

REST API 10 RZECZY

KTÓRE MUSISZ PRZETESTOWAĆ

Z PRZYKŁADAMI W POSTMANIE





WPROWADZENIE

Cześć!

Tu Bartek z AkademiaQA.pl

Jest mi ogromnie miło, że chcesz poświęcić chwilę swojego cennego czasu, aby przeczytać ten artykuł.

Przygotowałem go w formie 10 przykładów, wraz ze screenshotami z Postmana, aby dał Ci jak największą wartość.

Często dostaję pytanie: **"Co powinienem/powinnam przetestować w REST API?"**

Mam nadzieję, że przykłady, które przygotowałem odpowiedzą na to pytanie i pozwolą Ci spojrzeć na testy API z innej perspektywy.

Nie przedłużając, zapraszam Cię do pierwszej rzeczy, którą trzeba sprawdzić testując REST API.

~bartek

1. FIRST, CHECK THE SPEC!

API to **kontrakt** pomiędzy klientem a serwerem lub pomiędzy dwiema aplikacjami.

Zanim przejdziemy do sprawdzenia funkcjonalności API, warto sprawdzić, czy kontrakt jest poprawny.

W pierwszym kroku, utworzymy wymagany schemat odpowiedzi i porównamy go z odpowiedzią, którą otrzymamy z API.

W naszym przypadku, skorzystamy z serwisu

<https://jsonplaceholder.typicode.com>

i utworzymy schemat odpowiedzi dla dowolnego zasobu **post** z endpointu **/posts**

Zasób **post** składa się z czterech pól:

1. **userId** – liczba całkowita
2. **id** – liczba całkowita
3. **title** – tekst
4. **body** – tekst

W teście sprawdzimy czy odpowiedź, którą otrzymamy po wysłaniu requestu **GET** na endpoint:

<https://jsonplaceholder.typicode.com/posts/1>

- jest obiektem
- czy wszystkie pola mają poprawny typ
- czy wszystkie wymagane pola są obecne w odpowiedzi.

Na obrazku poniżej, pokazałem jak w Postmanie utworzyć wymagany schemat odpowiedzi oraz jak napisać test, który sprawdza, czy kontrakt jest poprawny.

Request i response:

GET  https://jsonplaceholder.typicode.com/posts/1

Params Authorization Headers (7) Body Pre-request Script **Tests ** Settings

```
1  const schema = {
2    "type": "object",
3    "properties": {
4      "userId": { "type": "integer" },
5      "id": { "type": "integer" },
6      "title": { "type": "string" },
7      "body": { "type": "string" }
8    },
9    "required": ["userId", "id", "title", "body"]
10  };
11
12  pm.test("Validate response schema", () => {
13    pm.response.to.have.jsonSchema(schema);
14  });
```

Body Cookies Headers (25) Test Results (1/1)

Pretty Raw Preview Visualize JSON  

```
1  {
2    "userId": 1,
3    "id": 1,
4    "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
5    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehend
6      rem eveniet architecto"
```

2. KOD ODPOWIEDZI

Kolejną rzeczą, którą **trzeba** sprawdzić podczas testowania REST API są kody odpowiedzi.

Za każdym razem, kiedy wyślemy request do API, w odpowiedzi otrzymamy ciało odpowiedzi, nagłówki oraz kod odpowiedzi.

Kod odpowiedzi (status code) informuje nas czy request został wysłany poprawnie czy może podczas wysyłania requestu wystąpił jakiś błąd.

W REST API mamy **5 grup** kodów odpowiedzi:

1xx - odpowiedzi informacyjne

2xx - sukces

3xx - przeniesienia

4xx - błąd po stronie klienta

5xx - błąd po stronie serwera

Poniżej przykład testu, który sprawdza kod odpowiedzi **200 OK**.

200 OK informuje nas, że request został wysłany poprawnie.

Korzystając ze snippetów wbudowanych w Postmana, możemy utworzyć taki test jednym kliknięciem.

