JavaScript

fetch API - CRUD w REST API

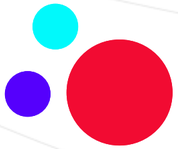
infoShareAcademy.com

HELLO

Dariusz Sibik

Senior Frontend Software Engineer (Spyrosoft)

infoShare ACADEMY



Zasoby do samodzielnego zdobywania wiedzy

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

https://developer.mozilla.org/en-US/docs/Glossary/CRUD

https://iq.opengenus.org/intro-to-fetch-api/

https://developer.mozilla.org/en-US/docs/Glossary/REST

https://bykowski.pl/rest-api-efektywna-droga-do-zrozumienia/

https://kobietydokodu.pl/niezbednik-juniora-protokol-http/

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE

Przypomnienie - Promise

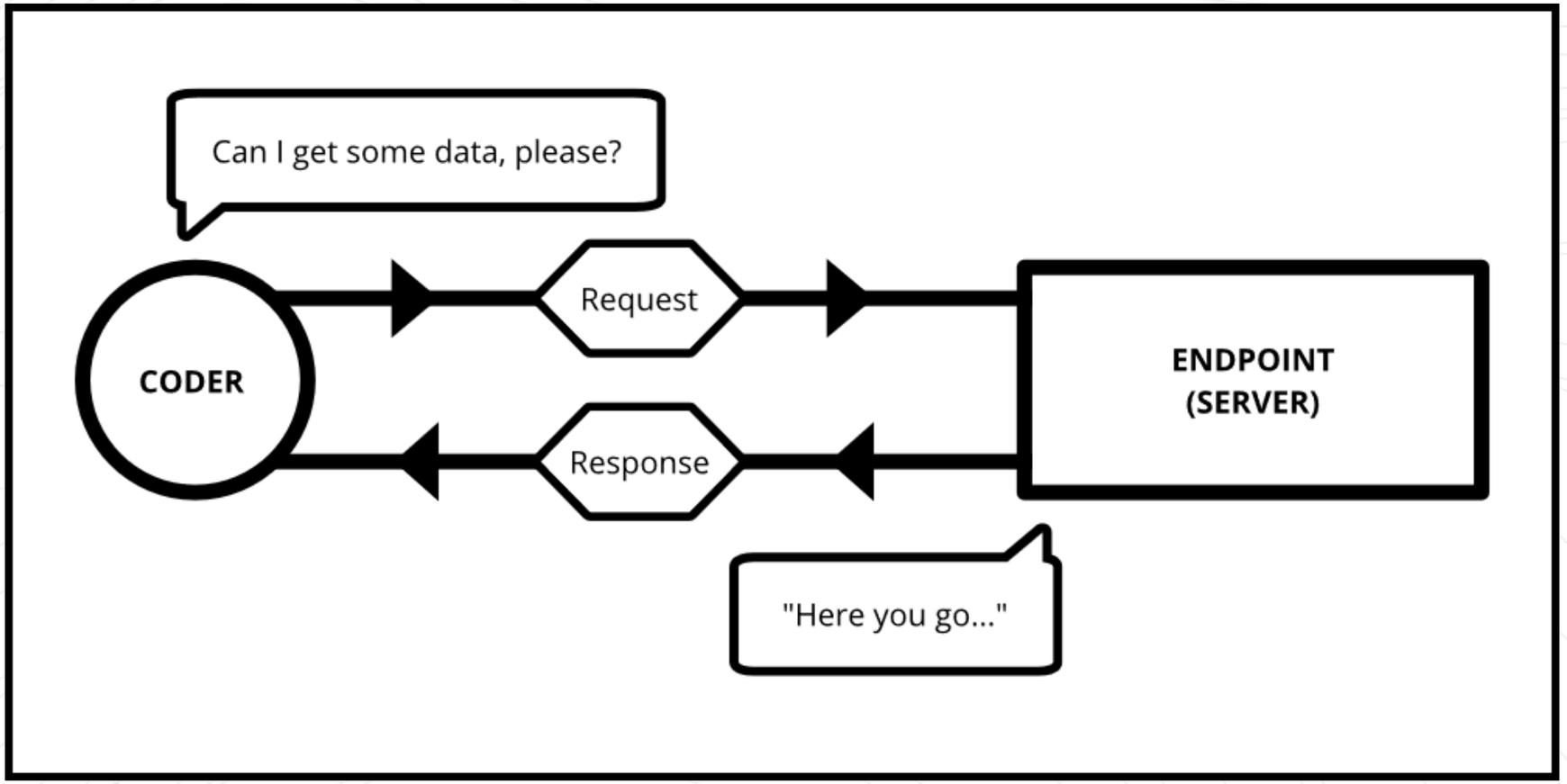
Promise - jako obiekt - może znajdować się w jednych z trzech stanów:

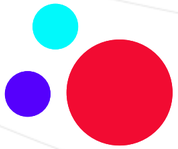
- Pending - cały czas coś trwa, asynchroniczna operacja jeszcze się nie zakończyła
- Fulfilled - operacja zakończona sukcesem
- Rejected - operacja zakończona porażką

Fetch API

- Pozwala na wykonywanie zapytań HTTP
- Jest dostarczany przez przeglądarkę (nie jest częścią języka JS)
- Zwraca Promise
- Operuje na generycznych typach Request i Response
- Pozwala anulować requesty za pomocą obiektu AbortSignal
- Nie wyrzuca błędów dla statusów HTTP > 400 (trzeba to zrobić manualnie)

Fetch API





Fetch API

```
fetch('https://jsonplaceholder.typicode.com/users/2')
    .then(function(res) {
        return res.json()    // zamienia na JSON
    })
    .then(function(data) {
        console.log(data);    // wyświetlamy dane
    })
    .catch(function(error) {
        console.log(err);    // wyświetlamy error
    });
```

Rest API

API (ang. Application Programming Interface) definiuje sposób komunikacji danego oprogramowania z innym np. aplikacji webowej z serwerem.

W przypadku API serwerowego określa reguły definiujące jak użytkownik może uzyskać dostęp do zasobów oraz w jakiej postaci je otrzymuje.

REST (ang. Representational State Transfer) to styl architektury definiujący kształt API.

Jest to standard przygotowywania API definiujący reguły jak takowe API powinno
powstać.

Celem REST API jest zaprojektowanie logicznego i wygodnego w użyciu API m.in.
określając strukturę endpointu.

CRUD

CRUD – cztery podstawowe funkcje w aplikacjach korzystających z pamięci trwałej, które umożliwiają zarządzanie nią.

Create - (POST)

