Załącznik nr 5.1.3.1 do Wzorów umów (Załączników nr 8.I-8.IV do SIWZ)

Znak sprawy: ZR/1/ZP/22/20

Platforma Integracyjna SPO dla ŚKUP

Używanie SPO WEB.API

Historia zmian

| Numer wersji | Data | Opis zmian | Autor |
| --- | --- | --- | --- |
| 1.01 | 2019-08-29 | Przekazanie w formie załącznika do Opisu API ŚKUP |  |

## **Spis treści**

[Wywołania usług 3](#_Toc17982717)

[Parametry zapytań REST 4](#_Toc17982718)

[Nagłówki HTTP 5](#_Toc17982719)

[Obsługa idempotentności 5](#_Toc17982720)

[Spo-DeviceId 5](#_Toc17982721)

[Spo-RequestId 6](#_Toc17982722)

[Spo-PreviousRequestId 6](#_Toc17982723)

[Obsługa kompresji 6](#_Toc17982724)

[Śledzenie wywołań 6](#_Toc17982725)

[Zabezpieczenia 7](#_Toc17982726)

[Dokumentacja poszczególnych usług API 9](#_Toc17982727)

[Struktura zwracanych komunikatów 12](#_Toc17982728)

[Obsługa błędów 12](#_Toc17982729)

# Wywołania usług

Usługi REST z obsługą operacji CRUD poprzez dedykowane metody http:

* **OPTIONS**: Wykorzystywana do pozyskania informacji o obsługiwanych przez usługę metodach. Wszystkie metody OPTIONS mogą być wołane wielokrotnie, gdyż nie modyfikują żadnych zasobów (są idempotentne).

Wywołania typu OPTIONS nie posiadają kontentu. W informacji zwrotnej kontent nie jest wykorzystywany lub zawiera specyfikację błędów. Informacja zwrotna zawiera natomiast nagłówek wskazujący obsługiwane przez usługę metody – np.:

Allow: HEAD,GET,PUT,DELETE,OPTIONS

* **GET**: Wykorzystywana do pobierania danych. Wszystkie metody GET mogą być wołane wielokrotnie, gdyż nie modyfikują żadnych zasobów (są idempotentne).

Wywołania typu GET nie posiadają kontentu. Parametry filtracji umieszczane są bezpośrednio w wywołaniu usługi (query parameters). Wywołania typu GET dotyczące pojedynczego zasobu (endpoint’y instancji zasobu lub zasobu płaskiego) nie wykorzystują parametrów filtracji. Ewentualne parametry (pozafiltracyjne) umieszczane są w adresie usługi (query parameters).

Informacja zwrotna zawiera w kontencie dane zasobu/kolekcji zasobów lub specyfikację błędów.

* **HEAD**: Wykorzystywana do pobierania informacji o zasobie i sprawdzenia jego dostępność. Wszystkie metody HEAD mogą być wołane wielokrotnie, gdyż nie modyfikują żadnych zasobów (są idempotentne).

Wywołania typu HEAD są analogiczne do wywołań GET.

Informacja zwrotna jest identyczna do zwracanej dla wywołań GET z zastrzeżeniem braku kontentu (danych).

Wywołania typu HEAD obsługiwane są dla **wszystkich** usług API.

* **POST**: Wykorzystywana do utworzenia nowego zasobu lub kolekcji zasobów. Dwukrotne wywołanie metody POST na danym zasobie spowoduje powołanie dwóch obiektów danego typu (wywołania **nie** są idempotentne). W celu utworzenia pojedynczego nowego zasobu wykorzystywana jest metoda POST na URI głównego zasobu usługi (np. /devices), do zapisywania kolekcji – metoda post na podzasobie specjalnym */bulkCreation* (np. /devices/bulkCreation).

Wywołania typu POST dla pojedynczego zasobu zawsze wykorzystują URI bez id i zawierają w kontencie dane zasobu.

Wywołania typu POST dla kolekcji zawsze wykorzystują URI bez id, ale z rozwinięciem */bulkCreation* i zawierają w kontencie dane kolekcji zasobów.

Dla tworzenia pojedynczego zasobu, informacja zwrotna zawiera identyfikator (w postaci URI) utworzonego zasobu w nagłówku Location, a kontent nie jest wykorzystywany lub zawiera specyfikację błędów.