Z listy dostępnych snippetów wybieramy:

Status code: Code is 200

Test scripts are written in JavaScript, and are run after the response is received. >

[Learn more about tests scripts](#)

SNIPPETS

Status code: Code is 200

Response body: Contains string

Response body: JSON value check

Response body: Is equal to a string

Response headers: Content-Type header check

Response time is less than 200ms

Status code: Successful POST request

Params Authorization Headers (7) Body Pre-request Script Tests ● Settings

```
1 pm.test("Status code is 200", function () {  
2   pm.response.to.have.status(200);  
3 });
```

3. DANE W ODPOWIEDZI

Sprawdzenie poprawności danych, które otrzymasz w odpowiedzi, to coś nad czym spędzisz najwięcej czasu.

Czytając, tworząc czy edytując zasoby, musisz potwierdzić, że dane, które zwraca API są poprawne.

W kursie "**Szkoła Testowania Backendu edycja II**"

(<https://testowaniebackendu.pl/>) pokazuję biblioteki, z których korzysta Postman oraz kilkanaście przykładów testów gotowych do użycia.

Poniżej pokażę Ci dwie najczęściej używane metody do sprawdzenia danych w odpowiedzi:

- ***eq***, która porównuje dwie wartości
- ***include***, która sprawdza, czy wartość zawiera inną wartość, np. czy długi tekst zawiera jakiś fragment.

W pierwszym kroku tworzymy stałą ***response***, do której przypisujemy odpowiedź w formacie **JSON**.

W kolejnych krokach, w teście sprawdzamy czy pole ***response.id*** czyli id danego postu jest równe **1**.

W kolejnych dwóch liniijkach sprawdzamy czy pola ***response.title*** oraz ***response.body*** zawierają fragment tekstu, który nas interesuje.

▼ <https://jsonplaceholder.typicode.com/posts/1>

Authorization Headers (7) Body Pre-request Script Tests ● Settings

```
const response = pm.response.json();

pm.test("Validate response data", () => {
  pm.expect(response.id).to.eql(1);
  pm.expect(response.title).to.include('sunt aut facere repellat provident');
  pm.expect(response.body).to.include('recusandae consequuntur expedita et');
});
```


4. WYSZUKIWANIE

Kontynuując temat sprawdzania danych w odpowiedzi, warto przetestować wyszukiwanie danych.

Jeżeli w adresie requestu, który wysyłamy do API, dodamy query params, czyli parametry podawane po znaku zapytania. W odpowiedzi powinniśmy dostać tylko zasoby spełniające kryteria wyszukiwania.

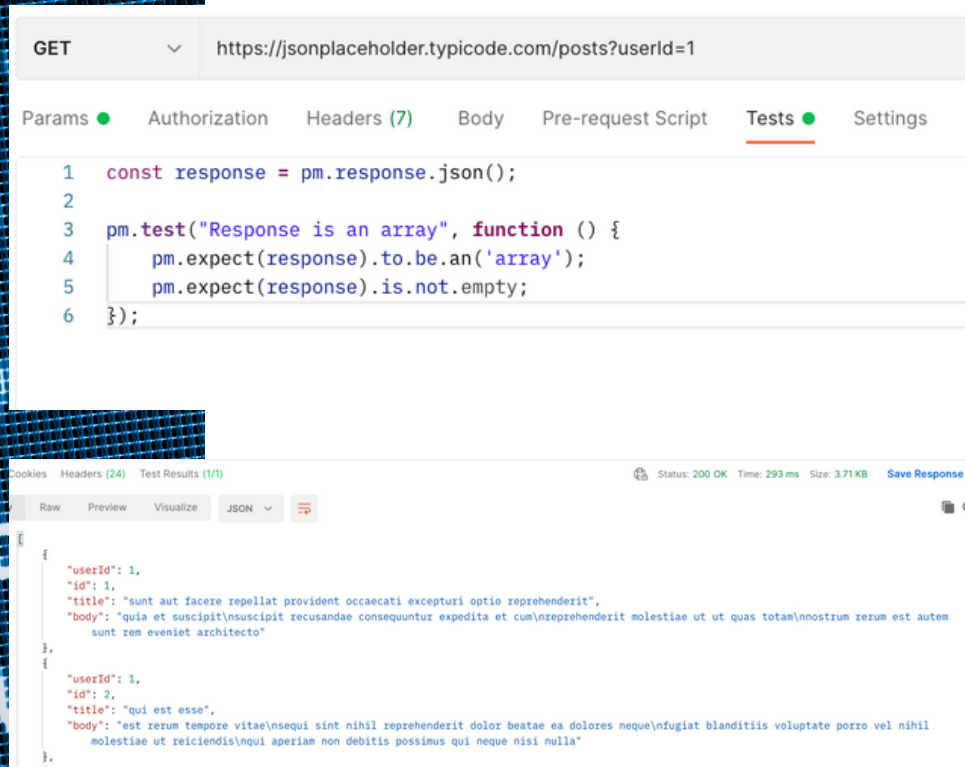
W przykładzie poniżej, z wszystkich dostępnych postów, chciałbym wyszukać tylko te, dla których pole **userId** jest ustawione na 1.

