Standard API

(interfejsu programistycznego aplikacji)

STANDARDY OTWARTOŚCI DANYCH

Fundusze Europejskie Polska Cyfrowa
Ministerstwo Cyfryzacji
OTWARTE DANE
Unia Europejska Europejski Fundusz Rozwoju Regionalnego

Spis treści

Słownik pojęć 3

1. Kategorie obecnych i potencjalnych odbiorców danych wykorzystujących API 5
2. Architektura usług sieciowych REST 10
3. Reguły nazewnictwa zasobów oraz parametry usług API 13
4. Opis składni oraz elementów URI 15
5. Opis dostępu do zasobów powiązanych za pomocą URI 17
6. Manipulowanie zasobami – filtrowanie, sortowanie, stronicowanie) 19
7. Metadane 22
8. Mechanizm linków do nawigacji 24
9. Wersjonowanie 26
10. Obsługa zdarzeń HTTP 28
11. Uwierzytelnianie i autoryzacja 32
12. Określenie reguł wprowadzania limitów danych per IP i per API-Key 34
13. Opis standardu dokumentacji usług API 37
14. Odstępstwa od standardu JSON API 40
15. Załączniki 42

Słownik pojęć

- **API** (ang. *Application Programming Interface*) – zbiór reguł ściśle opisujący, w jaki sposób programy lub podprogramy komunikują się ze sobą.
- **Autentykacja** (uwierzytelnianie) – weryfikuje użytkownika potwierdzając, że jest tym, za kogo się podaje, innymi słowy nadaje użytkownikowi kontekst.
- **Autoryzacja** – przyznaje uprawnienia do zawartości chronionej np. określonych zasobów API.
- **Centralne repozytorium** – centralne repozytorium informacji publicznej, o którym mowa w art. 9b ust. 1 ustawy z dnia 6 września 2001 r. o dostępie do informacji publicznej.
- **Dane publiczne** – liczby i pojedyncze wydarzenia lub obiekty na możliwie najniższym poziomie agregacji, które nie zostały poddane przez administrację publiczną przetworzeniu do postaci raportów, wykresów, itp. oraz nie został im nadany odpowiedni kontekst lub interpretacja.
- **Informacja sektora publicznego** – każda treść lub jej część, niezależnie od sposobu utrwalenia, w szczególności w postaci papierowej, elektronicznej, dźwiękowej, wizualnej lub audiowizualnej, będącą w posiadaniu podmiotów, o których mowa w art. 3 ustawy z dnia 25 lutego 2016 r. o ponownym wykorzystywaniu informacji sektora publicznego[1].

[1] „Informacje sektora publicznego" zdefiniowane w ustawie z dnia 25 lutego 2016 r. o ponownym wykorzystywaniu informacji sektora publicznego. Dotyczy następujących podmiotów: 1) jednostki sektora finansów publicznych w rozumieniu przepisów ustawy z dnia 27 sierpnia 2009 r. o finansach publicznych; 2) inne niż określone w pkt 1 państwowe jednostki organizacyjne nieposiadające osobowości prawnej; 3) inne niż określone w pkt 1 osoby prawne, utworzone w szczególnym celu zaspokajania potrzeb o charakterze powszechnym, niemających charakteru przemysłowego ani handlowego, jeżeli podmioty, o których mowa w tym przepisie oraz w pkt 1 i 2, pojedynczo lub wspólnie, bezpośrednio albo pośrednio przez inny podmiot: a) finansują je w ponad 50% lub b) posiadają ponad połowę udziałów albo akcji, lub c) sprawują nadzór nad organem zarządzającym, lub d) mają prawo do powoływania ponad połowy składu organu nadzorczego lub zarządzającego; 4) związki podmiotów, o których mowa w pkt 1-3.

Jednostka informacji publicznej – najmniejszy wyodrębniony element zasobu informacyjnego[2].

Ponowne wykorzystywanie – wykorzystywanie przez osoby fizyczne, osoby prawne i jednostki organizacyjne nieposiadające osobowości prawnej, zwane dalej „użytkownikami", informacji sektora publicznego, w celach komercyjnych lub niekomercyjnych innych niż pierwotny publiczny cel, dla którego informacja została wytworzona[3].

Zasoby informacyjne – informacje publiczne o szczególnym znaczeniu dla rozwoju innowacyjności w państwie i rozwoju społeczeństwa informacyjnego, które ze względu na sposób przechowywania i udostępniania pozwalają na ich ponowne wykorzystywanie w rozumieniu przepisów ustawy z dnia 25 lutego o ponownym wykorzystywaniu informacji sektora publicznego, w sposób użyteczny i efektywny; i są udostępniane w centralnym repozytorium[4].

[2] Zdefiniowane w rozporządzeniu Rady Ministrów z dnia 12 marca 2014 r. w sprawie Centralnego Repozytorium Informacji Publicznej.

[3] Zdefiniowane w ustawie z dnia 25 lutego 2016 r. o ponownym wykorzystywaniu informacji sektora publicznego.

[4] Zdefiniowane w ustawa z dnia 6 września 2001 r. o dostępie do informacji publicznej.

1.

Kategorie obecnych i potencjalnych odbiorców danych wykorzystujących API

Odbiorcami korzystającymi z danych publicznych (obecnie i docelowo) są zarówno osoby fizyczne jak i przedsiębiorstwa, jednostki administracji publicznej (centralnej i samorządu terytorialnego), organizacje pozarządowe, podmioty nauki, kultury i inni interesariusze.

Ponieważ dane publiczne mogą opisywać rzeczywistość w dowolnym aspekcie, to również interesariusze mogą przetwarzać te dane na wiele sposobów. To kształtuje bezpośrednio charakterystykę korzystania z API oraz wpływa na wymagania przed nim stawiane.

Większość użytkowników API będzie korzystać z produktów pochodnych opracowanych w oparciu o dane pochodzące z API (usług, aplikacji komputerowych, informacji przetworzonych, know-how) często nieświadomie. Jest to tzw. grupa użytkowników pośrednich, która bezpośrednio nie korzysta z API, ale czerpie z niego korzyści i może mieć wpływ na pożądany kierunek jego ewolucji.

Pozostałych, bezpośrednich użytkowników można podzielić na dwie grupy:

- użytkownicy okazjonalni (*ad-hoc*),
- użytkownicy profesjonalni.

Szczegółowa charakterystyka grup użytkowników znajduje się w **Załączniku nr 1.**

1.1. Rekomendacje szczegółowe ze względu na potrzeby użytkowników okazjonalnych

Rekomenduje się następujące wymagania w zakresie API, które musi:

1) Niwelować technologiczną barierę wejścia, rozumianą jako konieczność posiadania przygotowania technicznego do wykonywania podstawowych interakcji z API.
2) Wymagać niskich nakładów czasu i pracy oraz nauki do korzystania z API.
3) Pozwalać na korzystanie z zasobów w sposób okazjonalny bez uwierzytelniania.
4) Posiadać prosty w obsłudze interfejs webowy, który umożliwi skorzystanie z API bez konieczności wpisywania rozbudowanego kodu, lecz poprzez składanie

gotowych jego elementów, i natychmiastowe uzyskanie wyników zapytania oraz wykonywanie na nich podstawowych działań: filtrowania, sortowania, przeszukiwania i linkowania do poszczególnych URI.

5) Umożliwić użytkownikom zachowanie anonimowości, a do korzystania z kreatora nie jest wymagana rejestracja, zakładanie konta itp.

1.2. Rekomendacje szczegółowe ze względu na potrzeby użytkowników profesjonalnych

Rekomenduje się następujące wymagania w zakresie API, które musi:

1) Dostarczać zgodny z uznanymi standardami interfejs ze spójnie nazwanymi zasobami.
2) Być gotowe do natychmiastowego korzystania oraz mieć niską barierę wejścia.
3) Dostarczać danych z kompletem metadanych.
4) Pokazywać korzyści z używania API w kontekście konkretnych zastosowań (np. poprzez przykłady zawarte w dokumentacji).
5) Zaspokajać zróżnicowane potrzeby developerów.
6) Dostosowywać się do zmian i nowych potrzeb (zbierać propozycje od użytkowników).
7) Zapewniać podstawową dokumentację usług oraz udostępniać pełną informację o strukturze danych.
8) Od początku przewidywać obsługę wersjonowania API.
9) Obsługiwać manipulację danych: sortowanie, filtrowanie, przeszukiwanie.
10) Zapewniać stabilność, wydajność, skalowalność oraz bezpieczeństwo.
11) Być traktowane jako produkt wspierający innowacyjność. W praktyce oznaczać to będzie np. nacisk na sprawne rozwiązywanie problemów i obsługę pytań

developerów przez zespół obsługujący API lub zespół związany z centralnym repozytorium.

12) Zapewniać sprawną rejestrację nowych aplikacji.

13) Zezwalać na intensywne wykorzystanie i realizowanie gwałtownych wzrostów obciążeń.

14) Dostarczać dane umożliwiające przesłanie (np. poprzez email) pytań, uwag, zgłoszeń błędów itp. bezpośrednio do osoby / komórki odpowiedzialnej za zasób API.

1.3. Rekomendacje ogólne w kontekście użytkowników API

W kontekście użytkowników API formułuje się następujące rekomendacje standaryzujące API:

1) Posiada spójny interfejs, korzystający z uznanych standardów światowych.

2) Zapewnia dokumentację usług, w szczególności pełną informację o strukturze udostępnianych danych.

3) Wymaga niskich nakładów pracy do korzystania z API w podstawowym zakresie za pomocą prostego w obsłudze interfejsu webowego. Interfejs musi umożliwiać skorzystanie z API bez konieczności wpisywania rozbudowanego kodu, lecz poprzez składanie gotowych elementów zapytań. Interfejs musi dostarczać szybko rezultatów – danych prezentowanych w przystępny dla odbiorcy sposób z możliwością interakcji: filtrowania, sortowania i przeszukiwania pod kątem więcej niż jednego warunku. Musi pozwalać na korzystanie z zasobów bez uwierzytelniania.

4) Zapewnia gotowość do natychmiastowego korzystania (niska bariera wejścia).

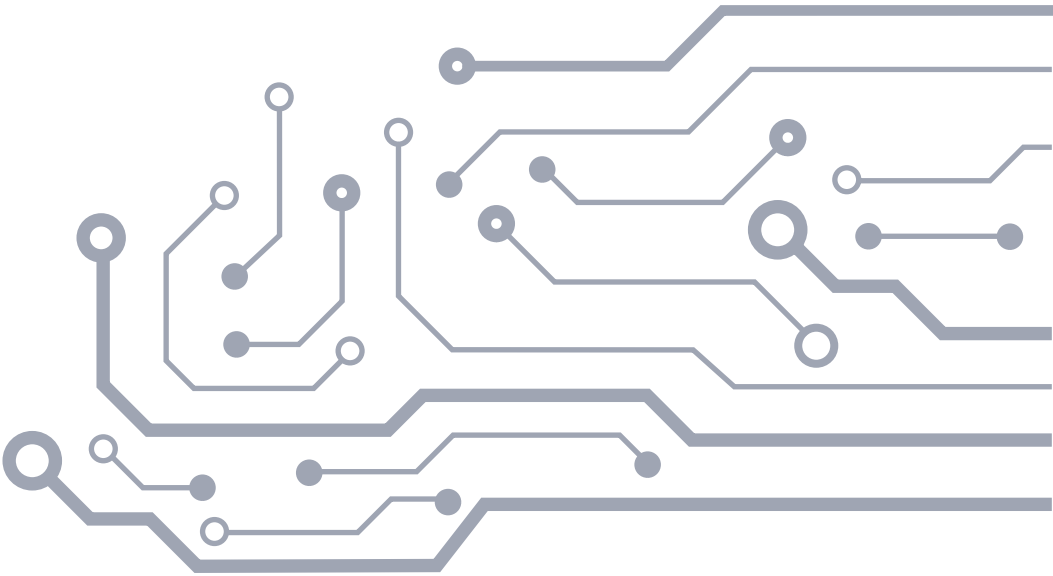
5) Zapewnia obsługę wersjonowania kolejnych odsłon API.

6) Obsługuje manipulację danych: sortowanie, filtrowanie, przeszukiwanie po więcej niż jednym warunku.

7) Zapewnia skalowalność, stabilność, wydajność oraz bezpieczeństwo.

8) Określa zasady limitów wykorzystania, jeśli takie są wprowadzone.

9) W przypadku możliwości korzystania z kluczy API, umożliwia natychmiastowe wygenerowanie odpowiednich API-Key.



2.