W przypadku tworzenia kolekcji, informacja zwrotna w nagłówku Location zawiera URI jak w wywołaniu */bulkCreation*, a kontent zawiera kolekcję zawierającą dla utworzonych zasobów: identyfikatora biznesowego, id i linku do utworzonego zasobu lub zawiera specyfikację błędów.

* **PUT**: Wykorzystywana do edycji zasobu. Dwukrotne wywołanie metody PUT na tym samym zasobie jest bezpieczne  (są idempotentne).

Wywołania typu PUT zawsze dotyczą pojedynczego zasobu i zawierają w kontencie zmienione dane całego zasobu wskazanego URI (głównego lub podrzędnego).

W informacji zwrotnej kontent nie jest wykorzystywany lub zawiera specyfikację błędów.

Wywołania typu PUT obsługiwane są przez wybrane usługi API.

* **DELETE**: Wykorzystywana do usuwania zasobów (są idempotentne).

Wywołania typu DELETE zawsze dotyczą pojedynczego zasobu i nie posiadają kontentu.

W informacji zwrotnej kontent nie jest wykorzystywany lub zawiera specyfikację błędów.

Wywołania typu DELETE obsługiwane są przez wybrane usługi API.

Dodatkowo obsługa metod HEAD oraz OPTIONS

# Parametry zapytań REST

Przez parametry zapytania REST rozumiane są parametry wywołania REST przekazywane w URI po nazwie endpoint i znaku ‘?’.

1. Poszczególne parametry rozdzielane są znakiem ‘&’
2. Kolejność parametrów nie jest istotna
3. Podanie parametrów dla rodzaju endpoint lub metody nieobsługujących parametrów powoduje zwrócenie 400 Bad Request oraz dedykowany numer błędu w contencie zwrotnym
4. Podanie w wywołaniu nieobsługiwanych nazw parametrów powoduje zwrócenie 400 Bad Request oraz dedykowany numer błędu w kontencie zwrotnym
5. Ze względu na przeznaczenie rozróżnia się następujące rodzaje parametrów:
6. Parametry filtracji (ograniczają wielkość zwracanych kolekcji obiektów do spełniających warunek filtrowania)
7. Parametry filtracji obsługiwane są tylko w metodzie GET dla endpoint zasobu głównego
8. Wynikowy warunek filtrowania powstaje poprzez połączenie kryteriów podanych parametrów filtracji operacją logiczną AND
9. Parametry filtracji mają postać (nawiasy *[]* oznaczają opcjonalność wystąpienia elementu): *<nazwa>=[<operator>]<wartość>*(przykład: *api.system.pl/v1/spo/resources?date=lt=2018-10-10&name=janek*)
10. Nazwa parametru (*<nazwa>*) odpowiada nazwie parametru zdefiniowanej w definicji funkcjonalnej
11. Operator *<operator>* określa sposób sprawdzania wartości dla parametru i przyjmuje wartości (o długości trzech znaków, ostatni =):
12. *eq=* - wartości równe wskazanej (wartość domyślna, gdy brak operatora)
13. *ne= -*wartości różne od wskazanej
14. *gt=*- wartości większe od wskazanej (dla dat – późniejsze)
15. *ge=*- wartości większe lub równe wskazanej
16. *lt=*- wartości mniejsze od wskazanej (dla dat – wcześniejsze)
17. *le=*- wartości mniejsze lub równe od wskazanej
18. *<wartość>* - wskazana do porównania wartość
19. Obsługiwane są tylko parametry filtracji wskazane w definicji funkcjonalnej, a dla typów specjalnych dodatkowo oznaczone w definicji funkcjonalnej do  takiej obsługi. Aktualnie obsługiwane są tylko typy specjalne FROM..TO.
20. Dla parametrów specjalnych typu FROM..TO oznaczenie w definicji funkcjonalnej realizowane jest przez nazewnictwo – parametry filtracji mają nazwy zaczynające się od przedrostka „From” lub „To” (np. FromLastModification lub ToLastModification) oraz stereotypy From i To (w przypadku konieczności biznesowej zastosowania słowa „To” i „From”  w nazwie parametru filtracji, słowa te będą stosowane na innej pozycji, czyli w środku lub na końcu.
21. Możliwość użycia danego operatora dla danego parametru filtracji i wynika z typu danych parametru filtracji (zdefiniowane w arkuszu definicji mapowania typów danych). Zawężenia dostępności operatorów w ramach danego typu danych następuje poprzez typy specjalne (np. typ pary parametrów FROM..TO).
22. Dla parametru filtracji oznaczonego stereotypem From następuje zawężenie do operatorów g*e, gt i eq*
23. Dla parametru filtracji oznaczonego stereotypem To następuje zawężenie do operatorów *le, lt i eq*
24. Brak podania w wywołaniu parametru wskazanego w definicji funkcjonalnej jako obowiązkowy powoduje zwrócenie 400 Bad Request oraz dedykowany numer błędu w kontencie zwrotnym
25. Parametry sortowania (wskazują porządek sortowania zwracanych kolekcji obiektów)
26. Parametry sortowania obsługiwane są tylko w metodzie GET dla endpoint zasobu głównego
27. Wynikowy porządek sortowania wynika z kolejności podania parametrów sortowania
28. Parametry sortowania mają postać (nawiasy *[]* oznaczają opcjonalność wystąpienia elementu, \* - nielimitowaną możliwość powtórzeń): sort=*[<porządek>]<nazwa>[*,*[<porządek>]<nazwa>]\**
29. Nazwa parametru (*<nazwa>*) odpowiada nazwie atrybutu zdefiniowanej w definicji funkcjonalnej
30. Element *<porządek>* określa kierunek sortowania i przyjmuje wartości:
31. *+* (operator domyślny) – rosnący porządek sortowania
32. *-* – malejący porządek sortowania
33. Obsługiwane są tylko nazwy atrybutów wskazane w definicji funkcjonalnej
34. Parametry stronicowania (wskazują porcję zwracanych kolekcji obiektów)
35. Parametry stronicowania obsługiwane są tylko w metodzie GET dla endpoint zasobu głównego
36. Kolejności podania parametrów stronicowania nie jest istotna
37. Parametry stronicowania mają postać: *<nazwa>*=*<wartość>*
38. Obsługiwane nazwy parametrów stronicowania:
39. *pageNumber* - określa numer strony (porcji) zwracanych danych i jest numerowany od 1.
40. *pageSize -* określa liczbę elementów na stronie, nie może być większy niż maksymalny limit obsługiwany przez zasób.
41. Brak podania parametru pageSize powoduje stronicowanie z maksymalnym limitem obsługiwanym przez zasób