Request **GET** wysyłam na adres:
<https://jsonplaceholder.typicode.com/posts?userId=1>

W teście sprawdzę, czy odpowiedź zwrócona przez API jest tablicą, która nie jest pusta.

Zaczynam od utworzenia zmiennej **response**, do której przypisuję odpowiedź w formacie **JSON**.

Następnie sprawdzam czy otrzymana odpowiedź jest tablicą **to.be.an('array')** oraz czy otrzymana tablica nie jest pusta **is.not.empty**.



```
GET https://jsonplaceholder.typicode.com/posts?userId=1
```

Params Authorization Headers (7) Body Pre-request Script **Tests** Settings

```
1 const response = pm.response.json();
2
3 pm.test("Response is an array", function () {
4   pm.expect(response).to.be.an('array');
5   pm.expect(response).is.not.empty;
6 });
```

Cookies Headers (24) Test Results (1/1) Status: 200 OK Time: 293 ms Size: 3.71 KB Save Response

Raw Preview Visualize JSON

```
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
},
{
  "userId": 1,
  "id": 2,
  "title": "qui est esse",
  "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
},
}
```

5. NAGŁÓWKI

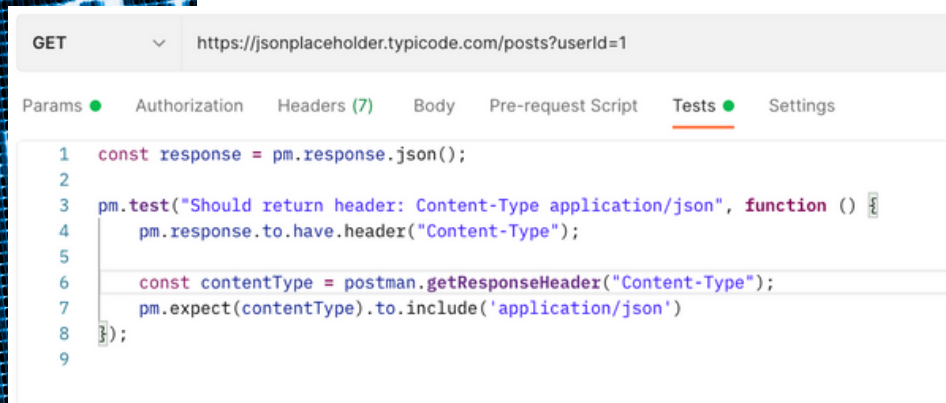
Kolejną rzeczą, którą warto sprawdzić podczas testowania REST API są **nagłówki** wysyłane w odpowiedzi.

Nagłówki mogą zawierać takie informacje jak:

- data wysłania odpowiedzi
- format odpowiedzi: text/html/json
- informacja o plikach Cookies
- kodowanie odpowiedzi
- informacje na temat serwera, który wysłał odpowiedź

Poniżej pokażę Ci jak sprawdzić czy odpowiedź zawiera dany nagłówek oraz jak sprawdzić wartość danego nagłówka.

Sprawdzimy czy odpowiedź zawiera nagłówek "**Content-Type**" oraz czy wartość nagłówka "Content-Type" jest ustawiona na "**application/json**".