Architektura usług sieciowych REST

Standard API jest oparty o standard REST. Ponadto referencyjnie wykorzystuje format reprezentacji danych zdefiniowany przez standard JavaScript Object Notation czyli JSON[5]. Jako standard notacji JSON przyjmuje się JSON Schema. Kontekst ułatwiający korzystanie z danych w sposób automatyczny jest generowany zgodnie ze standardem JSON-LD. Łączenie obiektów według zasad „linked data" zapewnione jest zgodnie ze standardem hydra:Link.

Opis przyjętej architektury znajduje się w **Załączniku nr 2**.

Rekomendacje dotyczące architektury usług sieciowych REST

W kontekście stosowania standardu REST formułuje się następujące rekomendacje standaryzujące:

1) Budowa API jest oparta o zasoby.
2) Manipulacja zasobami powinna być oparta o reprezentacje zasobów.
3) API wspiera obsługę buforowania, w tym zapewnia obsługę polecenia HEAD.
4) API jest wyposażone w jednolity interfejs z unikalną identyfikacją zasobów, dopuszczalne jest istnienie więcej niż jednego URI wskazującego zasób w przypadkach uzasadnionych (wersjonowanie API, różne wersje językowe itp.).
5) API funkcjonuje w modelu klient-serwer, gdzie klient komunikuje się z serwerem tylko poprzez interfejs API zgodny ze Standardem API.
6) API zapewnia możliwość wykorzystania samoopisujących się komunikatów.
7) API zapewnia informację o strukturze danych.
8) API zapewnia dostęp do zasobów w formacie JSON; opcjonalne jest udostępnianie danych również w formacie XML klientom oczekującym tego za pomocą nagłówka **Accept** lub parametru **&format=** w URL.
9) Jako standard notacji JSON przyjmuje się JSON Schema.

[5] Schema JSON http://www.yaml.org/spec/1.2/spec.html#id2803231 [Dostęp 12.02.2018]

10) Dla danych spełniających założenia „trzech gwiazdek”[6] otwartości – wymagane stosowanie podstawowego standardu JSON.
11) Dla danych spełniających założenia „czterech gwiazdek” otwartości – wymagane dodanie kontekstu obejmującego wszystkie atrybuty zgodnie ze standardem JSON-LD oraz paginacji kolekcji zgodnie z hydra:PartialCollectionView[7].
12) Dla danych spełniających założenia „pięciu gwiazdek” otwartości – wymagane stosowanie linkowania zgodnie ze standardem Hydra (hydra:Link[8]).
13) Interakcja z API jest możliwa wyłącznie przy użyciu szyfrowanego połączenia HTTPS.

[6] Zgodnie z klasyfikacją zaproponowaną przez Tima Berners-Lee, **http://5stardata.info** [dostęp: 24 kwietnia 2018 r.]

[7] **https://www.hydra-cg.com/spec/latest/core/#collections** [dostęp: 5 kwietnia 2018 r.]

[8] **https://www.hydra-cg.com/spec/latest/core/#adding-affordances-to-representations** [dostęp 5 kwietnia 2018 r.]

3.

Reguły nazewnictwa zasobów oraz parametry usług API

REST wykorzystuje identyfikatory zasobu do jednoznacznego określania konkretnego zasobu. Dla zachowania spójności zalecane jest stosowanie jednolitych reguł nazewnictwa, gdyż mają one istotny wpływ na łatwość użycia, prostotę i zrozumiałość funkcjonowania API opartego o REST.

Rekomendacje dotyczące reguł nazewnictwa zasobów oraz parametrów usług API

W kontekście nazewnictwa zasobów API formułuje się następujące rekomendacje standaryzujące:

1) Stosowanie prostego i objaśniającego nazewnictwa.
2) Oznaczanie zasobów rzeczownikami, np. czujniki, samochody.
3) Stosowanie rzeczowników w liczbie mnogiej (np. **formularze** zamiast ~~formularz~~, **czujniki** zamiast ~~czujnik~~ itp.).
4) Niestosowanie czasowników lub sformułowań opisujących czynności (np. „~~pobierz-formularze~~" lub „~~wyswietla-listę czujników~~").
5) Stosowanie nazewnictwa konkretnie wskazujące na zasób zamiast nazewnictwa, gdzie znaczenie wymaga znajomości kontekstu do jego zrozumienia (np. „**przeglady-rejestracyjne**" zamiast „~~przeglady~~").
6) Niestosowanie polskich znaków w polach będących identyfikatorem zasobów (np.: ~~usługi~~), zwłaszcza, jeżeli są one wykorzystywane w URI zasobu.
7) Oznaczanie zasobów za pomocą małych liter[9] (np. **pojazdy-zarejestrowane** zamiast ~~Pojazdy-Zarejestrowane~~).
8) Używanie „-" (łączników) do łącznia fraz złożonych (np. **pojazdy-zarejestrowane** zamiast ~~pojazdyzarejestrowane~~).
9) Nieużywanie „_" (znaków podkreślenia) do łącznia fraz złożonych (np. **pojazdy-zarejestrowane** zamiast pojazdy_zarejestrowane) lub innych znaków specjalnych.

[9] Zgodnie z RFC 2616 ścieżka i zapytanie w URL mają rozróżnianą wielkość liter http://w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.2.3 [Dostęp 12.02.2018]

4.

Opis składni oraz elementów URI

URI (z ang. Uniform Resource Identificator) jest unikalnym identyfikatorem zasobów, stosowanym zgodnie z RFC 1736[10], RFC 1737[11] oraz ustandaryzowanym zgodnie z RFC 3986[12].

Definicje, szczegółowy opis składni oraz przykłady zastosowań znajdują się w **Załączniku nr 3**.

Rekomendacje dotyczące składni oraz elementów URI

W kontekście składni URI formułuje się następujące rekomendacje standaryzujące:

1) Domena w URI musi jednoznacznie wskazywać na właściciela API. Domena ta musi udostępniać funkcjonalność API poprzez subdomenę „api", chyba że nie jest to możliwe z uzasadnionego powodu.
2) API może być wyposażone w portal dla developerów, który zawiera dokumentację API, informacje kontaktowe, generator kluczy API, przykładowy kod itp. Portal powinien być integralna częścią API i być dostępny poprzez subdomenę „dev", chyba że nie jest to możliwe z uzasadnionego powodu. W przypadku braku osobnego portalu, na głównej stronie API muszą znaleźć się podstawowe informacje pozwalające na korzystanie z niego – w tym informacje o limitach oraz o możliwości uzyskania klucza API, jeśli te opcje są dostępne.
3) Znak „/" (ukośnik, ang. *forward slash*) służy oznaczeniu zależności hierarchicznej między zasobami.
4) Stosowanie znaku „/" (ukośnik, ang. *forward slash*) na końcu URI nie jest zalecane.
5) Ścieżki URI są oznaczane z użyciem małych liter.
6) W URI nie stosuje się rozszerzeń plików oraz nazw formatów danych. Informacje te przekazywane są przez nagłówki HTTP lub w metadanych zasobów.

[10] https://www.ietf.org/rfc/rfc1736.txt [Dostęp 29.03.2018]

[11] https://www.ietf.org/rfc/rfc1737.txt [Dostęp 29.03.2018]

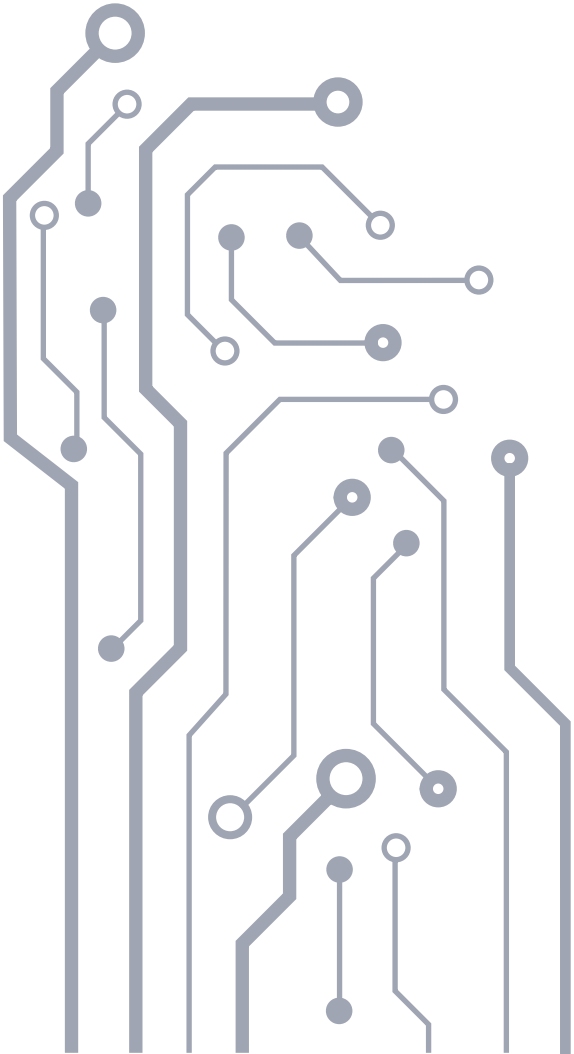
[12] https://www.ietf.org/rfc/rfc3986.txt [Dostęp 12.02.2018]

5.

Opis dostępu do zasobów powiązanych za pomocą URI

W REST klient dostarcza stan poprzez ciało zapytania, parametry zawarte w kwerendzie, nagłówki. Usługa API dostarcza klientowi stan poprzez ciało komunikatu, kody i nagłówki odpowiedzi. Technicznie oznacza to komunikację za pomocą hiperlinków i hipertekstu.

API musi używać techniki HATEOAS, czyli linków zawartych w ciele lub nagłówkach odpowiedzi, które zawierają identyfikatory URI do zasobów powiązanych.



6.

Manipulowanie zasobami – filtrowanie, sortowanie, stronicowanie

W założeniu szereg klienckich aplikacji będzie korzystać z zasobów API automatycznie. Aplikacje takie rzadko realizują zadania wymagające pobrania z API zbioru danych w pełnym przekroju informacyjnym oraz pełnej zawartości rekordów. Znacznie częściej potrzebują odpowiednio spreparowanego, niewielkiego wycinka zamiast pełnego zakresu dostępnych danych. Wycinek ten w stosunku do oryginalnego zbioru może być filtrowany względem zakresu pobieranych danych, w kontekście określonych kryteriów, porcji danych pobieranej za jednym razem oraz w odpowiednim uporządkowaniu. Dlatego założono, że API udostępnia funkcjonalność filtrowania, sortowania i stronicowania zasobów, które określane będzie mianem manipulacji danymi.

Zapytania i odpowiedzi systemu są realizowane zgodnie ze standardem JSON API w wersji 1.0 lub nowszej[13], chyba że specyfikacja OD API stanowi inaczej. W odpowiedziach zaleca się uzupełnienie JSON API o JSON-LD[14] w celu dodania kontekstu koniecznego do automatycznego przetwarzania danych API.

Podstawowe zasady filtrowania, sortowania i stronicowania zasobów wraz z przykładami znajdują się w **Załączniku nr 4**.

Rekomendacje dotyczące manipulowania zasobami

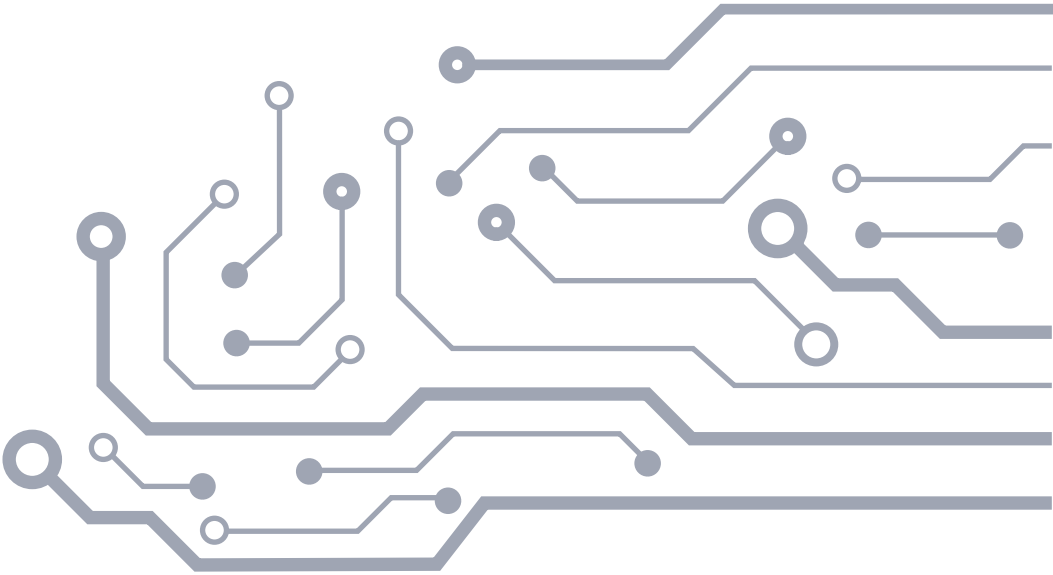
W kontekście manipulowania reprezentacjami zasobów – filtrowaniem, sortowaniem i stronicowaniem – formułuje się następujące rekomendacje standaryzujące:

1) Proste stronicowanie, filtrowanie i sortowania reprezentacji zasobów odbywają się poprzez URI.
2) Równolegle API może obsługiwać zapytania przesyłane w ciele wiadomości w zakresie szerszym niż przez URI.
3) Standardem wykorzystywanym przez OpenAPI jest JSON API w wersji 1.0 lub nowszej, chyba że niniejsza specyfikacja wprowadza od niej odstępstwa.