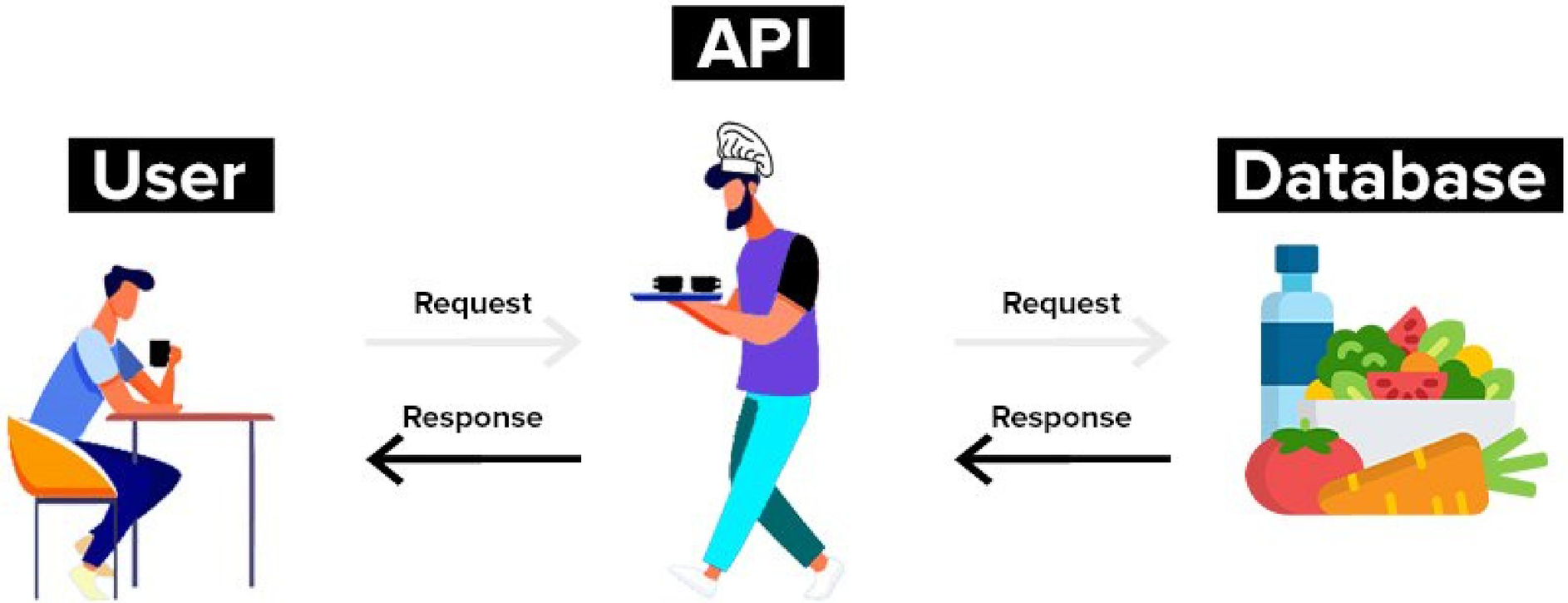
Read - (GET)

Update - (PUST, PATCH)

Delete - (DELETE)

infoShare ACADEMY

Rest API

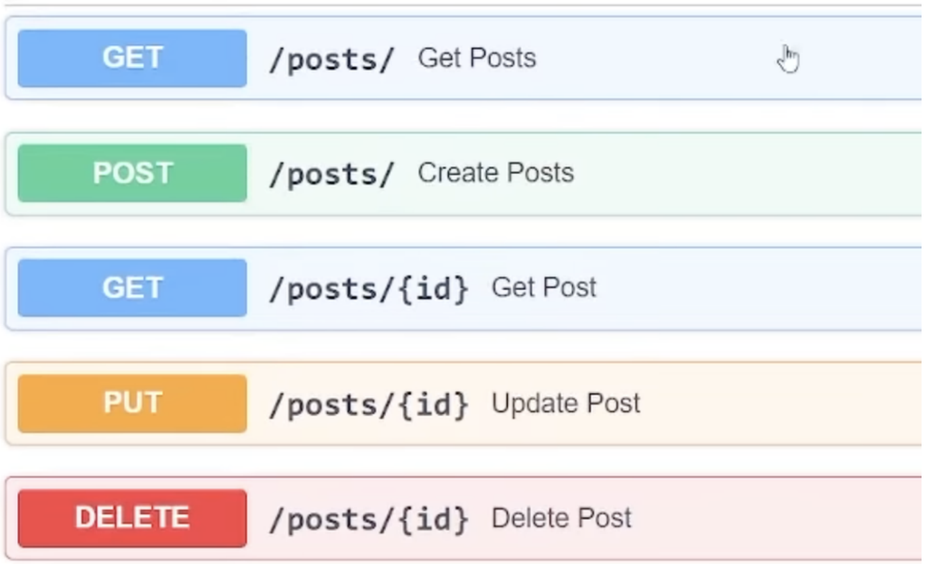


Endpoint

Punkt końcowy to jeden koniec kanału komunikacyjnego.