A screenshot of the Postman interface showing the 'Tests' tab for a GET request to 'https://jsonplaceholder.typicode.com/posts?userId=1'. The test script contains two assertions: one to check if the 'Content-Type' header is present and another to verify its value is 'application/json'.

```
1 const response = pm.response.json();
2
3 pm.test("Should return header: Content-Type application/json", function () {
4     pm.response.to.have.header("Content-Type");
5
6     const contentType = postman.getResponseHeader("Content-Type");
7     pm.expect(contentType).to.include('application/json')
8 });
9
```

A screenshot of the Postman interface showing the 'Headers' tab. It displays a table of response headers for the same request. The status is 200.

KEY	VALUE
Date ⓘ	Fri, 26 Aug 2022 07:40:26 GMT
Content-Type ⓘ	application/json; charset=utf-8
Transfer-Encoding ⓘ	chunked
Connection ⓘ	keep-alive
X-Powered-By ⓘ	Express
X-Ratelimit-Limit ⓘ	1000
X-Ratelimit-Remaining ⓘ	998

6. STAN APLIKACJI

Sprawdzenie kodu odpowiedzi to niezbędne minimum do potwierdzenia tego czy operacja została wykonana poprawnie.

Ale warto iść o krok dalej!

Wyobraź sobie, że za pomocą API stworzysz nowego użytkownika w systemie.

Wysyłasz request **POST** z danymi użytkownika, w response dostajesz kod odpowiedzi **201 CREATED** i wszystko wygląda poprawnie.

Kiedy próbujesz się zalogować do systemu podając dane tego użytkownika, dostajesz błąd z informacją, że **taki użytkownik nie istnieje!**

Sprawdzasz w bazie danych – **NIE MA!**

Przez pół dnia zastanawiasz się, gdzie jest błąd, aż nagle okazuje się, że programista, który nad tym pracuje jest na urlopie i zdążył tylko dodać

kod odpowiedzi **201 CREATED**, ale nie zdążył już zapisać danych użytkownika do bazy.

PS. Jest to prawdziwy przykład z jednej firmy, z którą miałem przyjemność współpracować ;)

Żeby upewnić się, że aplikacja znajduje się w poprawnym stanie warto to zweryfikować kolejnym requestem, który przeczyta dane z bazy danych.

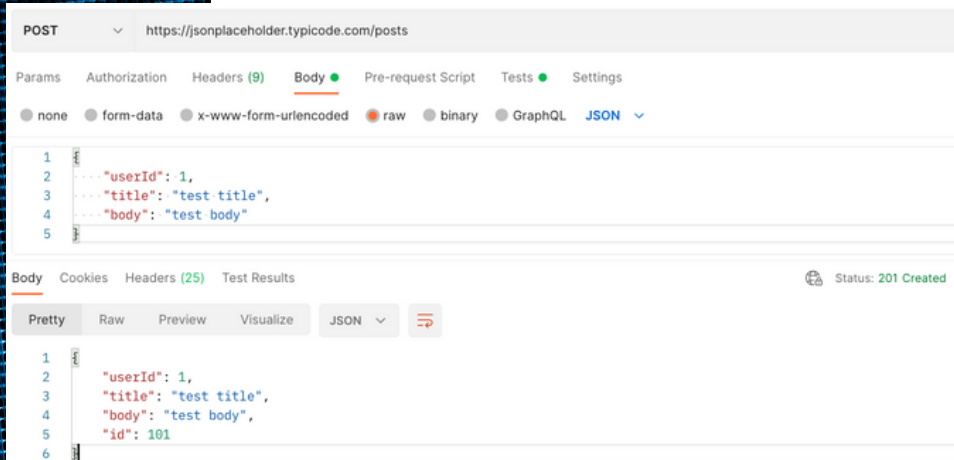
KROK 1 – Wysyłasz request
POST/PUT/PATCH/DELETE

KROK 2 – Wysyłasz request **GET** w celu zweryfikowania zmian w aplikacji.