[13] JSON API – Specification – http://jsonapi.org/format/ [dostęp: 31.03.2018]

[14] JSON for Linking Data – https://json-ld.org/spec/latest/json-ld/ [dostęp: 3.04.2018]

4) W odpowiedziach zaleca się dostarczanie dodatkowego kontekstu za pomocą JSON-LD.
5) Zapytania asynchroniczne są obsługiwane przez kody HTTP o numerach 202 i 303.
6) Obsługa kwerend SQL jest zabroniona.



7.

Metadane

Zasoby udostępniane przez API wyposażane są w metadane, stanowiące część odpowiedzi na zapytanie o reprezentację zasobu. Metadane to inaczej dane o danych, które opisując zasoby ułatwiają ich zrozumienie oraz zarządzanie nimi. Są szczególnie istotne w procesie maszynowego przetwarzania informacji, dlatego istotne jest wykorzystanie rodzajów metadanych tam, gdzie tylko jest to możliwe. W przypadku API, metadane muszą uwzględniać lokalne wytyczne określone przez PODP oraz rozporządzenie Rady Ministrów z dnia 12 marca 2014 r. w sprawie Centralnego Repozytorium Informacji Publicznej.

Tabela, w której przedstawiono minimalny zestaw metadanych dla zasobu znajduje się w **Załączniku nr 5**.

Rekomendacje dotyczące metadanych

W kontekście określania minimalnego zestawu metadanych formułuje się następujące rekomendacje standaryzujące:

1) Zasoby są opisywane za pomocą minimalnego zestawu metadanych określonego w niniejszym dokumencie – chyba, że dana metadana nie dotyczy konkretnego rodzaju treści.
2) Gdy metadane nie występują dla określonego rodzaju zasobów, nie są dołączane do metadanych danego zasobu.
3) Metadane określane przez dostawcę zasobu określane są zgodnie ze zdefiniowanymi zasadami nazewnictwa, w miarę możliwości z wykorzystaniem standardowych rozwiązań (schema.org, DCMI itp.).
4) W przypadku wprowadzania niestandardowych metadanych, konieczne jest dostarczenie słownika, do którego będą się odnosiły opisy metadanych.

8.

Mechanizm linków do nawigacji

API zapewnia funkcjonalność linków do nawigacji. Polega ona na dołączaniu w ciele zasobu poniższych linków (oznaczanych jako „links”), zgodnie z JSON API 1.0 lub późniejszym.

Objaśnienia i przykłady stosowania znajdują się w **Załączniku nr 6**.

Rekomendacje dotyczące linków do nawigacji

W kontekście linków do nawigacji formułuje się następujące rekomendacje standaryzujące:

1) W odpowiedzi na przesyłane przez użytkowników zapytania API udostępnia stany wraz ze zdefiniowanymi w JSON API 1.0 linkami ułatwiającymi poruszanie się miedzy zasobami powiązanymi. Linki są oznaczane:
   a) *prev* (poprzedni),
   b) *self* (bieżący),
   c) *next* (następny),
   d) *first* (pierwszy)
   e) *last* (ostatni).

9.

Wersjonowanie

Wersjonowanie jest jedną z najistotniejszych kwestii do rozważenia przy tworzeniu API. Od tego jak wersjonowanie zostanie zaplanowane i zaimplementowane zależy nie tylko wygoda, ale i możliwość korzystania z API w przyszłości oraz dostępność jego zasobów.

Zasady wersjonowania, metody oznaczania wersji wraz z przykładami oraz techniki zarządzania wersjami zawarte są w **Załączniku nr 7**.

Rekomendacje dotyczące wersjonowania API

W kontekście wersjonowania API formułuje się następujące rekomendacje standaryzujące:

1) Zmiany API oznaczać za pomocą standardu wersjonowania semantycznego API[15].
2) Do oznaczenia i wywoływania obsługi wersją o konkretnym numerze ma znaczenie tylko pierwszy człon (MAJOR) wersji, a pozostałe (MINOR i PATCH) są ignorowane zachowując jedynie wartość informacyjną w kontekście wersji interfejsu.
3) Obowiązkowe jest odwołanie się do wersji poprzez element URL (**/api/vXX/**).
4) Dopuszczalne jest odwoływanie się do wersji dodatkowo poprzez parametr URI **„v=XX”**, dodatkowy nagłówek „api-version=XX” lub w nagłówku **„Accept:”**.
5) Jeśli wersja API nie została określona, używa się wersji bieżącej (najnowszej stabilnej).
6) Informacje o aktualnej wersji można pobrać przez zasób /version.
7) Przed planowaniem zmiany wersji należy oznaczać wersję bieżącą statusem **„deprecated”**.
8) Poza uzasadnionymi wypadkami, nie należy wycofywać obsługi wersji API szybciej niż dwa lata po powiadomieniu użytkowników.
9) po wyłączeniu obsługi starej wersji API, odwołania do niej należy obsłużyć, w zależności od kontekstu, błędem HTTP o kodzie nr 406 lub 410.

[15] Zob. https://semver.org/spec/v2.0.0.html [Dostęp 12.02.2018]

10.

Obsługa zdarzeń HTTP

API będące tzw. RESTful API opera się o HyperText Transfer Protocol (HTTP), włączając w to wybrane metody, kody i nagłówki odpowiedzi. API korzysta głównie z metody GET. W API zakłada się niewykorzystywanie metod umożliwiających modyfikację zasobów. HTTP dostarcza pięć kategorii oraz ponad 30 kodów odpowiedzi. Jednak nie wszystkie kody będą w API przydatne, dlatego API przewiduje obsługę wybranych kodów.

Tabelę grup kodów zgodnie z RFC 7231 oraz przykłady odpowiedzi i komunikatów znajdują się w **Załączniku 8**.

10.1. Rekomendacje dotyczące kodów odpowiedzi w ramach obsługi zdarzeń HTTP

W kontekście obsługi zdarzeń HTTP – kody odpowiedzi, formułuje się następujące rekomendacje standaryzujące:

1) Określa się minimalny obsługiwany zestaw kodów odpowiedzi:

Grupa kodów	Znaczenie	Informacja na temat błędu
2xx	200	OK
	201	Utworzono nowy zasób
	202	Zaakceptowano zapytanie, odpowiedź jeszcze nie jest gotowa
	204	Brak zawartości
3xx	301	Przeniesiono na stałe
	303	Zobacz inne: odpowiedź na zapytanie dostępna pod innym adresem
	304	Nie modyfikowano od czasu wskazanego w zapytaniu

Grupa kodów	Znaczenie	Informacja na temat błędu
4xx	400	Błędne żądanie
	401	Brak autoryzacji
	403	Dostęp zabroniony
	404	Nie znaleziono
	405	Nieodpowiednia metoda
	406	Nieakceptowalne – treść nie może zostać przygotowana w sposób zasygnalizowany nagłówkami
	409	Konflikt
	410	Zasób usunięty i już nigdy nie będzie dostępny pod tym adresem
	422	Zapytanie niepoprawne semantycznie
	429	Zbyt wiele zapytań w jednostce czasu
5xx	500	Wewnętrzny błąd serwera
	502	Błąd bramki – serwer otrzymał niepoprawną odpowiedź od serwera nadrzędnego.
	503	Usługa czasowo niedostępna.
	504	Przekroczony czas oczekiwania na odpowiedź serwera wewnętrznego.

2) Do przesłania informacji o numerze błędu, wykorzystuje się element Status-Line w nagłówku HTTP.

   W przypadku kodu błędów opis uzupełnia się informacją na temat błędów, która powinna być zawarta w ciele odpowiedzi.

   Kody z kategorii 4xx i 5xx informują o błędach. Użytkownik nie ma wpływu na błędy z kategorii 5xx gdyż leżą one po stronie serwera. Przy uzyskaniu błędu z kategorii 4xx istnieje prawdopodobieństwo wystąpienia błędów charakterystycznych w określonych sytuacjach. Ranking tych błędów może być tworzony w oparciu o cykliczną analizę logów API. Ranking ten powinien służyć do aktualizacji informacji objaśniających błędy.

10.2. Rekomendacje dotyczące informacji o błędach w ramach obsługi zdarzeń HTTP

W kontekście obsługi zdarzeń HTTP – kody błędów grupy 4xx, formułuje się następujące rekomendacje standaryzujące:

1) W ciele odpowiedzi HTTP powinny być zawarte następujące informacje objaśniające:

Nazwa objaśnienia	Identyfikator objaśnienia
Wiadomość o błędzie	errorResult
Wiadomość objaśniająca przyczynę błędu	errorReason
Wiadomość objaśniająca możliwe rozwiązanie (opcjonalnie, tam gdzie jest to możliwe)	errorSolution
Kod błędu zdefiniowany przez dostawcę	errorCode
Link do dokumentacji z objaśnieniem numeru błędu dla danej wersji interfejsu	errorHelp

2) Komunikaty muszą dostarczać konkretnej informacji zwrotnej. Niedopuszczalne są komunikaty, które nie pomagają w rozwiązania problemu przez użytkownika. Oznacza to, że komunikat powinien być:

   a) sformułowany precyzyjnie w sposób konkretny,

   b) możliwy do sprawdzania w kontekście przyczynienia się do rozwiązania problemu,

   c) osiągalny, czyli informacje/zalecenia zawarte w komunikacie powinny być wykonalne przez użytkownika w kontekście informacji, którymi dysponuje,

   d) wartością dodaną w porównaniu do sytuacji, w której komunikat by nie występował.

11.

Uwierzytelnianie i autoryzacja

Systemy implementujące API mogą korzystać z systemu uwierzytelniania użytkowników. Nie jest dopuszczalne zamknięcie API dla użytkowników nieuwierzytelnionych, dozwolone jest wprowadzenie dla nich dodatkowych ograniczeń w korzystaniu z API.

Opisy technologii i rozwiązań dotyczących uwierzytelniania i autoryzacji znajdują się w **Załączniku nr 9**.

Rekomendacje dotyczące uwierzytelniania i autoryzacji

W kontekście uwierzytelniania i autoryzacji formułuje się następujące rekomendacje standaryzujące:

1) Uwierzytelnianie i autoryzacja użytkowników w API może być oparta o framework OpenID Connect.
2) Stosowanie OAuth 2.0. jest dopuszczone, ale w połączeniu z innymi rozwiązaniami wypełniającymi lukę autoryzacji.
3) Obowiązkowe jest stosowanie szyfrowania SSL.
4) Uwierzytelnianie użytkowników API nie jest obowiązkowe.

12.

Określenie reguł wprowadzania limitów danych *per* IP i *per* API-Key

W przypadku zbyt dużego obciążenia systemów udostępniających dane publiczne, dozwolone jest wprowadzenie szczególnych ograniczeń na konkretny adres IP lub dla konkretnego klucza API.

