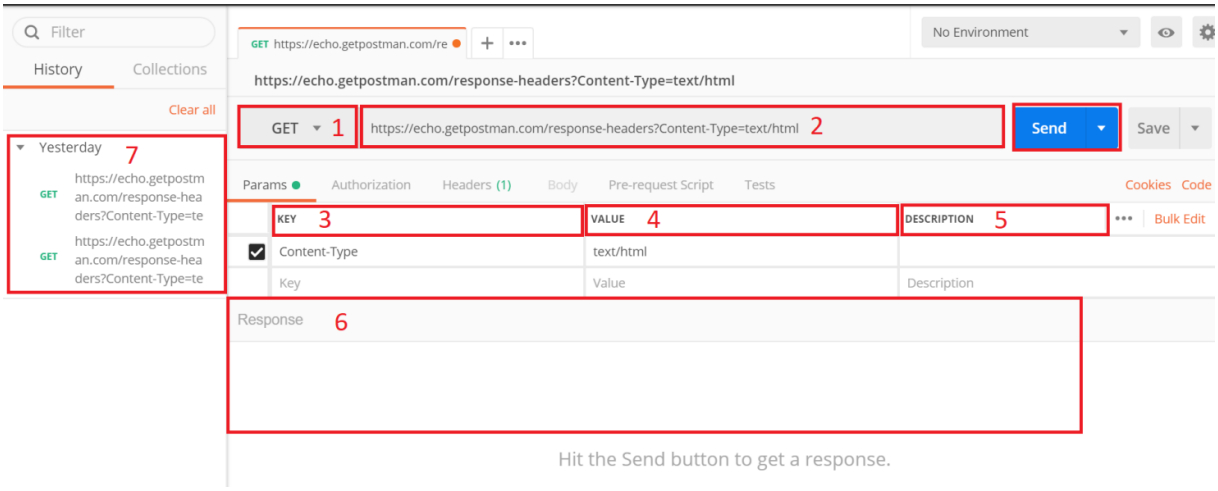
http://cms.softwarehouse.pl/Services/CustomerService.svc/DeleteAdres?id=955f7835-a8f7-408f-8aad-dc40856c31e0 .

W wyniku takiej operacji użytkownik powinien otrzymać status żądania o wartości 200 jeśli element może zostać usunięty. To oznacza, że metoda została pomyślnie wywołana i wykonała żądaną akcję. Podczas testów metod typu DELETE (w tym przypadku GET) należy wykonać testy negatywne, które sprawdzą, czy wprowadzona została odpowiednia walidacja i czy nie zostanie usunięty niedozwolony obiekt z bazy.

# 3. Testowanie API za pomocą narzędzi

Rezultat wywołanej metody zdefiniowanej jako GET można wyświetlić w przeglądarce. Natomiast metody POST-owe wymagają przeznaczonych narzędzi. Jednym z ciekawych narzędzi umożliwiających przeprowadzanie takich testów jest POSTMAN. Aplikacja jest obszernie opisana na stronie https://docs.postman-echo.com/ [2].

Aplikacja POSTMAN prezentuje się następująco:




Zawiera funkcjonalności oraz informacje niezbędne do przetestowania poprawności działania web serwisów:

a) Pole 1 służy do wyboru typu metody (GET, POST, PUT, DELETE).
b) Pole 2 to adres URL metody, którą należy przetestować. Wywołujemy ją niebieskim przyciskiem z etykietą *Send.*
c) Pole 3 to nazwa kolumny, w której wpisywane mogą być parametry, które są wymagane podczas testów API, np. Keyidentifier, Key lub inne klucze przy symulacji testów uwierzytelnienia lub zalogowanego użytkownika. Listę takich wymaganych parametrów osoba testująca powinna otrzymać od osoby tworzącej daną metodę.
d) Pole 4 przechowuje wartości parametrów z kolumny 3.
e) Pole 5 zawiera opis parametrów, nie wpływa na wynik testów.
f) Pole 6 prezentuje odpowiedź dla wykonanej metody oraz status odpowiedzi.
g) Pole 7 to historia wywoływanych metod. Przydatna jest, jeśli nie ma zapisanych metod w postaci arkusza testów.