Poniżej zobaczysz podobny przykład pochodzący z aplikacji JsonPlaceholder.

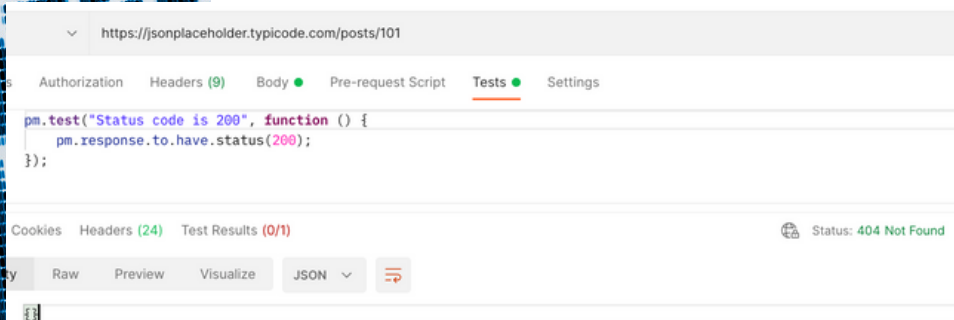
Różnica jest taka, że JsonPlaceholder to testowe, **fakowe** API, które oficjalnie informuje o tym, że nie zapisuje danych do bazy.

1. Request **POST** do utworzenia nowego zasobu:



2. Request **GET** w celu odczytania danych nowego zasobu.

Jak widzisz, pomimo tego, że **POST** zwrócił **201 CREATED** to **GET** zwraca **404 NOT FOUND**.



7. WALIDACJA DANYCH

Podczas tworzenia lub edycji zasobów, zawsze warto sprawdzić poprawność danych, które wysyłamy do API, oraz to jak API zareaguje na błędne dane.

W pierwszym punkcie sprawdziliśmy kontrakt dla zasobu **post**.

Wiemy, że pole **title** jest typu **string**, czyli w tym polu możemy przesyłać tylko tekst.

Co się stanie, jeżeli zamiast tekstu prześlemy liczbę, pusty string lub wartość null?

Warto sprawdzić czy API odpowie poprawnie zwracając komunikat błędu czy może wysłanie błędnych danych spowoduje krytyczny błąd po stronie API.

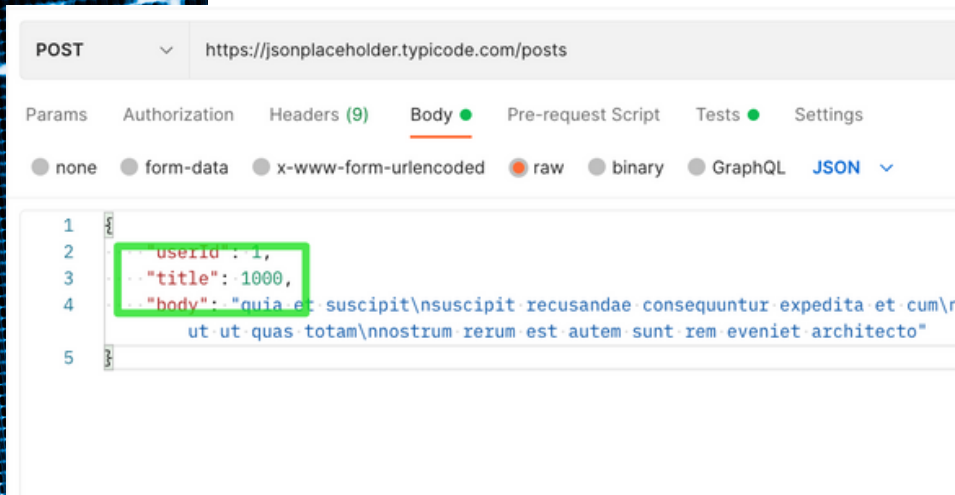
Kolejnym ciekawym przykładem jest sprawdzenie poprawności adresu email.

Jeżeli w requeście tworzysz lub edytujesz dane użytkownika, to warto sprawdzić czy API poprawnie reaguje na błędne adresy email.