Rekomendacje dotyczące reguł wprowadzania limitów danych

W kontekście reguł określania limitów danych formułuje się następujące rekomendacje standaryzujące:

1) Interfejs API nie powinien dopuszczać do nadmiernego obciążania zasobów API uniemożliwiającego korzystanie z niego przez użytkowników, dając w razie konieczności możliwość nałożenia limitów per-IP oraz per-API-Key.
2) Udostępniony webowy kreator zapytań powinien być osobno zabezpieczony przed wykorzystaniem go do nadmiernego obciążania zasobów API.
3) Klucze należy konstruować zgodnie z zasadami konstrukcji tokenów Web JSON, z domyślnie ustawionym prawem dostępu tylko-do-odczytu dla wszystkich zasobów API.
4) W przypadku wprowadzenia ograniczeń, dla API musi zostać opracowana polityka korzystania z API oraz plany korzystania z API przez użytkowników, rozróżniająca użytkowników niezarejestrowanych, użytkowników posiadających API-Key oraz użytkowników z białych list – jeśli takie rozróżnienie zostało wprowadzone.
5) W przypadku wprowadzenia limitów, musi zostać utworzona biała lista użytkowników korzystających z API z priorytetem wyższym niż inni użytkownicy – w szczególności systemów publicznych, dla których dane są danymi referencyjnymi. Reguły wpisu na białą listę powinny być jednoznacznie sformułowane.
6) Gdy nie narusza to stabilności systemu, dopuszczone jest obciążanie API w sposób wykraczający poza plany obciążeń, szczególnie przez użytkowników z białych list.

7) Należy używać reguł dławiących destrukcyjnych użytkowników, w przypadku zagrożenia zablokowaniem API (DOS attack). Zaleca się stosowanie polityki długoterminowego uznawania adresów IP za wrogie (czarne listy) i wykorzystywanie zewnętrznych czarnych list[16] do analizy zdarzeń i rozszerzania listy adresów blokowanych długoterminowo.

8) Transfer może być ograniczamy dla użytkowników generujących transfer i obciążenie zagrażające ciągłości i stabilności API.

9) Transfer może być ograniczany dla użytkowników w oparciu o dynamicznie aktualizowane zasady wynikające z analiz faktycznego sposobu korzystania z API przez użytkowników.

10) Transfer może być ograniczany dla użytkowników z czarnej listy adresów IP.

11) Adresy z czarnej listy adresów IP powinny być weryfikowane co najmniej co 6 miesięcy i usuwane z niej w przypadku braku podejrzanego ruchu z danego adresu.

12) Ograniczenie transferu powinno być w każdym przypadku ostatecznością wynikającą ze stosowania określonych reguł, a nacisk należy położyć na podnoszenie wydajności API.

13) Nałożenie limitów powinno skutkować błędem HTTP o kodzie 429 z informacją o polityce limitów oraz możliwości dodania do białej listy.

[16] Np. http://apility.io [Dostęp 12.02.2018]

13.

Opis standardu dokumentacji usług API

Rekomendacje dotyczące standardów dokumentacji usług API

W kontekście standardów dokumentacji usług API formułuje się następujące rekomendacje standaryzujące:

Dokumentacja API musi zawierać co najmniej:

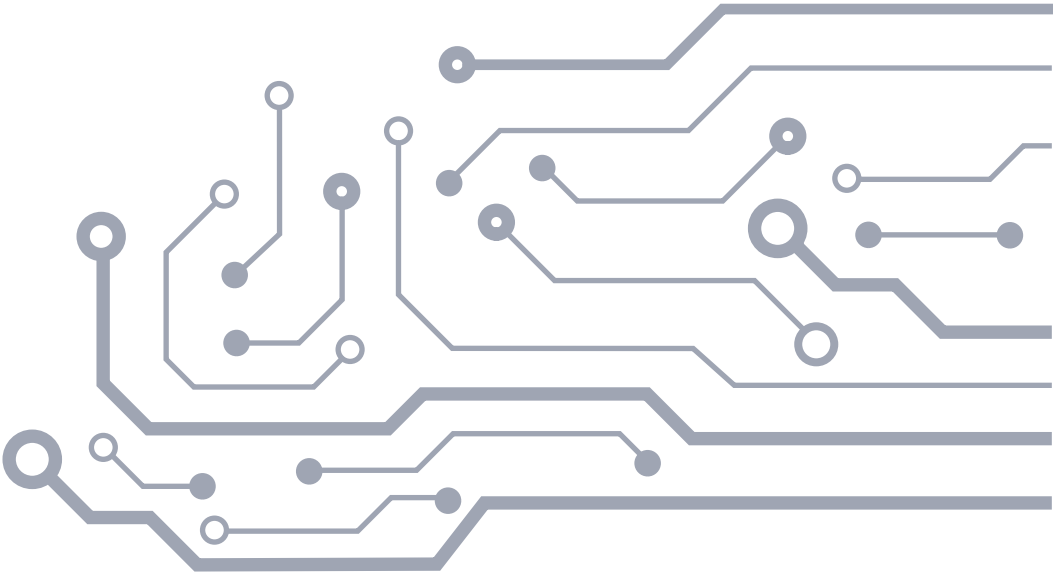
1) Opis zasobów – opis informacji dostarczanej przez każdy zasób, znaczenia zasobu, punktów dostępowych, metod do każdego punktu dostępowego.
2) Opis obsługiwanych metod i punktów dostępowych, sposobu dostępu do zasobów, dopuszczonych metod interakcji z zasobami.
3) Parametry obsługiwane przez punkt dostępowy w szczególności dotyczące:
   a) metody HTTP (GET, POST, PATCH itp.),
   d) nagłówków,
   e) ścieżki dostępu (URI),
   f) parametrów żądania (query),
   g) ciała żądania (body), np.: JSON.
8) Przykłady kodu żądań obrazujące użycie złożonych, ale nieskomplikowanych zapytań.
9) Przykłady odpowiedzi: struktur i treści przesyłanych klientowi w odpowiedzi na żądania z powyższego punktu – w formie zapisu w dokumentacji lub odwołania do rzeczywistego systemu.

Zaleca się, by dokumentacja API:

- stosowała oznaczenia zapewniające możliwość odczytu maszynowego,
- była dynamiczna i interaktywna,
- automatycznie generowała przykłady kodu dla bibliotek klienta.

W zakresie realizacji wytycznych w zakresie dokumentacji oraz uzupełniająco zaleca się stosowanie standardu OpenAPI w obowiązującej obecnie wersji 3.0. lub nowszej.

Na rynku dostępne są narzędzia do generowania dynamicznej i interaktywnej dokumentacji API. Umożliwiają one budowanie kreatorów zapytań do API, a niektóre dedykowane są do współpracy z API typu RESTful. Do tworzenia dokumentacji zaleca się rozwiązania wspomagające implementację standardu OpenAPI z rodziny rozwiązań Swagger w szczególności Swagger Editor, oraz Swagger UI.



14.

Odstępstwa od standardu JSON API

Poniższe odstępstwa są umotywowane koniecznością odciążenia serwerów danych publicznych przez zmniejszenie potrzeby kopiowania całej bazy w celu uzyskania konkretnych odpowiedzi, lub ułatwienie korzystania z danych.

Rekomendacje dotyczące odstępstw od JSON API:

1) API musi wspierać wybór pól do dostarczenia klientowi (specyfikacja JSON API mówi, że jest to opcjonalne – sekcja Sparse Fieldsets).
2) API musi wspierać sortowanie pól tam, gdzie jest to logicznie uzasadnione rodzajem danych (specyfikacja JSON API mówi, że jest to opcjonalne – sekcja Sorting).
3) API musi wspierać filtrowanie (specyfikacja JSON API mówi, że jest to wysoce zalecane – sekcja Filtering).
4) API musi wspierać dokładnych informacji o błędach (specyfikacja JSON API mówi, że jest to opcjonalne – sekcja Error Objects).

Standard API

(interfejsu programistycznego aplikacji)

Załączniki

STANDARDY OTWARTOŚCI DANYCH

Spis treści

Załącznik nr 1.

Kategorie obecnych i potencjalnych odbiorców danych wykorzystujących API – szczegółowa charakterystyka grup użytkowników 44

Załącznik nr 2.

Architektura usług sieciowych REST 47

Załącznik nr 3.

Opis składni oraz elementów URI – definicje, szczegółowy opis składni oraz przykłady zastosowań 50

Załącznik nr 4.

Manipulowanie zasobami: filtrowanie, sortowanie, stronicowanie – zasady wraz z przykładami 55

Załącznik nr 5.

Metadane 60

Załącznik nr 6.

Mechanizm linków do nawigacji – objaśnienia i przykłady stosowania 63

Załącznik nr 7.

Wersjonowanie – zasady, metody oznaczania wersji wraz z przykładami oraz techniki zarządzania wersjami 65

Załącznik nr 8.

Obsługa zdarzeń HTTP – tabela grup kodów zgodnie z RFC 7231 oraz przykłady odpowiedzi i komunikatów 69

Załącznik nr 9.

Uwierzytelnianie i autoryzacja – opisy technologii i rozwiązań 71

Załącznik nr 1

Kategorie obecnych i potencjalnych odbiorców danych wykorzystujących API – szczegółowa charakterystyka grup użytkowników

Odbiorcami korzystającymi z danych publicznych (obecnie i docelowo) są osoby fizyczne, przedsiębiorstwa, jednostki administracji publicznej (centralnej i samorządu terytorialnego), organizacje pozarządowe, podmioty nauki, kultury i inni interesariusze. Dane publiczne mogą opisywać rzeczywistość w dowolnym aspekcie, a interesariusze mogą przetwarzać te dane na wiele sposobów. Kształtuje to metody korzystania z API oraz wpływa na wymagania przed nim stawiane. Dlatego na tym etapie ważne jest dokonanie podziału odbiorców danych na grupy wyodrębnione ze względu na charakterystykę ich potrzeb i sposobów wykorzystywania API, zamiast grupowania według wyżej wymienionych cech podmiotowych.

Większość użytkowników API będzie korzystać z produktów pochodnych opracowanych w oparciu o dane pochodzące z API (usług, aplikacji komputerowych, informacji przetworzonych, know-how) często nieświadomie. Jest to tzw. grupa użytkowników pośrednich, która nie korzysta wprost z API, ale czerpie z niego korzyści i może mieć wpływ na pożądany kierunek jego ewolucji.

Pozostałych użytkowników można podzielić na dwie grupy:

- użytkownicy okazjonalni (ad-hoc),
- użytkownicy profesjonalni.

Grupa użytkowników okazjonalnych (*ad hoc*) – stanowią ją użytkownicy, którzy będą korzystać z API doraźnie dla rozwiązania lub poznania zagadnień związanych np. z zadaniami w pracy, w szkole, na uczelni, często z ciekawości. Do grupy tej

należeć będą zarówno osoby fizyczne (obywatele), ale i przedsiębiorcy prowadzący działalność gospodarczą w różnych jej formach prawnych i organizacyjnych (spółki prywatne i publiczne, organizacje wirtualne, jednostki JST, start-upy), organizacje pozarządowe, jednostki naukowe, instytucje obszaru kultury oraz pozostałe jednostki administracji publicznej. Użytkownicy z tej grupy to osoby w znacznej mierze bez zaplecza technicznego, posiadające podstawowe kompetencje cyfrowe takie jak obsługa systemu plików, tworzenie dokumentów cyfrowych, obsługa skrzynki e-mail przez stronę internetową, działania w mediach społecznościowych oraz interaktywne przeglądanie stron internetowych.

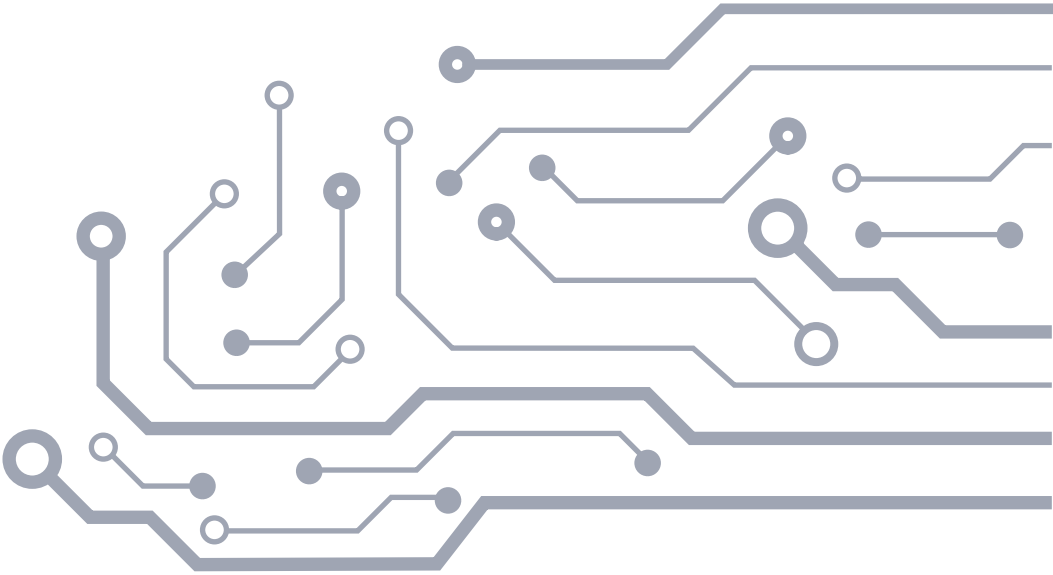
Do opisywanej grupy należą także innowatorzy lub wcześni naśladowcy. To oni w pierwszej kolejności zidentyfikują potrzebę korzystania z danych udostępnionych przez API i weryfikacji, na ile te dostępne zasoby będą przydatne do szybkiego rozwiązywania konkretnych zadań. Użytkownicy ci nie będą skłonni do przeznaczenia ani znacznych nakładów, ani zasobów (w szczególności czasu), by zgłębiać i eksperymentować z API. Dlatego powinni po wejściu w interakcję z API szybko uzyskać pozytywne efekty, które będą stanowiły wzmocnienie i zachętę do dalszego eksplorowania zasobów API. Jest to istotna cecha konieczna do uwzględnienia przy tworzeniu zarówno API, jak i jego dokumentacji. Dokumentacja API powinna na prostych przykładach obrazować przydatność danych publicznych w rozwiązywaniu codziennych zadań. Czynności techniczne, w szczególności związane z zachowaniem poprawności składni, które mogą użytkownikom nastręczać trudności, powinny przyjąć formę gotowych elementów – pasujących do siebie bloków funkcjonalnych.