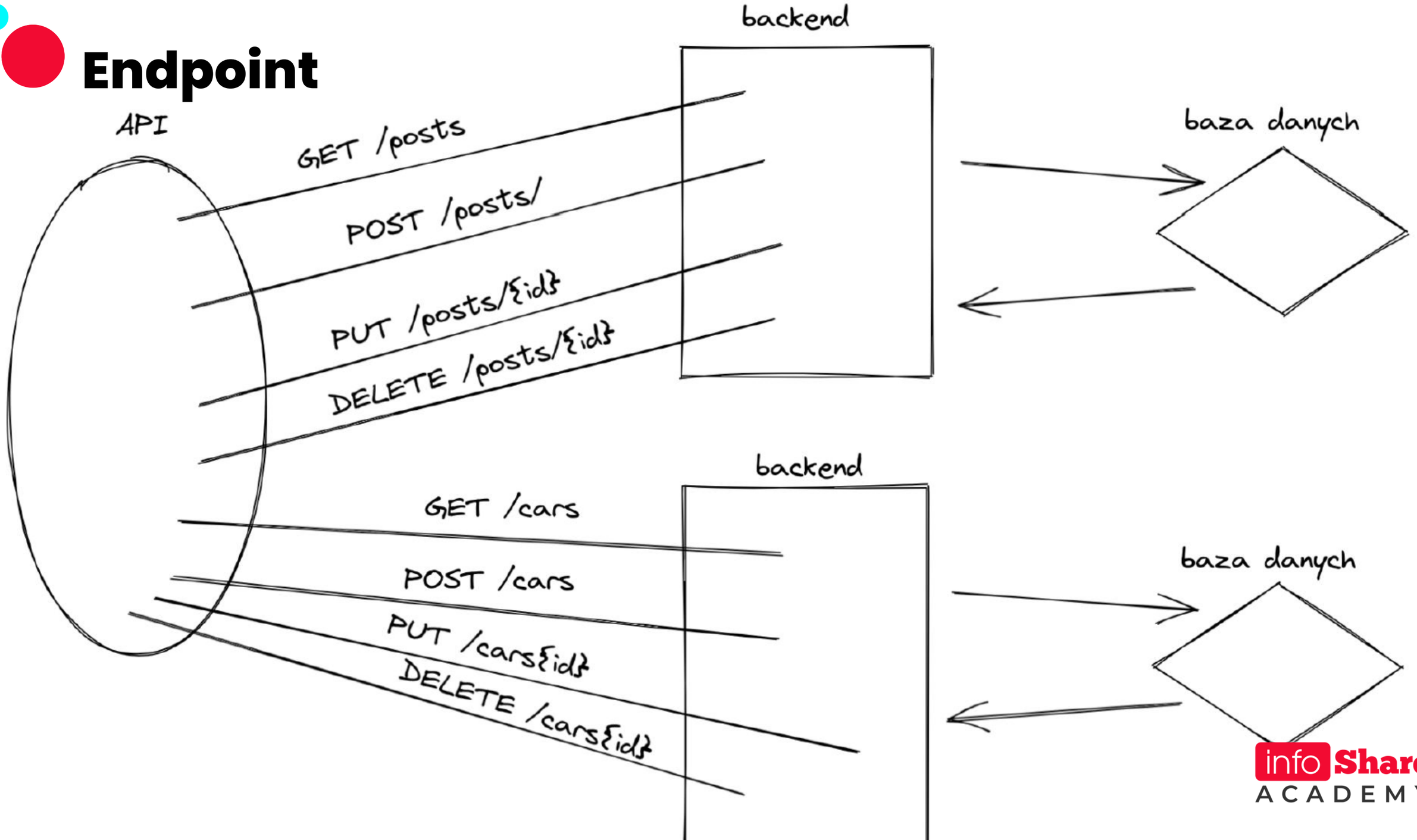
W przypadku interfejsów API punkt końcowy może zawierać adres URL serwera lub usługi.