Zasady budowy adresów e-mail są zawarte w standardach RFC. Można przyjąć, że poprawny adres składa się z:

- **dużych i małych** liter alfabetu łacińskiego [A-Za-z],
- **cyfr** [0-9],
- znaku **kropki** (co najmniej jednego),
- znaku **"@"** (dokładnie jednego),
- znaków **myślnika i podkreśleń** dolnych.

Poniżej znajdziesz przykłady requestów zawierających błędne dane jak również kilka przykładów poprawnych oraz błędnych adresów email.



```
1  const validEmails = [
2    'bartek@akademiaqa.pl',
3    'bartek.kita@akademiaqa.pl',
4    'bartek@akademiaqa.com.pl',
5    'Bartek@akademiaqa.pl',
6    'bartek123@akademiaqa123.pl',
7    'bartek_123@akademiaqa.pl',
8    '123bartek@akademiaqa.pl'
9  ]
10
11  const invalidEmails = [
12    '-bartek@akademiaqa.pl',
13    '_bartek.kita@akademiaqa.pl',
14    'bartek.akademiaqa.pl',
15    'bartek@akademiaqa',
16    'bartek@-akademiaqa.pl',
17    'bartek@_akademiaqa.pl'
18  ]
```

8. CZAS ODPOWIEDZI

Testując REST API warto sprawdzić nie tylko, jak zachowuje się nasze API, ale również ile czasu zajmuje wysłanie odpowiedzi.

Bardzo często klienci, dla których tworzymy i testujemy API, określają maksymalne czasy odpowiedzi dla wybranych requestów.

Postman umożliwia nam dodanie takiego sprawdzenia za pomocą jednego kliknięcia.

Ponownie posłużymy się wbudowanymi snippetami. Tym razem, z listy gotowych przykładów wybieramy:
Response time is less than 200ms.

Następnie w teście możemy określić jaki czas odpowiedzi nas interesuje.

SNIPPETS

Status code: Code is 200

Response body: Contains string

Response body: JSON value check

Response body: Is equal to a string

Response headers: Content-Type header check

Response time is less than 200ms

Status code: Successful POST request

Status code: Code name has string

Response body: Convert XML body to a JSON Object

Use Tiny Validator for JSON data

GET https://jsonplaceholder.typicode.com/posts

Params Authorization Headers (7) Body Pre-request Script Tests Settings

```
1 pm.test("Response time is less than 200ms", function () {  
2   pm.expect(pm.response.responseTime).to.be.below(200);  
3 });
```

Body Cookies Headers (25) Test Results (1/1)

200 OK 66 ms 27.95 KB

Pretty

Raw

Preview

Visualize

JSON

9. CRUD

CRUD to podstawowe operacje, które możemy wykonać na zasobie.

C - create (utworzenie)

R - read (odczyt)

U - update (edycja)

D - delete (usunięcie)

Wyobraź sobie, że testujesz pojedynczy endpoint, np. dla zasobu **bug**.

Dobłą praktyką jest przetestowanie wszystkich operacji na danym zasobie.

Chcemy mieć pewność, że będziemy mogli utworzyć, przeczytać, edytować oraz usunąć błąd.

Jeżeli dokumentacja API mówi, że któraś operacja nie jest dozwolona, np. usunięcie błędu, to i tak warto sprawdzić co się stanie, jeżeli spróbujemy usunąć istniejący zasób oraz jak API zareaguje na próbę usunięcia zasobu, który nie istnieje.

API powinno zwrócić informację, że dana metoda jest niedozwolona.