Mimo, że użytkownicy ad-hoc stanowią grupę użytkowników nieprofesjonalnych, to jednak ta grupa jest bardzo ważna, gdyż zakłada się, że to właśnie ona będzie inicjować, a nierzadko decydować o zaangażowaniu API w instytucjach. Mając na uwadze, że świadomość potencjału oraz możliwości korzystania z danych publicznych jest jeszcze stosunkowo na niskim poziomie[1] (stosownie do wczesnej fazy rozwoju tego rodzaju źródeł informacji), to należy położyć odpowiedni nacisk na możliwość uzyskania szybkiej satysfakcji przez użytkowników tej grupy podczas korzystania z API. Ten aspekt może być decydujący dla powodzenia i popularności przygotowanego interfejsu.

Grupa użytkowników zaawansowanych – to druga grupa bezpośrednich użytkowników API. W odróżnieniu od użytkowników okazjonalnych, dysponują oni wiedzą techniczną i mają już sformułowane bezpośrednie cele i korzyści

[1] Jest to jedna z głównych barier przy korzystaniu zdanych publicznych. Zob. M. Możejewski, „Strategia, standaryzacja i pomiar jakości informacji – determinanty efektywnego ponownego wykorzystania informacji sektora publicznego", [w:] A. Gołębiowska, P.B. Zientarski (red.), „Ponowne wykorzystanie informacji sektora publicznego w administracji", Senat RP, Warszawa 2017, s. 65-90.

korzystania z API, gdyż podjęli już decyzję o chęci korzystania z niego. To oznacza, że użytkownicy ci są zdeterminowani do integracji usług API z tworzonymi nowymi lub modyfikowanymi systemami informatycznymi. Użytkowników tych cechuje potrzeba korzystania z danych w sposób przewidywalny. Oczekują dostępności, odpowiedniej jakości danych oraz wydajności mechanizmu API, który zaspokoi każde bieżące potrzeby (których często sami nie są w stanie określić z wyprzedzeniem). Zatem w grupie użytkowników zaawansowanych znajdą się także użytkownicy okazjonalni, którzy wypróbowali już API i określili swoje oczekiwania i cele w stosunku do API. Posiadają przy tym wiedzę techniczną, zaplecze techniczne oraz zasoby, które mają być wzbogacane w oparciu o dane zasilane z API. W odróżnieniu od użytkowników okazjonalnych użytkownicy profesjonalni posiadają też wiedzę nt. zaawansowanych narzędzi i form komunikacji z API. Użytkownikami profesjonalnymi w głównej mierze są deweloperzy. Są oni użytkownikami, którzy mają mało czasu, podchodzą do problemów zadaniowo i chcą szybko uzyskiwać satysfakcjonujące efekty.



Załącznik nr 2

Architektura usług sieciowych REST

Standard API jest oparty o standard REST. Ponadto referencyjnie wykorzystuje format reprezentacji danych zdefiniowany przez standard JavaScript Object Notation czyli JSON . Jako standard notacji JSON[2] przyjmuje się JSON Schema. Kontekst ułatwiający korzystanie z danych w sposób automatyczny jest generowany zgodnie ze standardem JSON-LD. Łączenie obiektów według zasad „linked data" zapewnione jest zgodnie ze standardem hydra:Link.

REST – (ang. *REpresentational State Transfer*) jest popularnym stylem architektonicznym stosowanym w celu budowy usług wymiany danych w sieci Internet takich jak API. API zbudowane w oparciu o zasady zdefiniowane przez REST zwane jest RESTful API. REST wykorzystuje protokół HTTP[3] z uwzględnieniem jego zalet i ograniczeń. Komunikaty REST wykorzystują statusy odpowiedzi HTTP, takie jak 2xx (żądanie obsłużone poprawnie), 4xx (komunikaty o błędzie po stronie klienta), czy 5xx (błędy po stronie serwera).

REST jest stylem zaproponowanym w doktoracie R. Fieldinga[4] z 2000 roku i stanowi odwzorowanie idei efektywnej dystrybucji treści w rozproszonym systemie hipertekstowym. Idea ta uwzględnia jednak istotne ograniczenia, które stanowią jednocześnie kluczowe wytyczne dla REST:

- Zasoby (ang. resource). W REST kluczowym elementem jest zasób. Zasób to każda porcja informacji, której postać może być określona. Przykładami zasobów są dokumenty, obrazy, informacje aktualne w określonym czasie,

[2] Schema JSON http://www.yaml.org/spec/1.2/spec.html#id2803231 [Dostęp 12.02.2018]

[3] Hypertext Transfer Protocol (HTTP).

[4] Rozprawa doktorska R. Fieldinga „Architectural Styles and the Design of Network-based Software Architectures", University of California 2010.

pogrupowane zasoby czy obiekty, pozwalające się w uproszczony sposób scharakteryzować.

- Zasób jest odwzorowaniem uproszczonych cech obiektu. Cechy te muszą być spełnione (odwzorowane) zgodnie z definicją obiektu. Tylko obiekt, który odpowiada definicji odwzorowania może być zasobem. Oznacza to, że porcja informacji staje się zasobem, gdy określone są wszystkie jej cechy określone w jego definicji. Zależność tę można zdefiniować za pomocą funkcji przynależności na podstawie warunku logicznego, który przyporządkowuje zasobowi w określonym czasie unikalny identyfikator lub określone wartości. Zasób może być określony w postaci niezmiennej (gdy przyjmuje wartości statyczne), zmiennej (gdy przyjmuje wartości dynamiczne w czasie), ale może też być pusty (brak przyjętych wartości). Jedyny element, który jest wymagany i jest stały w określeniu zasobu, to semantyka mapowania tj. struktura jego elementów znaczeniowych. Nawet w przypadku zmienionych, braku lub tożsamych wartości pozwala ona na odróżnienie jednego zasobu od drugiego.
- Komunikacja w modelu klient-serwer. W wymianie informacji opartej o REST istnieje wyraźna granica między klientem i serwerem. Domeną klienta jest interfejs użytkownika i generowanie zapytań. Domeną serwera jest obsługa żądań klientów, w szczególności zarządzanie przechowywaniem i dostępem do danych, obciążeniem i bezpieczeństwem. Elementy te mogą być rozwijane po stronie klienta i serwera niezależnie od siebie tak długo, jak komunikacja między stronami odbywa się poprzez wspólnie jednoznacznie rozumiany protokół.

  Samoopisujące się komunikaty. Każdy komunikat wyposażony jest w odpowiednie informacje, które pozwalają na zarządzanie tym zasobem. Oznacza to, że komunikat oprócz treści zawiera także metadane pozwalające umieścić ją w kontekście i prawidłowo zinterpretować.
- Bezstanowość (ang. *stateless*) – oznacza rozpatrywanie zapytań niezależnie od innych (tego samego lub innego użytkownika). Cecha ta związana z wymianą informacji w modelu klient-serwer powoduje, że nie są akumulowane informacje o poprzednich transakcjach klienta, co pozwala na odciążenie serwera, jednak wymaga dodatkowych mechanizmów, jeśli podczas wymiany informacji potrzebne są dodatkowe informacje na temat użytkownika lub jego żądań[5]. Wszystkie operacje klient-serwer powinny być bezstanowe, a zarządzanie stanem powinno odbywać się po stronie klienta.

[5] Takie informacje mogą być dostarczane poprzez mechanizm ciasteczek, sesji lub parametrów w nagłówku HTTP.

- Buforowanie reprezentacji – w kontekście REST odpowiedzi serwera mogą być po stronie klienta przechowywane w buforze (cache), tak aby mogły być użyte podczas realizacji kolejnych zapytań bez konieczności pobierania ich pełnych wyników. Jednak, aby uniknąć odczytu przez klienta nieodpowiednich lub nieaktualnych danych, pobierane treści powinny być oznaczane jako możliwe do buforowanie oraz na jakich warunkach (głównie czasu ważności treści). Zaleca się, aby wszystkie zasoby mogły być buforowane, chyba, że taka możliwość zostanie wprost wyłączona. Prawidłowe określenie tych parametrów pozwala eliminować kolejne interakcje klienta i serwera. Buforowanie wpływa zatem pozytywnie na szybkość dostarczania treści do odbiorcy, pozwala korzystać z treści, nawet gdy serwer jest chwilowo niedostępny, oraz zmniejsza obciążenie serwera. Oznaczenie zasobów informacyjnych powinno być określane przez ich dostawców na podstawie odrębnej polityki określającej wytyczne dla warunków buforowania.
- Zastosowanie jednolitego interfejsu i unikalna identyfikacja zasobów – zasoby w systemach REST powinny być jednoznacznie oznaczane i identyfikowane za pomocą adresu URL. W przypadku Standardu API, działania wykonywane na tych zasobach (np. PUT, GET) powinny być wykonywane za pomocą poleceń protokołu HTTP z włączonym szyfrowaniem (HTTPS).
- Enkapsulacja systemów warstwami (ang. *Layered System*), czyli w uproszeniu przesłonienie systemów zewnętrznymi powłokami, które mają na celu wystandaryzowaną wymianę informacji. Zaletą takiego podejścia jest to, że każdy system pozostaje czarną skrzynką, a o jego przydatności decyduje tylko oddzielający go interfejs. Natomiast szczegóły wewnętrzne systemu serwera pozostają poza percepcją klientów.
- Kod na żądanie (ang. Code-On-Demand), opcjonalnie – serwery są zdolne do rozszerzania lub dostosowania funkcjonalności, z jakiej korzysta klient poprzez przesyłanie mu logiki, która może być wykonywana. Przykładami takiego działania może być przesyłanie klientom apletów Java lub kodu JavaScript, który wykonuje się u klienta. Jest to cecha, która w połączeniu z pozostałymi prowadzi do zwiększanie skalowalności, adaptowalności, przenośności i modyfikowalności rozwiązań budowanych w oparciu o REST.

Załącznik nr 3

Opis składni oraz elementów URI – definicje, szczegółowy opis składni oraz przykłady zastosowań

Definicje związane z oznaczeniem zasobów:

Schemat (ang. *authority*) – komponent URI, który definiuje protokół wymiany informacji (np. HTTP) oraz przestrzeń.

Zasób główny (ang. *docroot*) – zasób, który jest nadrzędny wobec pozostałych w API w modelu REST.

Dokument (ang. *document*) – reprezentacja pojedynczego wystąpienia zasobu.

Kolekcja (ang. *collection*) – grupa zasobów po stronie serwera.

Magazyn (ang. *store*) – repozytorium zarządzane przez klienta.

Kontroler (ang. *controller*) – opis logiki proceduralnej.

CRUD – akronim od nazw czterech podstawowych funkcji operowania na danych w bazach danych: *create* (utwórz), *retrieve* (odczytaj), *update* (aktualizuj) and *delete* (usuń).

„Bezpieczne metody" HTTP – metody GET i HEAD, które umożliwiają uzyskiwanie reprezentacji zasobów, ale nie pozwalają na modyfikację i usuwanie zasobów na serwerze.

Zapytanie (ang. *query*) – fragment URI występujący za ścieżką, ale przed opcjonalnym fragmentem.