# Nagłówki HTTP

Wszystkie wywołania i odpowiedzi powinny być przekazane w kodowaniu **UTF-8**.

Wszystkie dane, wysyłane i odbierane z serwera (kontent) są w formacie JSON (*application/json*) i kodowaniu UTF-8 (*charset=utf-8)*.

## Obsługa idempotentności

SPO API wspiera obsługę idempotentności metod http POST. O ile nie jest ona wyłączona wymagane jest podawanie dodatkowych nagłówków.

### Spo-DeviceId

Nagłówek ten jednoznacznie identyfikuje urządzenie w Systemie Poboru Opłat. Składa się z dwóch elementów zakodowanych algorytmem base64. Konstrukcja jest następująca:

X’~’Y, gdzie:

X - Typ urządzenie (DeviceType) w formacie tekstowym [String(8)]

~ – Separator informujący, że następną daną będzie numer seryjny

Y – numer seryjny urządzenia w formacie liczbowym unikalny w systemie SPO dla danego typu

Dane w opisanym wyżej formacie powinny być zakodowane base64 i następnie przekazane: np.:

Dla urządzenia typu 123 o numerze seryjnym 123 powinniśmy utworzyć nagłówek :

123~123 – (base64) - > MTIzfjEyMyA=

Czyli przy zapytaniu należy przekazać

**Spo-DeviceId: MTIzfjEyMyA=**

### Spo-RequestId

Nagłówek ten jednoznacznie identyfikuje zapytanie do systemu SPO w kontekście konkretnego urządzenia. Składa się on z trzech elementów zakodowanych algorytmem base64. Konstrukcja jest następująca:

ABYYYYMMDD, gdzie:

A – znacznik czy jest to pierwszy request danego dnia A{0;1} 0=false, 1=true

B – identyfikator wywołania w formie liczbowej INT B

YYYYMMDD - Data przesłania żądania, gdzie YYYY – rok, MM – miesiąc, DD- dzień

Dane w opisanym wyżej formacie powinny być zakodowane base64 i następnie przekazane: np.:

Dla pierwszego żądania z dnia 11 stycznia 2019 powinniśmy utworzyć nagłówek:

1120190111 – (base64) - > MTEyMDE5MDExMQ==

Czyli przy zapytaniu należy przekazać:

**Spo-RequestId: MTEyMDE5MDExMQ==**

### Spo-PreviousRequestId

Nagłówek ten wskazuje żądanie, które było wysłane jako poprzednie. W przypadku gdy jest to pierwsze żądanie atrybut ten powinien mieć wartość identyczną jak Spo-RequestId.