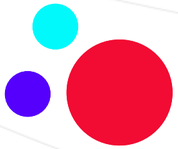
Każdy punkt końcowy to lokalizacja, z której interfejsy API mogą uzyskiwać dostęp do zasobów potrzebnych do wykonywania swoich funkcji.



infoShare ACADEMY

Endpoint





Kody Statusów w REST API

Operacje przeprowadzane na REST API podsumowywane są za pomocą kodów odpowiedzi HTTP. Dostępne kody znajdziesz w dokumentacji:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

1xx — informacyjne, nieczęsto można spotkać, dotyczą bardziej środowiska niż samej aplikacji (np. 111 — serwer odrzucił połaczenie)

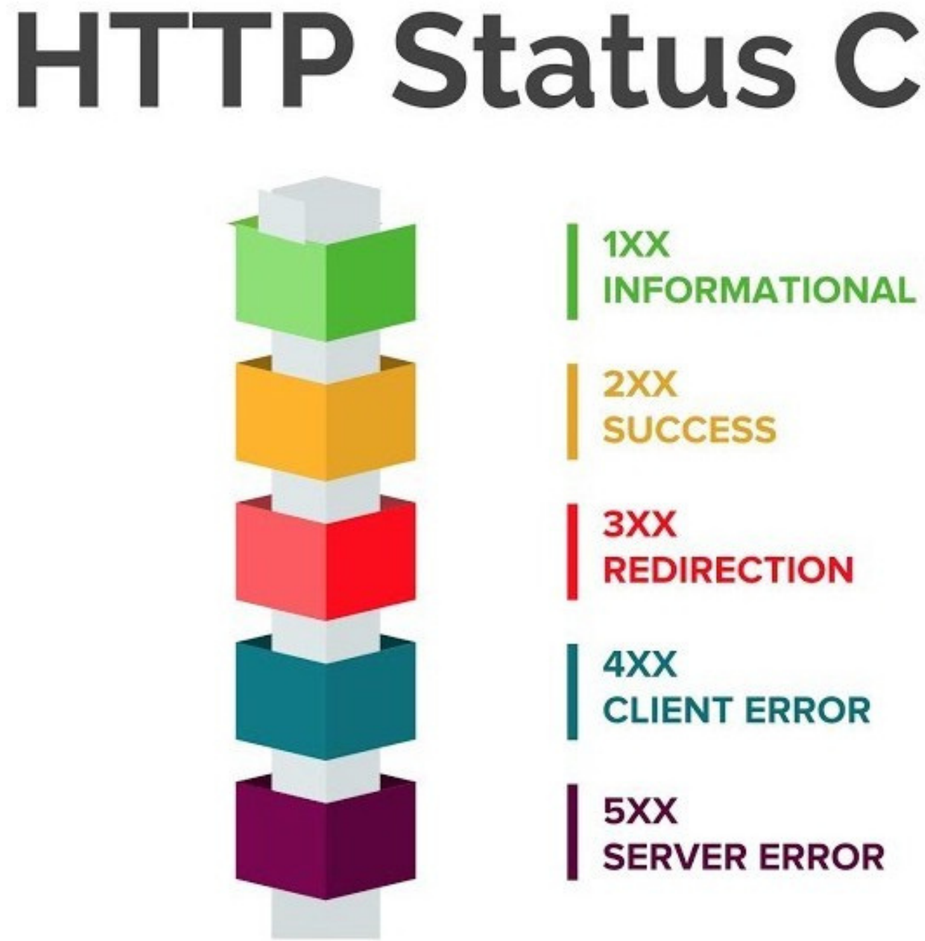
2xx — zapytanie się powiodło

3xx — przekierowanie, zapytanie należy kierować pod inny adres / serwer

4xx — błąd aplikacji spowodowany działaniem użytkownika (np. wspomniany 404 — nie znaleziono — czy 403 — brak dostępu lub 400 — niepoprawnie zapytanie)

5xx — błąd serwera (np. nieobsłużony wyjątek w Javie)

Kody Statusów w REST API



Metody HTTP

Metody pozwalają nam ‘rozdzielać’ zapytania trafiające pod ten sam adres.

Na przykład metody GET używamy do pobrania danych.

Metody POST do przesłania danych z przeglądarki do serwera — obie rzeczy możemy realizować pod tym samym adresem url.