URI (z ang. *Uniform Resource Identificator*) jest unikalnym identyfikatorem zasobów, stosowanym zgodnie z RFC 1736[6], RFC 1737[7] oraz ustandaryzowanym zgodnie z RFC 3986[8]. Podstawowe oznaczenie URI przyjmuje następująca postać:

```
URI = <schemat> „:” <część hierarchiczna> [ „?” <pytanie> ]
                  [ „#” <fragment> ]
```

Ogólna postać URI zakłada występowanie schematu (np. protokołu), określonej hierarchii (np. domeny i/lub odwołania lokalnego), zapytań oraz wskazania wybranych fragmentów. Część hierarchiczna może zawierać odwołanie do domeny podmiotu i/lub ścieżki lokalnej. Elementy **<pytanie>** oraz **<fragment>** są składowymi, które nie muszą być podane. Element <pytanie> zwany jest też kwerendą.

1) Domena w URI powinna wskazywać jednoznacznie na właściciela API. Domena ta powinna udostępniać funkcjonalność API poprzez subdomenę „api”. Odstępstwa od tej reguły są dopuszczalne w sytuacji, gdy poddomena API jest już zajęta do innych celów.

   Przykład:

   ```
   https://api.danepubliczne.gov.pl
   ```

   Powyższe zalecenie wynika z zastosowania tzw. dobrej praktyki przy udostępnianiu danych poprzez API. Mimo, że popularne są także odmienne podejścia (np. udostępnianie zasobów API w ścieżce) to wydzielenie zasobów w zaproponowany sposób jest szczególnie stosowane przez liczne światowe API udostępniające informacje sektora publicznego[9]. Takie rozwiązanie poprawia również czytelność i rozpoznawalność zasobów API. Dlatego warto tę konwencję utrzymać. Z drugiej strony API dostępne w wydzielonej subdomenie pozwala na

[6] **https://www.ietf.org/rfc/rfc1736.txt** [Dostęp 29.03.2018]

[7] **https://www.ietf.org/rfc/rfc1737.txt** [Dostęp 29.03.2018]

[8] **https://www.ietf.org/rfc/rfc3986.txt** [Dostęp 12.02.2018]

[9] Światowe zasoby danych dostępne poprzez API w subdomenie: **api.data.gov**, **api.nasa.gov**, **data.norge.no**, **api.trafikinfo.trafikverket.se** [Dostęp 12.02.2018]

zwiększenie bezpieczeństwa poprzez możliwość izolacji ustawień (per-domena) oraz ułatwia zarządzanie przyszłą migracją. Zalecenie takie wynika również z literatury przedmiotu, która określa, że stosując wytyczne REST dla API powinno udostępniać się API w subdomenie[10].

W uzasadnionych przypadkach dopuszczalne jest udostępnianie API w głównej domenie identyfikującej podmiot zarządzający API, pod adresem /api.

2) API może zostać wyposażone w portal dla developerów, który zawiera podstawową dokumentację API – lub odnośnik do niej – oraz informację o ewentualnych limitach nałożonych na API oraz możliwości uzyskania klucza aplikacji (API-Key). Portal powinien być integralna częścią API i być dostępny poprzez subdomenę „dev". Odstępstwo od tej reguły jest dopuszczalne w sytuacji, gdy poddomena dev jest już zajęta do innych celów.

   W przypadku braku oddzielnego serwisu dla developerów, strona w domenie API musi zawierać informacje o limitach w wykorzystaniu API oraz o możliwości uzyskania klucza aplikacji.

   Przykład:

   ```
   https://dev.danepubliczne.gov.pl
   ```

   Wytyczne, podobnie jak w przypadku punktu pierwszego, wynikają z zastosowania przyjętej dobrej praktyki do zapewniania wsparcia dla deweloperów w jednym wydzielonym miejscu. Literatura przedmiotu także w tym wypadku określa, że miejsce to powinno być oznaczone wydzieloną subdomeną „devel", „dev" lub „developer"[11].

   W uzasadnionych przypadkach dopuszczalne jest udostępnianie informacji dla programistów w głównej domenie identyfikującej podmiot zarządzający API, pod adresem /dev lub /devel.

3) Znak „/" (ukośnik, ang. forward slash) służy oznaczeniu zależności hierarchicznej miedzy zasobami.

[10] Zob. s. 14, M. Masse, *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*, O'Reilly 2012.

[11] Tamże s. 14.

Przykład:

```
https://api.danepubliczne.gov.pl/cepik/pojazdy/
zarejestrowane/zachodniopomorskie/szczecin
```

4) Stosowanie znaku „/” (ukośnik, ang. forward slash) na końcu URI nie jest zalecane.

Przykład:

~~`https://api.data.gov.pl/cepik/pojazdy/zarejestrowane/zachodniopomorskie/szczecin/`~~

5) Oznaczanie ścieżki URI z użyciem małych liter.

Przykład:

~~`https://API.danepubliczne.gov.pl/lista-zasobow`~~

oraz

```
https://api.danepubliczne.gov.pl/lista-zasobow
```

URI zapisane wielkimi i małymi literami w powyższym przykładzie w praktyce są tożsame, gdyż zgodnie z RFC 3986 nie jest rozróżniana wielkość liter w elemencie reprezentowanym przez domenę i subdomenę. Ponieważ wytyczne w stosunku do nazewnictwa zasobów wskazują konieczność używania małych liter to dla zachowania jednoznaczności także dla URI formułuje się taką zasadę.

6) W URI nie stosuje się rozszerzeń plików oraz nazw formatów danych. Informacje te są przekazywane przez nagłówki HTTP lub w parametrach.

Przykład:

```
https://api.danepubliczne.gov.pl/cepik/pojazdy-zarejestrowane/zachodniopomorskie/szczecin.json
```

Używanie w URI nazwy użytkownika z hasłem może stanowić poważną lukę w bezpieczeństwie i z tego powodu nie jest dozwolone. Informacja o wykorzystanym kluczu aplikacji może zostać przekazana jako parametr żądania lub w nagłówkach HTTP.

Załącznik nr 4

Manipulowanie zasobami: filtrowanie, sortowanie, stronicowanie – zasady wraz z przykładami

Z założenia, szereg klienckich aplikacji będzie korzystać z zasobów API automatycznie. Aplikacje takie rzadko realizują zadania wymagające pobrania z API zbioru danych w pełnym przekroju informacyjnym oraz pełnej zawartości rekordów. Znacznie częściej potrzebują odpowiednio spreparowanego, niewielkiego wycinka zamiast pełnego zakresu dostępnych danych. Wycinek ten w stosunku do oryginalnego zbioru może być filtrowany względem zakresu pobieranych danych, w kontekście określonych kryteriów, porcji danych pobieranej za jednym razem oraz w odpowiednim uporządkowaniu. Dlatego założono, że API udostępnia funkcjonalność filtrowania, sortowania i stronicowania zasobów, które określane będzie mianem manipulacji danymi.

Zapytania i odpowiedzi systemu realizowane zgodnie ze standardem JSON API w wersji 1.0 lub nowszej[12]. W odpowiedziach zaleca się uzupełnienie JSON API o JSON-LD[13] w celu dodania kontekstu koniecznego do automatycznego przetwarzania danych API.

Interakcja z API w zakresie filtrowania, sortowania, stronicowania wybranych zasobów oparta jest o mechanizm URI, który zawiera sekcję **<pytanie>.** Jest to element nieobowiązkowy w URI, lecz niezbędny w przypadku potrzeby manipulacji zakresem lub strukturą danych przed pobraniem.

[12] JSON API – Specification – **http://jsonapi.org/format/** [dostęp: 31.03.2018]

[13] JSON for Linking Data – **https://json-ld.org/spec/latest/json-ld/** [dostęp: 3.04.2018]

```
URI = <schemat> „:” <część hierarchiczna> [ „?” <pytanie> ]
                    [ „#” <fragment> ]
```

Podstawowe zasady manipulacji danymi zostaną przedstawione za pomocą kolejnych przykładów.

Punktem odniesienia do wykonywania operacji manipulowania danymi jest pełna lista zasobów, czyli tzw. kolekcja, uzyskiwana za pomocą następującego odwołania do zasobu głównego:

```
GET /
```

W wyniku uzyskiwana jest odpowiedź, która jest reprezentacją stanu zawierającą listę wszystkich zasobów.

Natomiast po wykonaniu kodu z przykładu:

```
GET /cepik/pojazdy
```

uzyskiwana jest reprezentacja stanu zawierająca listę elementów zasobu „pojazdy”.

Filtrowanie zasobów służy do ograniczania wyników jedynie do tych, które spełniają zadane warunki. W rezultacie kolekcję ogranicza się jedynie do listy zawierającej zasoby odpowiadające kryteriom przekazanym w URI w części **<pytanie>**.

```
GET /?filter=zasoby-danych:cepik
```

W wyniku wykonania powyższego zapytania uzyskiwana jest reprezentacja stanu zawierająca zasób danych o nazwie „cepik”.

```
GET /cepik/pojazdy?filter=rodzaj:osobowe
```

W wyniku uzyskiwana jest reprezentacja stanu zawierająca listę pojazdów, które dla wymiaru „rodzaj” spełniają warunek „osobowe”.

W przypadku filtrowania po więcej niż jednym kryterium, do rozdzielenia warunków korzysta się z operatora **„&”**:

```
GET /cepik/pojazdy?filter=rodzaj:osobowe&kolor:czerwony
```

W wyniku można ograniczyć liczbę zwracanych pól jedynie do wybranych: na przykład przebiegu, roku oraz przy zawężeniu wyników tylko do pojazdów osobowych:

```
GET /cepik/pojazdy?filter=rodzaj:osobowe&fields[pojazdy]=przebi
                        eg,rok-produkcji
```

Sortowanie zasobów służy do szeregowania elementów rosnąco lub malejąco według określonego kryterium lub kryteriów. Domyślnie zasoby powinny być sortowane rosnąco według zadanego kryterium.

```
GET /cepik/pojazdy?sort=rok-produkcji
```

Wynik: sortowanie wyników za pomocą jednego kryterium, w porządku domyślnym (rosnąco) w kolekcji „pojazdy”.

```
GET /cepik/pojazdy?sort=rok-produkcji,przebieg
```

Wynik: sortowanie wyników za pomocą więcej niż jednego kryterium, w porządku malejącym.

```
GET /cepik/pojazdy?sort=-rok-produkcji,przebieg
```

Wynik: sortowanie za pomocą więcej niż jednego kryterium, w porządku malejącym.

Uwaga: aby wyznaczyć porządek malejący należy przed wybrane kryterium wstawić znak **„-”** (minus). W odpowiedzi zostanie zwrócona lista zawierająca pojazdy uszeregowane od najmłodszego do najstarszego, przy czym pojazdy z tego samego roku zostaną posortowane rosnąco według przebiegu.

Jeśli serwer nie obsługuje danego rodzaju sortowania dla zadanego parametru serwer powinien zwracać błąd. Jeśli parametr „sort” nie został wskazany serwer używa domyślnego sortowania dla zwracanych elementów.

Stronicowanie zasobów służy do ograniczania wyników zapytania do określonej liczby elementów zwanych stroną. Stronicowanie oparte jest o dwa parametry:

page – oznacza numer strony,

limit – oznacza liczbę elementów wyświetlanych na jednej stronie.

```
GET /cepik/pojazdy?page=5&limit=48
```

Oznacza wyświetlenie 48 elementów na piątej stronie.