## Obsługa kompresji

API wspiera obsługę kodowania gzip. W szczególnych przypadkach może się zdarzyć, że odpowiedzi będą kodowane, w takich sytacjach zostanie zwrócony nagłówek **content-encoding: gzip** w przypadku podawania danych zakodowanych należy zastosować ten sam nagłówek.

## Śledzenie wywołań

Każda odpowiedź HTTP zawiera m.in. nagłówki *Trace-Id*i *Span-Id*, których wartości jednoznacznie identyfikują zapytanie HTTP klienta. W przypadku zgłaszania problemu z zapytaniem HTTP należy przekazać te wartości w zgłoszeniu. Nagłówek *Span-Id* związany jest z danym zapytaniem HTTP, natomiast *Trace-Id* z identyfikacją procesu obejmującego zestaw wywołań usług API. Jeżeli *Trace-Id* występuje w wywołaniu http, w informacji zwrotnej wywołania przekazywana jest ta sama wartość dla tego nagłówka. Spinanie zestawu usług w jeden proces leży po stronie klienta usług – w kolejnych wywołaniach HTTP powinna być przekazywana wartość nagłówka *Trace-Id*uzyskana z informacji zwrotnej pierwszego wywołania HTTP w ramach procesu.

Przykładowa odpowiedź wywołania usługi API (dla pierwszego wywołania w procesie):

HTTP/1.1 404 Not Found

Connection: keep-alive

**Trace-Id: 2485ec27856c56f4**

**Span-Id: 2485ec27856c56f4**

Content-Type: application/json; charset=utf-8

Content-Length: 94

# Zabezpieczenia

Usługi wspierają autoryzację i autentykację użytkowników metodą OAuth2. Podczas wywołania usługi niezbędne jest podania tokena autoryzacyjnego.

* 1. Metody uzyskania Tokena
* **Client credentials**

Jest to metoda autoryzacja pozwalająca zautentykować jedynie urządzenie/aplikację. Aby wykorzystać tą metodę logowania należy posiadać:

* + client\_id
  + client\_secret

Uzyskanie tokenu poprzez wywołanie POST:

<authorization\_server\_url>/oauth/token?grant\_type=client\_credentials

Podając dane logowania klienta poprzez Basic Auth

* **Resource owner**

Metoda autoryzacji pozwalająca na sprawdzenie zarówno użytkowniak jak i klienta/aplikacji, z której się łączy. Aby wykorzystać tą metodę logowania należy posiadać:

* + client\_id
  + client\_secret
  + username
  + password
  + organization

Uzyskanie tokenu poprzez wywołanie POST:

<authorization\_server\_url>/oauth/token?grant\_type=password&username=<username>&password=<password>&org=<company>

Podając dane logowania klienta poprzez Basic Auth. Dodatkowe dane (username, password, org) możemy również podać w formie formularza

* **Refresh token**

Metoda ta dostępna jest tylko dla wybranych klientów/aplikacji/urządzeń. Aby wykorzystać tą metodę logowania należy posiadać:

* + Client\_id
  + Client\_secret
  + Wcześniej uzyskany refresh token

Uzyskanie tokenu poprzez wywołanie POST:

<authorization\_server\_url>/oauth/token?grant\_type=refresh\_token&refresh\_token=<refresh\_token>

Podając dane logowania klienta poprzez Basic Auth.

* **Authorization code**

Metoda wykorzystywania głównie przy integracji systemów webowych lub aplikacji mobilnych. Aby wykorzystać tą metodę logowania należy posiadać:

* + client\_id
  + client\_secret
  + username
  + password
  + organization
  + redirect\_uri – strona, na którą ma nastąpić przekierowanie po poprawnym logowaniu
  + state – opcjonalny atrybut, unikalny identyfikator generowany przez klienta. Zalecany ze względu na CSRF

Uzyskanie autoryzacji w tej metodzie jest dwuetapowe:

* + - 1. żądanie autoryzacji poprzez klienta/aplikację

<authorization\_server\_url>/oauth/authorize?response\_type=code&client\_id=<clientid>&redirect\_uri=<redirecturi>&state=V2VkIE5vdiAwNyAyMDE4IDEwOjQ1OjAxIEdNVCswMTAwIChjemFzIMWbcm9ka293b2V1cm9wZWpza2kgc3RhbmRhcmRvd3kp

Po wywołaniu następuje przekierowanie na stronę logowania

* + - 1. logowanie użytkownika

Po wprowadzeniu danych logowania i poprawnym ich zweryfikowaniu użytkownik zostaje przekierowany na stronę podaną w redirect\_uri

* 1. Opis zwracanych danych

*{*

*"access\_token": "eyJhbGciOiJIUzM4NCIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NDY5NDU3MTAsInVzZXJfbmFtZSI6InNhbXBsZV91c2VyIiwiYXV0aG9yaXRpZXMiOlsiMV83ZjVhIiwiMF83ZTJlIl0sImp0aSI6ImE4MzEwZDY3LWY0Y2UtNDU1Ni05Yjg3LWVhYzNmMWY1NDFlNyIsImNsaWVudF9pZCI6InNhbXBsZV9jbGllbnQiLCJzY29wZSI6WyI0XzgyZTAiLCI0XzgyZTEiXX0.mdenaICB0G7zueDw664trFTEnFFGqUmxBENpcICgm6-z24iUZB6dmMiQoHsAkd9V",*

*"token\_type": "bearer",*

*"refresh\_token": "eyJhbGciOiJIUzM4NCIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1NDY5NDU3MTAsInVzZXJfbmFtZSI6InNhbXBsZV91c2VyIiwiYXV0aG9yaXRpZXMiOlsiMV83ZjVhIiwiMF83ZTJlIl0sImp0aSI6ImE4MzEwZDY3LWY0Y2UtNDU1Ni05Yjg3LWVhYzNmMWY1NDFlNyIsImNsaWVudF9pZCI6InNhbXBsZV9jbGllbnQiLCJzY29wZSI6WyI0XzgyZTAiLCI0XzgyZTEiXX0.mdenaICB0G7zueDw664trFTEnFFGqUmxBENpcICgm6-z24iUZB6dmMiQoHsAkd9V",*

*"expires\_in": 120,*

*"scope": "4\_82e0 4\_82e1",*

*"jti": "a8310d67-f4ce-4556-9b87-eac3f1f541e7"*

*}*

* Access token – token autoryzacyjny
* Token type – typ wydanego tokenu
* Refresh token – token do odświeżenia tokenu w przypadku wygaśnięcia ważności (UWAGA: Refrsh token jest zwracany tylko dla klientów z uprawnieniami do jego wykorzystania)
* Expires in – czas ważności tokenu
* Scope – zakres dostępu funkcji dla klienta w danym tokenie
* Jti – identyfikator kontrolny tokena
  1. Przekazanie Tokena do usług

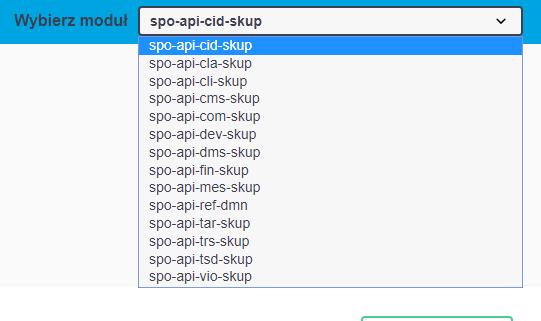
Aby zautoryzować się w usłudze należy przekazać, uzyskany jedną z wcześniejszych metod ,token poprzez nagłówek *Authorization*, np.:

Authorization: Bearer <uzyskany\_token>

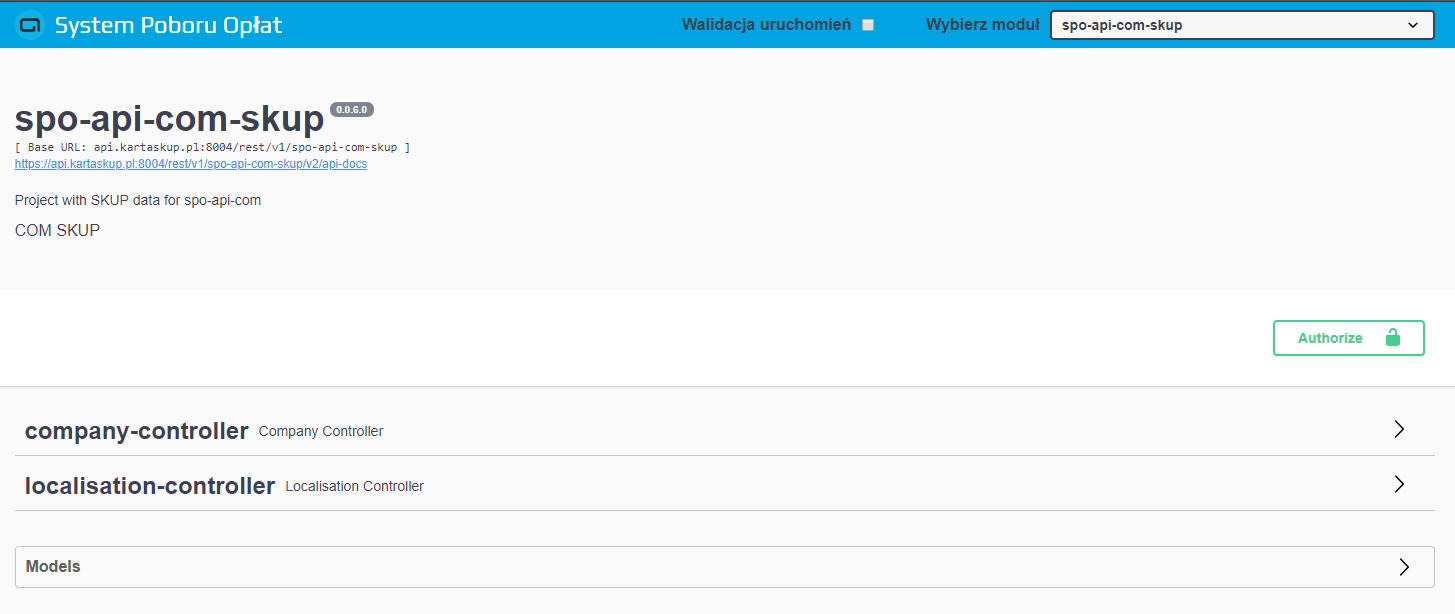
# Dokumentacja poszczególnych usług API

Opis poszczególnych zasobów oraz metod w api dostępy jest z poziomu scentralizowanego serwera dostępnego pod adresem <api\_root\_url>/swagger-ui.html

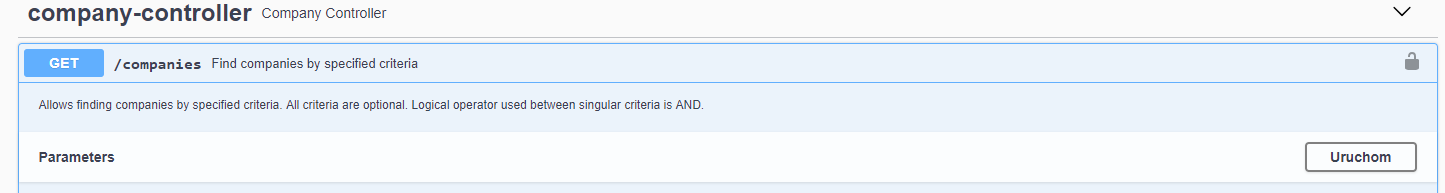
W celu zapewnienia porządku poszczególne funkcje pogrupowane są w moduły. Opis poszczególnych zasobów w modułach jest dostępny poprzez wybór modułu:



Po wybraniu modułu wyświetlana jest lista dostępnych zasobów oraz model danych wymieniany w ramach tych wywołań.

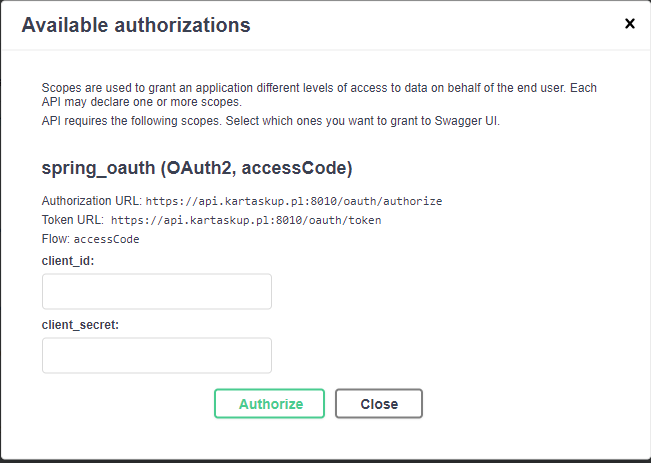


Możliwe jest sprawdzenie działania usług poprzez uruchomienie (przycisk uruchom po rozwiązaniu danego kontrolera).