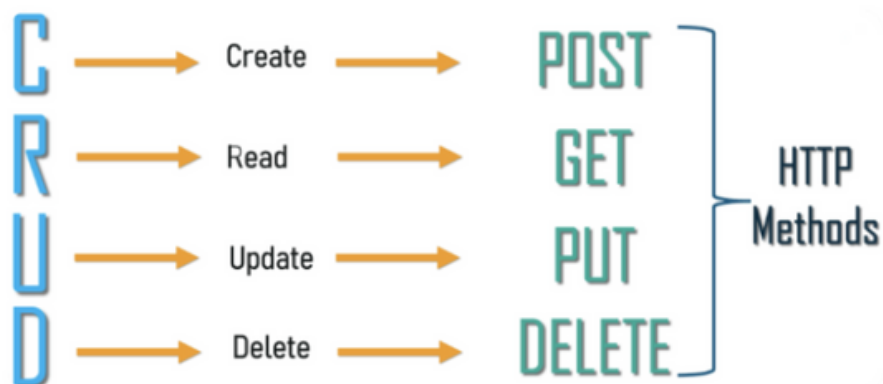
Dodatkową rzeczą, którą warto sprawdzić jest edycja zasobu.

W REST API mamy dwie możliwości edycji danych.

1. Wysłanie requestu **PUT** - nadpisanie całego obiektu.
2. Wysłanie requestu **PATCH** - częściowa edycja obiektu.

Jeżeli dokumentacja mówi, że możemy edytować zasób tylko za pomocą jednej metody to warto sprawdzić, czy API poprawnie obsługuje drugą metodę.

Poniżej pokazuję przełożenie **CRUDa** na metody **HTTP** oraz przykład testów **CRUD** dla zasobu bug.



ScrumTeam

> GET

✓ POST

POST Create new bug

DEL Delete existing bug Copy

> PUT

> PATCH

✓ DELETE

POST Create new bug

DEL Delete existing bug

DEL Delete non existing bug

10. SCENARIUSZ E2E

Ostatnią rzeczą, którą chciałbym Ci pokazać, i którą warto przetestować są **scenariusze end to end**.

Testy REST API to nie tylko pojedyncze zasoby i walidacja danych.

Możemy też tworzyć całe scenariusze, które będą wysyłały requesty do różnych endpointów i sprawdzały logikę biznesową po API.

To rozwiązanie ma jedną **OGROMNĄ** zaletę.

Testy po API są dużo szybsze niż te same scenariusze zautomatyzowane na frontendzie.

Na kolejnej stronie pokazuję przykład takiego scenariusze dla aplikacji ClickUp.

Przykład pochodzi z kursu "**Szkoła Testowania Backendu II**".

1. Tworzymy przestrzeń (space)
2. Tworzymy folder
3. Tworzymy zadanie
4. Edytujemy zadanie
5. Zamykamy zadanie
6. Usuwamy przestrzeń (space)

+

☰

...

> AuthDemo

> ClickUp API V2

▼ ClickUp E2E

POST Create a Space

POST Create Folderless List

POST Create task

PUT Update task name and descript...

PUT Close task

DEL Delete a Space

> ClickupDemo

> JavaScript

> JsonplaceholderDemo

> PostmanTests

> ScrumTeam

> Trello Create a Board

> Trello E2E

> Variables

ClickUp E2E Clickup Prod, j

All Tests

Passed (10)

1

POST Create task {{url}}/lis

Pass

Status code

Pass

Should retur

PUT Update task name and

Pass

Status code

Pass

Should retur

PUT Close task {{uri}}/task

Pass

Status code

Pass

Should retur

DELETE Delete a Space {{

Pass

Status code

AkademiaQA.pl - kursy i szkolenia dla testerów

PODZIĘKOWANIE

Jeżeli to czytasz, to znaczy, że wspólnie dotarliśmy do końca 😊

To już wszystko co dla Ciebie przygotowałem.

Jak możesz się spodziewać, to tylko szczyt góry lodowej zwanej testowanie REST API.

Jeżeli temat testowania REST API jest dla Ciebie interesujący i chcesz się rozwijać w tym kierunku to zachęcam Cię do zapoznania się z moim kursem, w którym uczę jak zacząć testować REST API od samych podstaw.

Wszystkie informacje o kursie znajdziesz na stronie:

<https://testowaniebackendu.pl/>

Będzie mi bardzo miło, jeżeli poświęcisz 30 sekund i wyślesz mi opinię, dosłownie 2-3 zdania, na temat tego ebooka.

Mój email: **bartek@akademiaqa.pl**

Do zobaczenia gdzieś w internecie.
~bartek