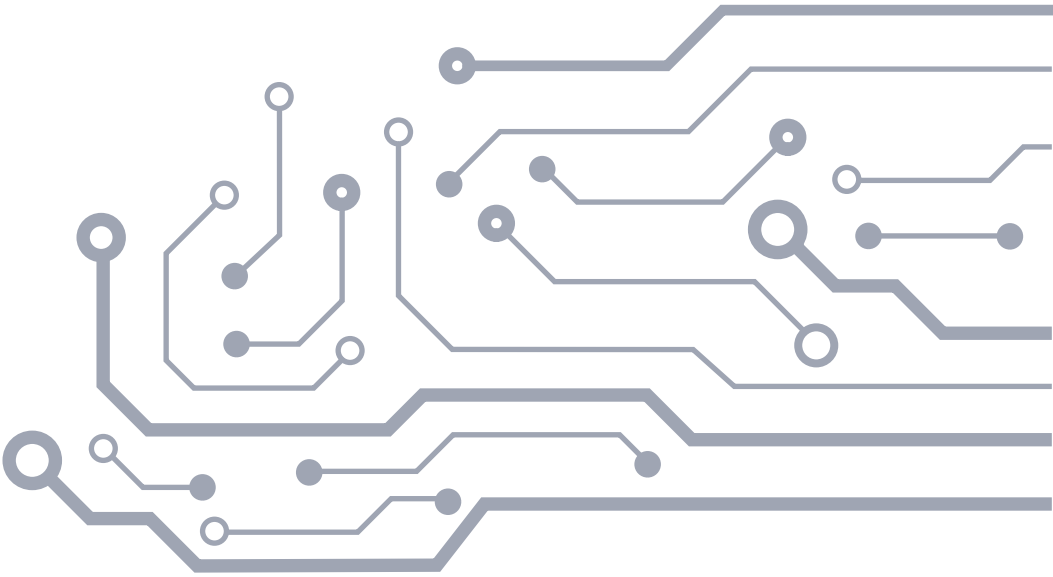
Podsumowując powyższe kod obrazujący przykład najbardziej złożonego ze wszystkich powyższych warunków będzie miał postać następującą:

```
GET /cepik/pojazdy?filter=rodzaj:osobowe&sort=
        -rok,przebieg&page=5&limit=48
```

W przypadku, gdy przygotowanie odpowiedzi zajmie więcej czasu, serwer powinien odpowiedzieć komunikatem HTTP o numerze 202, ze wskazaniem zasobu informującego o szacunkowym czasie pozostałym do końca realizacji żądania.

```
HTTP /1.1 202 Accepted
Location: /queue/[id_żądania]
```

Do momentu przygotowania danych, pod adresem /queue/[id_żądania] znajduje się informacja o przygotowywaniu odpowiedzi wysyłana ze statusem o numerze 200. W momencie, gdy zasób jest gotowy do przekazania odbiorcy, pod tym adresem znajduje się odpowiedź o numerze 303 i lokalizacja odpowiedzi na pierwotne zapytanie. Po upływie okresu, po którym przygotowane w ten sposób dane są usuwane z serwera, zapytanie o lokalizację docelową zwraca kod błędu 410.



Załącznik nr 5

Metadane

Zasoby udostępniane przez API wyposażane są w metadane, stanowiące część odpowiedzi na zapytanie o reprezentację zasobu. Metadane to inaczej dane o danych, które opisując zasoby ułatwiają ich zrozumienie oraz zarządzanie nimi. Są szczególnie istotne w procesie maszynowego przetwarzania informacji, dlatego istotne jest wykorzystanie rodzajów metadanych tam, gdzie tylko jest to możliwe. W przypadku Standardu API, metadane muszą uwzględniać lokalne wytyczne określone przez PODP oraz rozporządzenie Rady Ministrów z dnia 12 marca 2014 r. w sprawie Centralnego Repozytorium Informacji Publicznej (Dz. U. poz. 361). Dlatego w poniższej tabeli przedstawiono minimalny zestaw metadanych dla zasobu z założeniem, że może zostać rozbudowany przez twórcę konkretnej implementacji API gdy będzie to konieczne. Metadane niewykorzystane w konkretnym przypadku (np. schema:expires) są pomijane.

Nazwa	Oznaczenie	Znaczenie
Nazwa zasobu	dc:title	Nazwa zasobu informacyjnego, która umożliwia jego identyfikację (ew. nazwa zasobu z elementami charakterystycznymi). http://dublincore.org/documents/dcmi-terms/#terms-title

Nazwa	Oznaczenie	Znaczenie
URL zasobu	schema:url	Identyfikator lokalizacji do pobrania zasobu informacyjnego w przypadku, gdy stanowi osobny plik (URL). http://schema.org/url
Identyfikator dostawcy zasobu	schema:provider	Opis dostawcy zasobu informacyjnego. http://schema.org/provider
Data publikacji zasobu	schema:datePublished	Data i czas publikacji zasobu, z dokładnością do jednej sekundy. Data i czas zgodnie ze standardem ISO-8601. https://schema.org/datePublished
Data modyfikacji zasobu	schema:dateModified	Data ostatniej modyfikacji zasobu z dokładnością do jednej sekundy. Data i czas zgodnie ze standardem ISO-8601. https://schema.org/dateModified
Data zakończenia publikacji zasobu	schema:expires	Data określająca, do kiedy zasób jest dostępny w API – jeśli zasób będzie dostępny przez określony czas. https://schema.org/expires
Opis zasobu	dc:description	Opis zasobu informacyjnego. http://dublincore.org/documents/dcmi-terms/#terms-description
Słowa kluczowe	schema:keywords	Słowa kluczowe, czyli zestaw słów lub wyrażeń zwięźle opisujących zasób informacyjny. https://schema.org/keywords

Nazwa	Oznaczenie	Znaczenie
Zakres czasowy danych	schema:temporalCoverage	Informuje, że zbiór zawiera dane za konkretne okresy. https://schema.org/temporalCoverage
Język zasobu	dc:language	Określa język zasobu. http://dublincore.org/documents/dcmi-terms/#terms-language
Częstotliwość aktualizacji	sy:updatePeriod sy:updateFrequency sy:updateBase	Informuje o tym, jaki jest interwał aktualizacji zasobu, np.: co miesiąc, co rok, co godzinę, co 10 sekund Zdefiniowana w purl.org: http://web.resource.org/rss/1.0/modules/syndication/ Nie występuje w przypadku danych stałych lub o nieprzewidywalnym czasie aktualności.
Kodowanie	schema:contentType	Określa informację o rodzaju zasobu i jego kodowaniu. https://schema.org/contentType
Licencja	dc:rights	Określenie rodzaju licencji w przypadku wprowadzenia ograniczeń w wykorzystaniu zasobu. http://dublincore.org/documents/dcmi-terms/#terms-rights
Katalog	schema:isPartOf	Określenie przynależności do grupy np. zasobów. https://schema.org/isPartOf

Powyższa tabela zawiera minimalny zestaw metadanych, który może być rozszerzany. Ponadto szczególną rolę odgrywa właściwie dobrany i bezbłędny zestaw metadanych określany przez dostawcę, gdyż są to informacje najczęściej wykorzystywane do precyzyjnego manipulowania danymi w ramach zasobów.

Załącznik nr 6

Mechanizm linków do nawigacji – objaśnienia i przykłady stosowania

W odpowiedzi na przesyłane przez użytkowników zapytania, API udostępnia stany zawierające odpowiednie zasoby informacyjne. Jednak nie wszyscy użytkownicy potrzebują w odpowiedzi otrzymywać całe zasoby. Większość aplikacji użytkowników wykorzystujących z API potrzebuje w danym momencie często jedynie niewielkiej porcji informacji odfiltrowanej z zasobu. Takie informacje są filtrowane za pomocą dodatkowych warunków zapytania zawartych w URI. Jednak przesłanie użytkownikowi samego stanu zawierającego odfiltrowany zasób nie jest wystarczające. Dlatego Standard API zapewnia funkcjonalność linków do nawigacji. Polega ona na dołączaniu w ciele zasobu poniższych linków (oznaczanych jako „links”), zgodnie z JSON API 1.0 lub późniejszym:

Nazwa linku	Objaśnienie
prev	Link do poprzedniego powiązanego zasobu.
self	Link do bieżącego zasobu.
next	Link do następnego powiązanego zasobu.
last	Link do ostatniego powiązanego zasobu.
first	Link do pierwszego powiązanego zasobu

Przykład:

```
{
  „links”: {
    „prev”: „http://api.danepubliczne.gov.pl/cepik/
    pojazdy?start=15&limit=5”,
    „self”: „http://api.danepubliczne.gov.pl/cepik/pojazdy”,
    „next”: „http://api.danepubliczne.gov.pl/cepik/
    pojazdy?start=25&limit=5”,
    „first”: „http://api.danepubliczne.gov.pl/cepik/
    pojazdy?start=1&limit=5”
    „last”: „http://api.danepubliczne.gov.pl/cepik/
    pojazdy?start=50&limit=5”,
  },
}
```

Załącznik nr 7

Wersjonowanie – zasady, metody oznaczania wersji wraz z przykładami oraz techniki zarządzania wersjami

Wersjonowanie jest jedną z najistotniejszych kwestii do rozważenia przy tworzeniu API. Od tego jak wersjonowanie zostanie zaplanowane i zaimplementowane zależy w przyszłości nie tylko wygoda, ale i możliwość korzystania z API oraz dostępność jego zasobów. Wiele organizacji udostępniających API nie udoskonala go później. Przyczyną często jest nie tyle brak pomysłów na zmiany, co niemożność ich dokonania, gdyż w API nie przewidziano i nie uwzględniono jego ewolucyjnego charakteru.

Wśród twórców API przeważają następujące podejścia do wersjonowania API:

- w adresie (np. https://api.danepubliczne.gov.pl/api/v1/[…])
- w parametrze (np. https://api.danepubliczne.gov.pl/cepik/pojazdy?v=2
- w nagłówku Accept zapytania HTTP (Accept: application/danepubliczne.v1+json)
- przez dodatkowy nagłówek zapytania HTTP (api-version: 1)

Wersjonowanie w adresie (wariant pierwszy) w Standardzie API jest obowiązkowe. Ze względu na przydatne cechy pozostałych rozwiązań, ijednocześnie względną prostotę zaimplementowania wszystkich jednocześnie, dopuszczone jest umożliwienie korzystania dodatkowo z innych wariantów przy formułowaniu zapytania przez klienta.

Przykład czterech równoważnych zapytań:

- GET /v2/cepik/pojazdy
- GET /cepik/pojazdy?v=2

- GET /cepik/pojazdy

  Accept: application/vnd.kowalski.user+json; version=2

- GET cepik/pojazdy
  api-version: 2

W przypadku braku wskazania wersji API przez klienta, system powinien przyjmować wykorzystanie najnowszej stabilnej wersji API.

Oznaczenie wersji

Oznaczenie wersji składa się z trzech liczb oddzielonych kropkami, na przykład:

1.2.5

gdzie kolejna pozycja oznacza rodzaj wersji: MAJOR, MINOR, PATCH, oznaczające:

- wersja MAJOR, gdy dokonuje się zmian niekompatybilnych z poprzednią wersją API,
- wersja MINOR, gdy dodaje się nową funkcjonalność, która jest kompatybilna z poprzednimi wersjami,
- wersja PATCH, gdy naprawia się błąd nie zrywając kompatybilności z poprzednimi wersjami.

Dodając nowe funkcjonalności i linki należy dążyć do wstecznej kompatybilności, która nie będzie zakłócała funkcjonalności, z których korzystają dotychczasowi użytkownicy i ich aplikacje. Jeśli kompatybilność wsteczna zostanie zakłócona należy zwiększyć wersję MAJOR.

Do oznaczenia i wywoływania obsługi wersją o konkretnym numerze będzie mieć znaczenie tylko pierwszy człon (MAJOR) wersji, a pozostałe (MINOR i PATCH) będą ignorowane. API powinno wyświetlać numer aktualnie używanej wersji co najmniej w dedykowanej metodzie, oraz informację o spodziewanym terminie wyłączenia aktualnej wersji, jeśli taki jest znany.

Przykład:

```
GET /version
```

Odpowiedź:

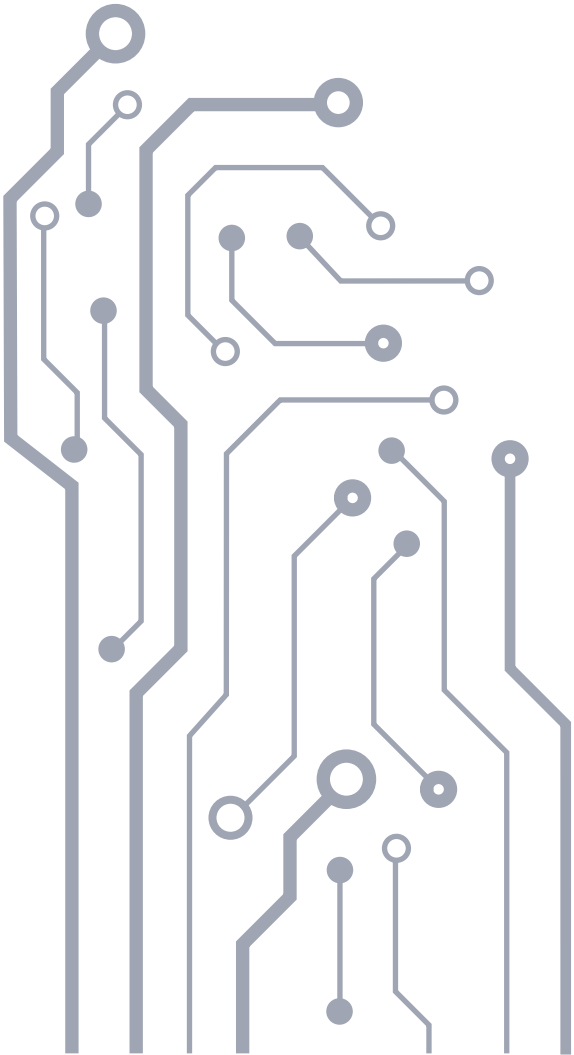
```
{
  „api-version”: {
    „major”: „1”,
    „minor”: „11”,
    „patch”: „5”,
    „date-mod”: „2018-05-20 12:15:30”,
    [opcjonalnie] „deprecated”: „2020-05-20”,
  },
}
```

Odwołanie do wersji API może być wykonane poprzez parametr URI zawierający element MAJOR wersji. Jeśli zawiera pozostałe elementy oznaczenia semantycznego wersji, są one ignorowane.