GET

GET - Służy do pobierania danych z REST API. Jeśli użyjemy ogólnej ścieżki zasobu, REST Api zwróci listę wszystkich rekordów.

/todos - Odpowiedzią będzie tablica dostępnych obiektów Todo.

/todos/{id} - Odpowiedzią będzie pojedynczy obiekt Todo o wybranym id (lub 404 gdy wybrany zasób nie istnieje).

```
const fetchTodos = () => {
    fetch("http://localhost:3000/todos")
        .then((response) => {
            return response.json();
        })
        .then((todos) => {
            console.log(todos)
        });
    };
```

POST

POST - Służy do dodawania nowych elementów.

POST /todos - Przekazując jako body nowy element Todo (bez ID) stworzymy nowy zasób. W odpowiedzi otrzymamy utworzony zasób wraz z nadanym przez serwer ID.

```
const addTodo = (action) => {
    fetch("http://localhost:3000/todos", {
        method: "POST",
        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify({ action }),
    }).then((response) => {
        if (response.ok) {
            console.log('ok')
        }
    });
};
```

PUT & PATCH

PUT – Służy do nadpisywania istniejących elementów:
PUT /todos/{id} - Przekazując jako body edytowany element Todo (z istniejącym już ID) nadpiszemy zasób. W odpowiedzi otrzymamy edytowany zasób.

PATCH – Służy do edycji wybranych pól w zasobie.
PATCH /todos/{id} - Jako body przekazujemy pola które chcemy edytować. Jeśli nie przekażemy jakiegoś pola, to nie zostanie ono zmienione. W odpowiedzi (zazwyczaj) otrzymamy pełny zedytowany obiekt.

```
const updateTodo = (id, action) => {
    fetch(`http://localhost:3000/todos/${id}`, {
        method: "PUT",
        headers: {
        "Content-Type": "application/json",
        },
        body: JSON.stringify({ action }),
    }).then((response) => {
        if (response.ok) {
            console.log('ok')
        }
    });
};
```

infoShare ACADEMY

DELETE

DELETE - Służy do usuwania elementów.

DELETE /todos/{id} - Nie ma body. Usuwa zasób o podanym id.

```
const removeTodo = (id) => {
    fetch("http://localhost:3000/todos/" + id, {
        method: "DELETE",
    }).then((response) => {
        if (response.ok) {
            console.log('removed')
        }
    });
};
```

Metody HTTP - podsumowanie

Metoda	Request body	Response body	Zastosowanie / opis
GET	niedozwolone	opcjonalne	Pobieranie zasobu lub jego wyświetlenie, np. wyświetlenie formularza lub strony. Parametry można przekazywac jedynie poprzez adres (np. ?nazwa=wartosc&nazwa2=wartosc2).
POST	opcjonalne	opcjonalne	Przesłanie danych zapisanych jako pary klucz-wartość do serwera (np. wysłanie formularza, gdzie kluczem jest nazwa danego pola a wartością wpisana przez nas wartość). Metoda ta pozwala przesyłać także pliki (a także wiele pliki oraz pary klucz-wartość jednocześnie). Parametry są przekazywane w ciele zapytania, można także przekazywać parametry poprzez adres (tak jak w metodzie GET).
PUT	opcjonalne	opcjonalne	Przesyłanie ‘paczki’ danych, np. jednego pliku. Metoda ta ma pewne ograniczenia, np. nie ma możliwości łaczenia par klucz-wartość z inną przesyłaną treścią (np. plikiem). Obecnie używana głównie w przypadku RESTowych serwisów, gdzie ciałem jest np. formularz zapisany w postaci JSONa.
DELETE	opcjonalne	opcjonalne	Usuwanie zasobu na serwerze, z racji bezpieczeństwa praktycznie zawsze jest wyłaczona domyślnie. Obecnie używana głównie w przypadku RESTowych serwisów, wskazując, że dany zasób ma być usunięty (i obsługiwany przez aplikację, a nie sam serwer).

THANK YOU FOR YOUR ATTENTION

infoShareAcademy.com