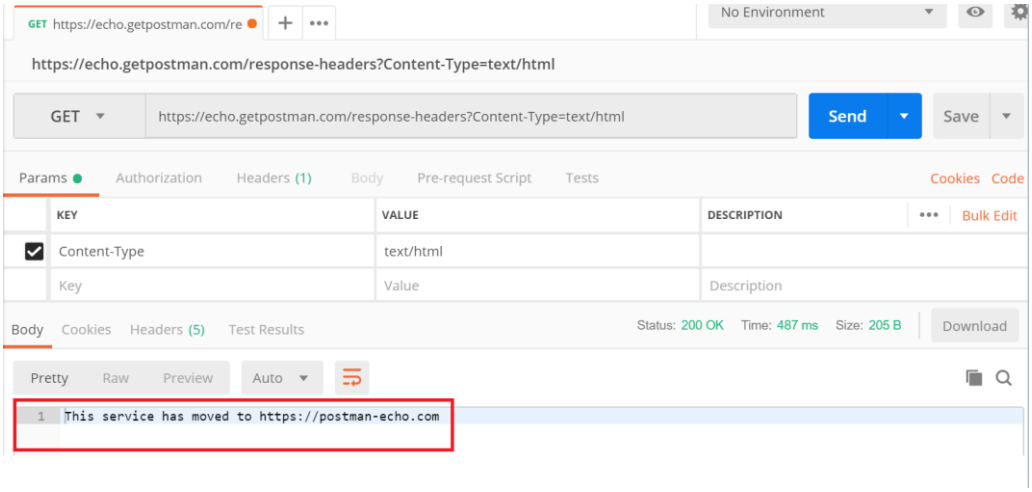
Postać odpowiedzi może być zdefiniowana w dowolny sposób. Może być pojedynczym zdaniem, listą parametrów z wartościami, może też zawierać tylko status.

**Przykład 4.**

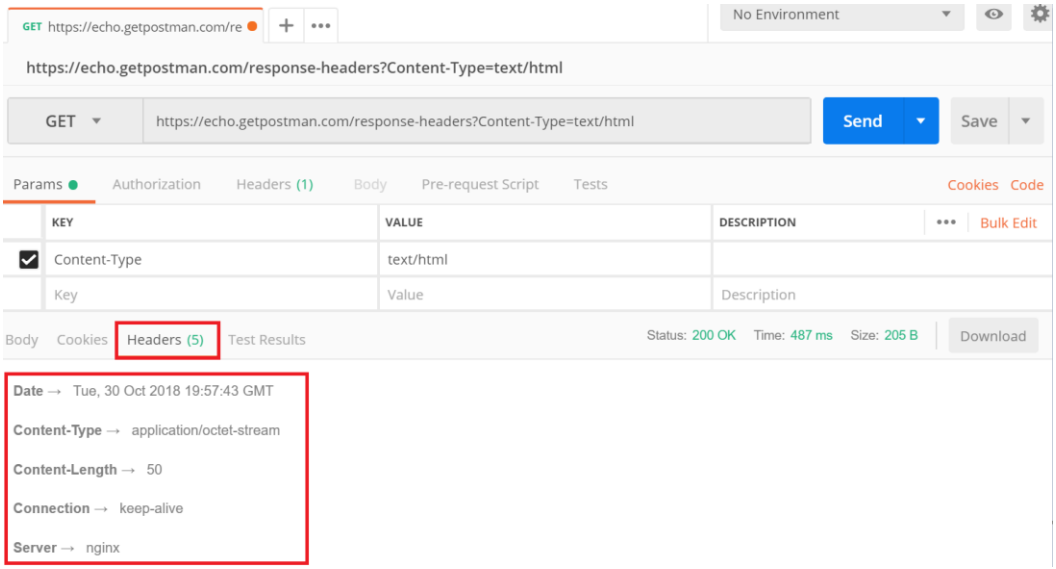
Wywołanie niżej przedstawionej metody

https://echo.getpostman.com/response-headers?Content-Type=text/html

daje następującą odpowiedź:




Taka odpowiedź jest w tym przypadku prawidłowa. Odpowiedź można obejrzeć w innej postaci, wybierając w sekcji 6 zakładkę *Headers,* jak na obrazku niżej:

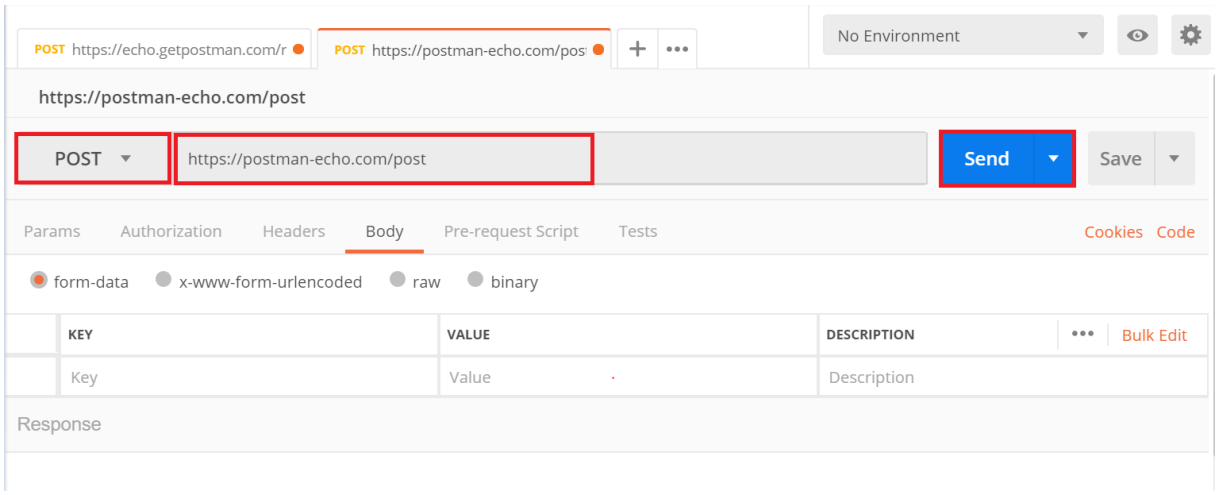



**Przykład 5.**

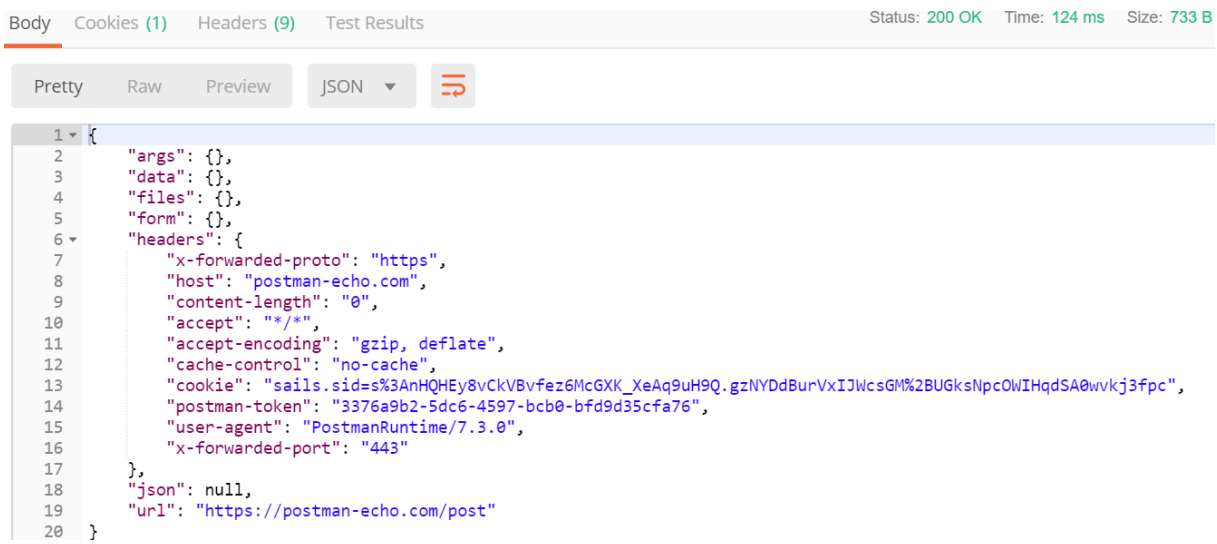
W tym przykładzie przedstawiona będzie metoda, która jest POST-em. Adres URL metody to https://postman-echo.com/post (post w tym przykładzie to nazwa metody). Ta metoda nie przyjmuje parametrów w adresie URL. Może natomiast, ale nie musi, otrzymać wprowadzone ręcznie parametry w formacie JSON. Aby przetestować tą metodę w aplikacji POSTMAN, należy:

a) wkleić adres metody w przeznaczone do tego pole,
b) wybrać typ POST,
c) kliknąć Send,

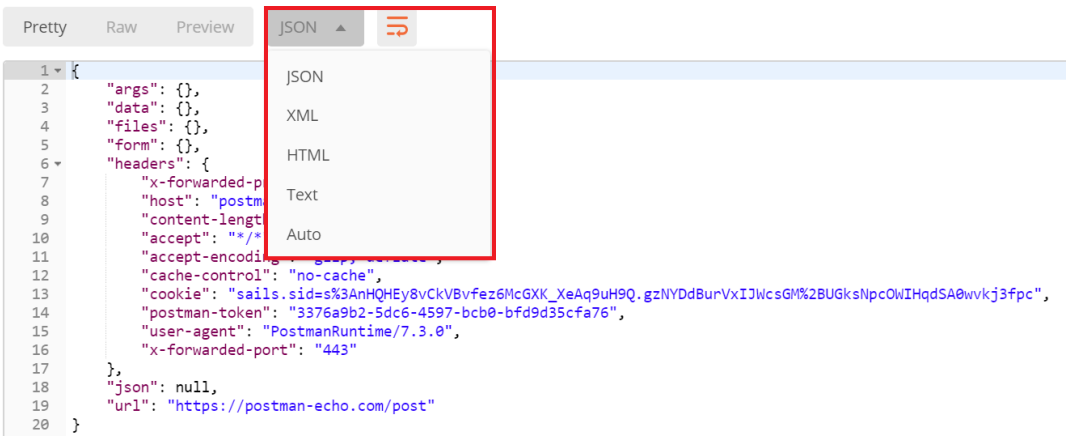
jak niżej na obrazku:



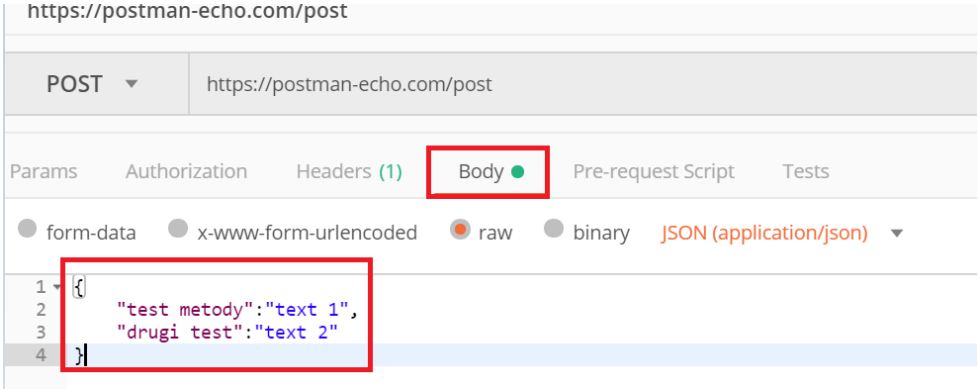

Odpowiedź dla wysłanego żądania będzie prezentowana tak, jak pokazuje niżej umieszczony obrazek.