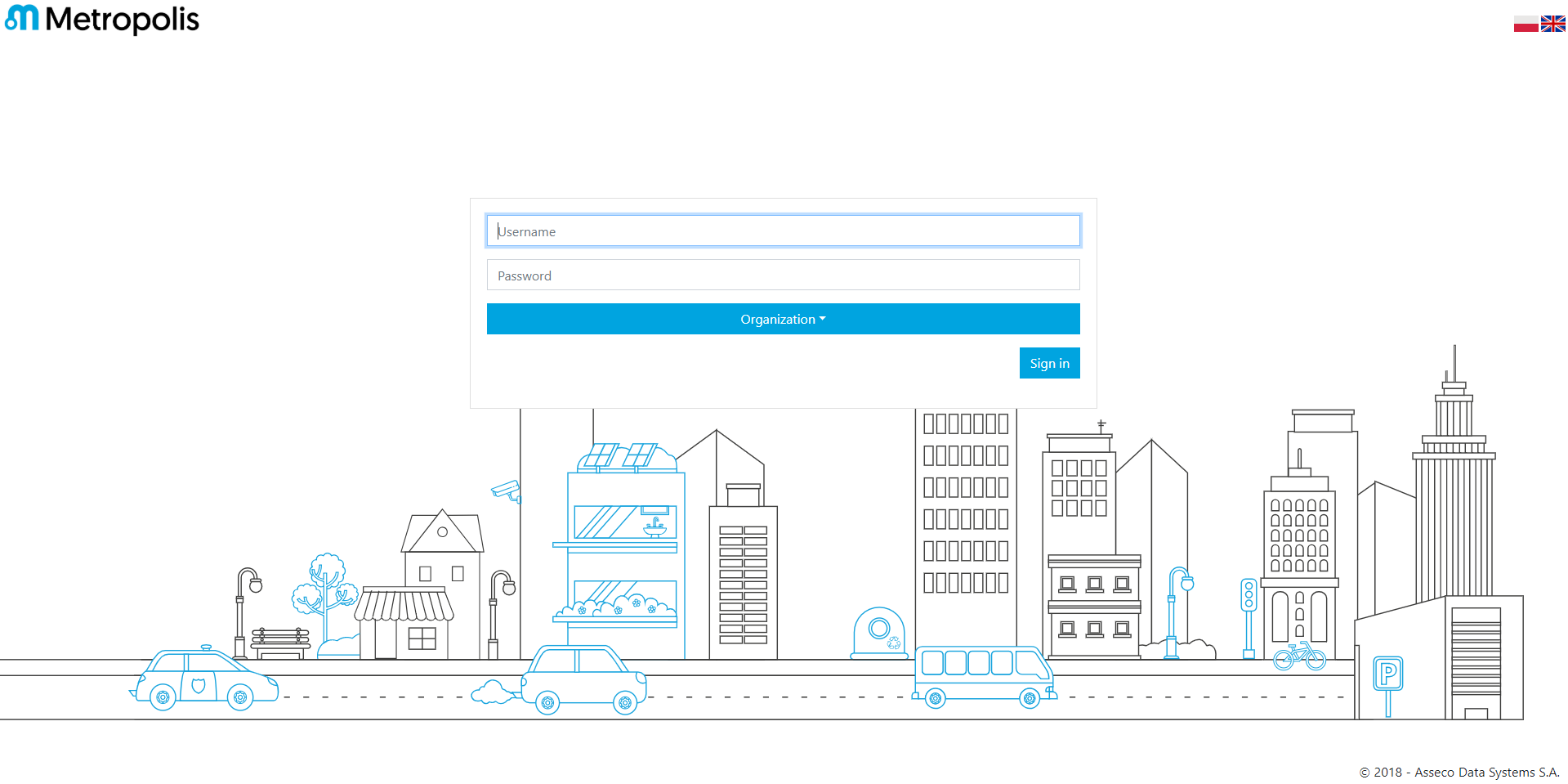


Aby wykonać poprawne wywołanie niezbędne jest podanie wszystkich wymaganych atrybutów oraz zautoryzowanie poprzez przejście procesu „Authorize” . Opcja ta pozwala na testowanie z różnymi kontekstami uprawnień.