Zarządzanie zmianami wersji

Należy jednak pamiętać, że wersjonowanie nie jest jedynie kwestią techniczną. Istota modyfikacji API tkwi w celu i zakresie zmian. Zmiany, które są pożądane mogą oddziaływać na API tak dalece, że aplikacje użytkowników po wprowadzeniu zmiany stracą dostęp do dotychczasowych zasobów. Oprócz zarządzania zmianą istotna jest komunikacja z użytkownikami oraz przewidywalność API. Dlatego rozwiązania organizacyjne towarzyszące API powinny zapewniać możliwość komunikowania użytkownikom API zakresu planowanych i wprowadzanych zmian oraz dostarczania informacji nt. przewidywanego okresu działania danej wersji i wsparcia dla API. Spełnienie tego pozwala budować przewidywalne usługi API. Dlatego przed planowaną zmianą wersji należy oznaczyć wersję bieżącą statusem *„deprecated”* (przestarzałe), który poinformuje dotychczasowych użytkowników, aby nie podejmowali działań w oparciu o tę wersję. Należy zakomunikować ten fakt dotychczasowym użytkownikom na stronie z opisem API oraz w opisie aktualnej wersji dostępnym pod adresem /version, a także określić okres wsparcia, zakończenia wsparcia i zakończenia funkcjonowania bieżącej wersji. Ogłaszany okres zakończenia funkcjonowania wersji nie powinien być krótszy niż dwa lata – chyba, że istnieje uzasadniona potrzeba skrócenia tego czasu – i należy go dostosować

do potrzeb kluczowych użytkowników. W chwili komunikacji zmian użytkownikom powinna być już dostępna dokumentacja dla nowej wersji API, tak aby mogli oni rozpocząć prace nad nowym interfejsem. Po upłynięciu terminu wyłączenia API, zapytanie nawiązujące do jego nieaktualnej wersji będzie zwracać błąd HTTP 410 (Gone) w przypadku wersjonowaniu API w URL lub 406 w przypadku innych metod, natomiast w ciele błędu powinny się znaleźć objaśnienia na temat wyłączenia API oraz adres.



Załącznik nr 8

Obsługa zdarzeń HTTP – tabela grup kodów zgodnie z RFC 7231 oraz przykłady odpowiedzi i komunikatów

API będące tzw. RESTful API opera się o HyperText Transfer Protocol (HTTP), włączając w to wybrane metody, kody i nagłówki odpowiedzi. Standard API korzysta głównie z metody GET. W Standardzie API zakłada się niewykorzystywanie metod umożliwiających modyfikację zasobów. HTTP dostarcza pięć kategorii oraz ponad 30 kodów odpowiedzi. Jednak nie wszystkie kody będą w API przydatne, dlatego Standard API przewiduje obsługę wybranych kodów. Poniżej przedstawiono grupy kodów i ich znaczenie[14].

Tabela grup kodów

Grupa kodów	Znaczenie	Objaśnienie
1xx	Informacyjne	Komunikaty transferu na poziomie protokołu.
2xx	Powodzenie operacji	Wskazuje na powodzenie realizacji żądania klienta.
3xx	Przekierowanie	Informuje, że do ukończenia żądania klient musi podjąć dodatkową akcję.
4xx	Błąd klienta	Wskazanie błędu leżącego po stronie klienta
5xx	Błąd serwera	Wskazanie błędu, za który odpowiada serwer.

[14] Na podstawie RFC 7231.

Przykład odpowiedzi w ciele HTTP z informacja o błędach:

```
{
  „errors”: [
    {
      errorResult: „Przepraszamy, zasób nie istnieje”,
      errorReason: „Nie znaleziono pojazdu w bazie”,
      errorSolution: „Id pojazdu powinno być liczbą.”
      errorCode: 25,
      errorHelp: „http://dev.danepubliczne.gov.pl/blog/api/2/
      help/error25”
    }
  ]
}
```

Przykład komunikatu errorSolution dla kategorii błędów 4xx zwracający odpowiednią informację w przypadku przesłanej przez klienta błędnej wartości parametru:

- zakres parametru
- typ parametru
- listę dopuszczonych wartości (jeśli ustalona)
- zapis poprawnego żądania z wykorzystaniem parametru klienta.

Przykład komunikatu dla błędu 404:

```
{
  „errors”: [
    {
      „errorResult”: „Zasób nie może być pobrany”,
      „errorReason”: „Nie znaleziono zasobu w podanej
      reprezentacji”,
       „errorSolution”: „API wersja 1 jest już niedostępna.
      Konieczna migracja do API w wersji 2.
      „errorCode”: 95,
      „errorHelp”: „http://dev.danepubliczne.gov.pl/blog/api/2/
      help/error95”
    }
  ]
}
```

Załącznik nr 9

Uwierzytelnianie i autoryzacja – opisy technologii i rozwiązań

Systemy implementujące API mogą korzystać z systemu uwierzytelniania użytkowników. Nie jest dopuszczalne zamknięcie API dla użytkowników nieuwierzytelnionych, dozwolone jest wprowadzenie dla nich dodatkowych ograniczeń w korzystaniu z API.

Uwierzytelnianie użytkownika może być przeprowadzane na wiele sposobów i związane jest z identyfikacją, autentykacją i autoryzacją (patrz: Słownik pojęć).

Autentykacja może być realizowana za pomocą metody Basic Auth (RFC 2617). Jest to prosty schemat interakcji, w którym podaje się identyfikator użytkownika i hasło, które wcześniej użytkownik uzyskuje rejestrując konto na powiązanej stronie.

Podczas wszystkich operacji związanych z rozpoznawaniem użytkownika, konieczne jest korzystanie z protokołu HTTPS. Niedopuszczalne jest korzystanie z metod pozbawionych szyfrowania.

W przypadku konieczności autoryzowania poszczególnych użytkowników, zaleca się korzystanie z metody OAuth 2.0 (RFC 6749), (RFC 6750). OAuth 2.0 jest wyposażony w protokół, którego działanie ma następujący przebieg:

1) Klient wysyła do właściciela zasobów żądanie autoryzacyjne. Żądanie autoryzacyjne może być wysłane również metodą pośrednią. W tej metodzie żądanie wysyłane jest poprzez serwer autoryzacji jako pośrednika. Jest to metoda preferowana.

2) Właściciel zasobu wysyła do klienta uprawnienie do autoryzacji. Uprawnienie do autoryzacji zależy od używanej przez klienta metody do żądania autoryzacyjnego oraz metod wspieranych przez serwer autoryzacji.

3) Klient wysyła do serwera autoryzacji uprawnienie do autoryzacji, uwierzytelnia się i żąda tokena.

4) Serwer autoryzujący uwierzytelnia klienta, waliduje uprawnienie do autoryzacji i wysyła do klienta token autoryzujący, gdy uprawnienie i uwierzytelnienie zakończą się pomyślnie.

5) Klient przesyła do właściciela zasobów żądanie dostępu do zasobów oraz token autoryzujący.

6) Właściciel zasobu waliduje token i jeśli to się powiedzie, przesyła klientowi chroniony zasób.

Zaletą takiego podejścia jest unikalna identyfikacja czynności wykonywanych przez konkretnego klienta na zasobach API. W OAuth zewnętrzne aplikacje nie mają styczności z danymi uwierzytelniającymi użytkowników. Dodatkowo tokeny mogą ekspirować po określonym czasie, co pozwala na automatyczne wygaszanie uprawnień klienta API, który w przypadku wygaśnięcia może przejść przez procedurę odświeżenia tokena. W OAuth jeden serwer autoryzacyjny chroni wiele zasobów. Prowadzi to do redukcji kont zakładanych przez użytkowników i posługiwania się w nich często tymi samymi hasłami. Powoduje też, że użytkownicy chętniej podejmują integrację, gdyż jej rozpoczęcie nie wymaga realizowania dodatkowych czynności związanych z zakładaniem nowego konta (co realizuje cel zmniejszania bariery rozpoczęcia pracy z API).

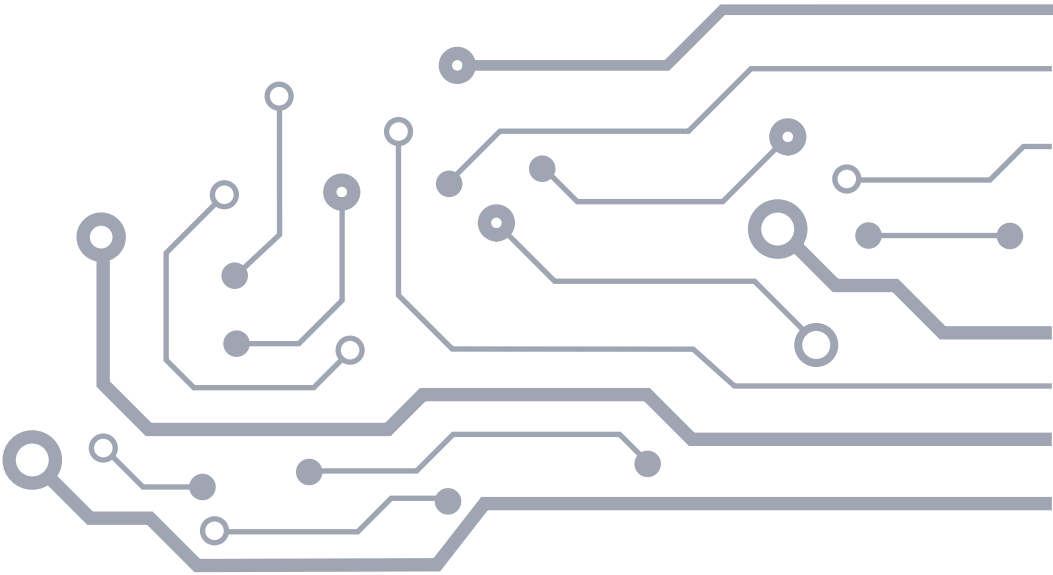
Mimo, że OAuth 2.0 nie jest jeszcze standardem zatwierdzonym przez IETF (status Proposed Standard), jest on rekomendowany jako rozwinięcie OAuth 1.0 pozbawione części jego wad, oraz dostarczający bardziej kompleksowej funkcjonalności niż BasicAuth.

Protokół OAuth nie jest jednak protokołem autentykacyjnym[15], gdyż dokonuje jedynie autoryzacji – nie służy do weryfikacji użytkownika, czy jest tym, za kogo się podaje. Zastosowanie czystego OAuth 2.0 do autentykacji będzie rozwiązaniem błędnym generującym między innymi następujące problemy:

- tokeny będą uznawane w systemie za poświadczenie uwierzytelnienia w systemie, podczas gdy w założeniu OAuth realizuje tylko funkcję delegowania autoryzacji,
- dostęp do chronionego fragmentu staje się dowodem na uwierzytelnienie, a wykorzystanie tokenów na okaziciela (Bearer Tokens) skutkuje możliwością podszycia się pod wybraną tożsamość.

[15] Zob. **https://oauth.net/articles/authentication** [Dostęp 12.02.2018]

Naprzeciw opisanym wyżej problemom wychodzi OpenID Connect 1.0, które jest prostą warstwą nad OAuth 2.0. OpenID Connect umożliwia weryfikację wykonywaną za pomocą serwera autoryzującego, ale dodatkowo również zapewnia pozyskanie profilu klienta w sposób dedykowany stylowi REST i oparty jest o tokeny JSON Web Token (JWT)[16]. OpenID Connect obsługuje wszystkie rodzaje klientów włączając w to klientów opartych o przeglądarkę, mobilnych czy funkcjonujących bezpośrednio z Javascript. Specyfikacja OpenID Connect jest adaptowalna i rozszerzalna oraz umożliwia obsługę sesji, kiedy jest to konieczne.



[16] Zob. Standard JWT dostarczany przez IETF **https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32** [Dostęp 12.02.2018]