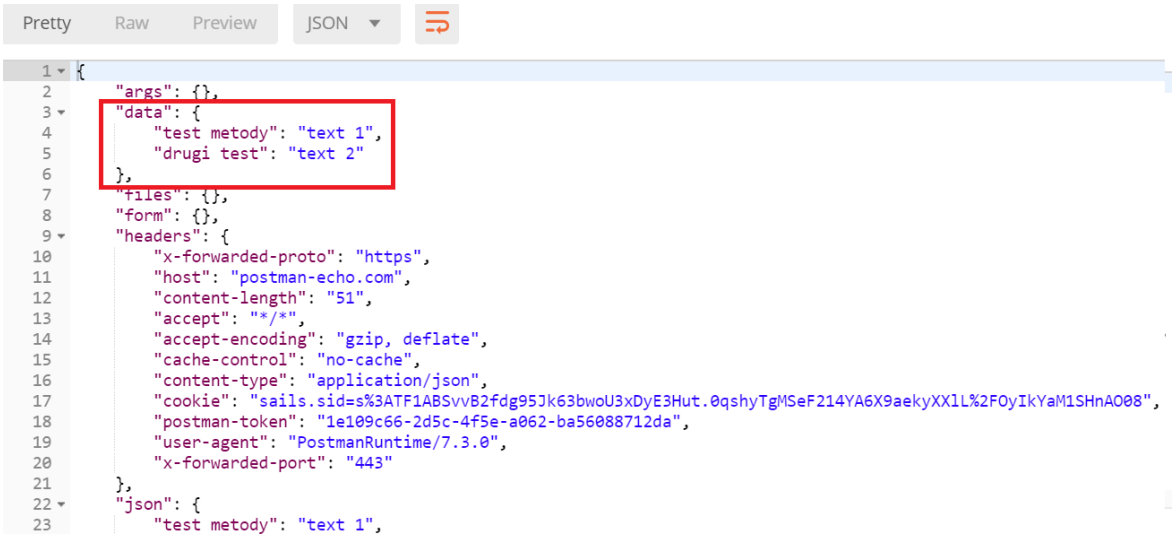

W aplikacji POSTMAN użytkownik ma możliwość zmienić prezentowanie danych na taki format, jaki jest dostępny, co pokazuje kolejny obrazek:



W tym przykładzie najlepszym formatem dla odpowiedzi jest JSON, ponieważ tak są ułożone dane w sekcji *Response.* Odpowiedź zawiera obiekt "data": {}, który póki co jest pusty. Jeśli jednak do ciała metody w aplikacji wprowadzimy dane w formacie JSON np.:

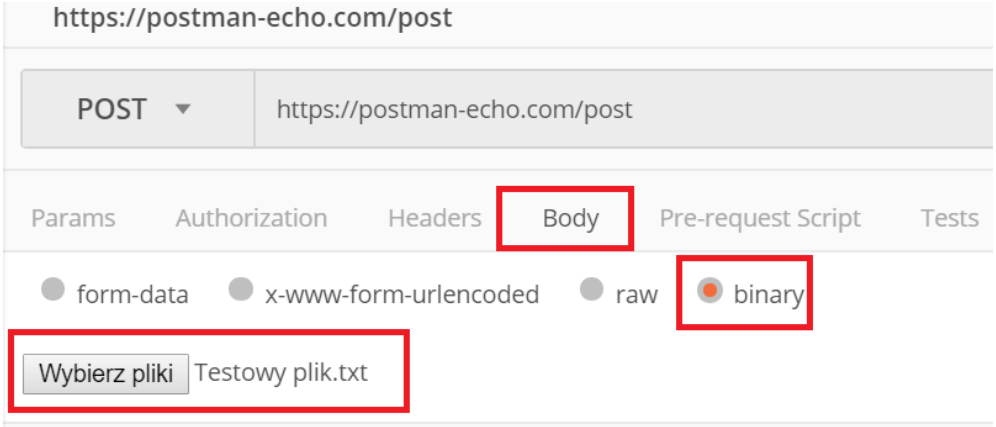



to rezultatem testu będzie odpowiedź:

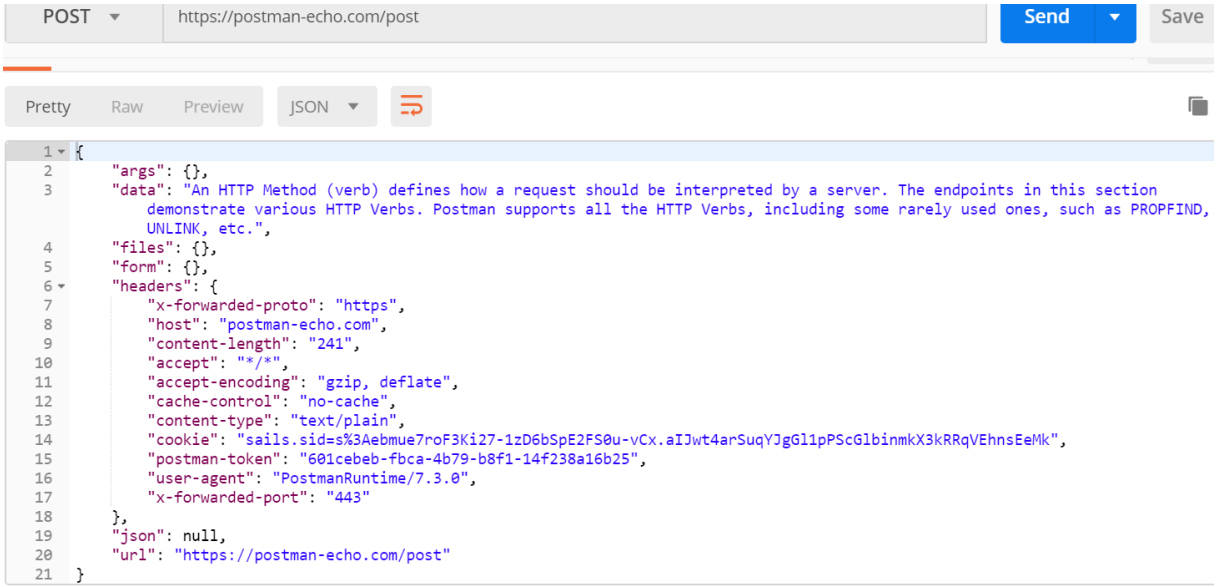



Taki jest w tym przypadku oczekiwany rezultat, a więc test przebiegł pomyślnie. Istnieją również sytuacje, w których należy dołączyć plik podczas testów POST-owych metod. POSTMAN zawiera taką funkcjonalność. Jeśli dołączymy do metody *post* zwykły plik *.txt z tekstem:

*An HTTP Method (verb) defines how a request should be interpreted by a server. The endpoints in this section demonstrate various HTTP Verbs. Postman supports all the HTTP Verbs, including some rarely used ones, such as PROPFIND, UNLINK, etc.*




to rezultatem wywołania metody będzie odpowiedź:




## 4. Podsumowanie

Artykuł powstał w celu wprowadzenia początkujących testerów w obszar weryfikowania jakości web serwisów oraz API. Początki na ogół bywają trudne, a sporo testerów nie posiada wykształcenia informatycznego czy programistycznego. Postawienie początkującego testera przed testowaniem API wymaga wdrożenia go w ten obszar. Nawet, jeśli ktoś ma doświadczenie a nigdy nie testował API, to również musi przyswoić tą wiedzę. Z taką myślą został napisany niniejszy artykuł.

## 5. Literatura

[1] https://pl.wikipedia.org/wiki/Interfejs_programowania_aplikacji

[2] https://docs.postman-echo.com/

[3] Sylabus ISTQB Techniczny Analityk Testów

[4] http://www.samouczekprogramisty.pl/protokol-http/#odpowied%C5%BA-http

## Autor

Marek Żukowicz jest absolwentem matematyki na Uniwersytecie Rzeszowskim. Obecnie pracuje jako tester. Jego zainteresowania skupiają się wokół testowania, matematyki, zastosowania algorytmów ewolucyjnych oraz zastosowania matematyki w procesie testowania. Interesuje się również muzyką, grą na akordeonie oraz na perkusji.