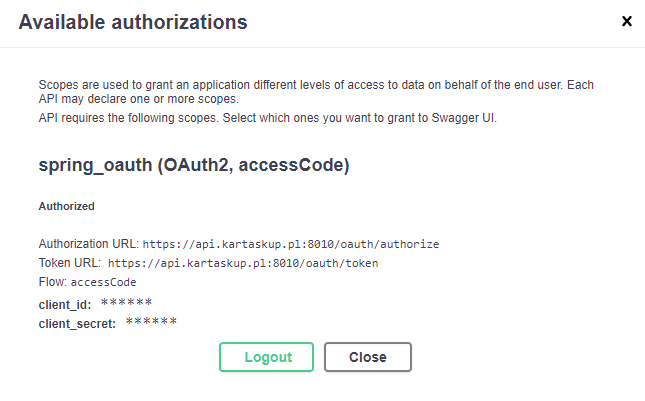
Po kliknięciu pojawi nam się okno do wprowadzenia danych klienta/aplikacji.



Następnie zostaniemy przekierowaniu na stronę logowania



Po zalogowaniu zostaniemy ponownie przekierowaniu do strony serwera dokumentacyjnego i dostaniemy możliwość wylogowania.



# Struktura zwracanych komunikatów

Struktura zwracanych obiektów jest możliwa do wglądu w ramach dokumentacji opisanej w pkt 4.

* API zwraca dane w formacie apllication/json
* API wspiera hipermedia w standardzie HAL
* API zwraca błędy w ustrukturyzowanym formacie

{

   "status": "fail",

   "errors":[{ "code": "1FA064",

"message": "Database connection error",

"details": null

"field”: null }]

}

* Status: przyjmuje dwie wartości: error w przypadku błędów klienta oraz fail w przypadku błędów serwera
* Errors: zawiera listę błędów z atrybutami: code – kod błędu, message –ogólna wiadomość, details – wiadomość szczegółowa (opcjonalnie) field – pole, którego dotyczy błąd (opcjonalnie)

# Obsługa błędów

W razie wystąpienia błędów wykorzystywany jest ogólny mechanizm zwracania błędów zdefiniowany dla protokołu HTTP oraz specyfikacja błędów w kontencie odpowiedzi.

2xx (brak błędu 🡪 success)

* 200 OK: Zwracany w przypadku powodzenia (dotyczy metod GET, HEAD, OPTIONS).
* 201 Created: Potwierdza utworzenie zasobu lub wszystkich elementów kolekcji zasobów (dotyczy metody POST).
* 202 Accepted: Potwierdza przyjęcie do dalszego przetwarzania. Wykorzystywane w asynchronicznych metodach API
* 204 No Content: Zwracany przy powodzeniu operacji, kiedy żądanie nie zwraca żadnych danych (dotyczy metod PUT, DELETE).

4xx (błędy klienta 🡪 error)

* 400 Bad Request: Wysłano niepoprawne parametry lub dane JSON (dotyczy metod GET, PUT, POST, HEAD).
* 401 Unauthorized: Negatywna weryfikacja przez usługę uwierzytelnienia lub brak autoryzacji dla usług API (dotyczy metod GET, PUT, POST, DELETE, HEAD, OPTIONS).
* 403 Forbidden: Wywołanie jest uwierzytelnione, ale nie ma praw do danego zasobu lub danej operacji HTTP na zasobie (dotyczy metod GET, PUT, POST, DELETE, HEAD, OPTIONS).
* 404 Not Found: Odpytywany zasób nie istnieje w API lub nie istnieje zasób spełniający kryteria filtracji (dotyczy metod GET, POST, DELETE, HEAD, OPTIONS).
* 405 Method Not Allowed: Usługa nie obsługuje danej operacji HTTP (dotyczy metod PUT, POST, DELETE).
* 406 Not Acceptable: W nagłówku Accept lub Content-Type przekazany został nieobsługiwany przez zasób typ danych (dotyczy metod GET, PUT, POST, DELETE, HEAD).
* 411 Length required: brak nagłówka Content-Length w zapytaniu (dotyczy metod PUT, POST).
* 422 Unprocessable Entity: Wysłano niepoprawne wartości pól, np. walidacja obiektu zwróciła błąd, albo któreś z pól nie spełnia kryteriów nałożonych na nie przez zasób (dotyczy metod GET, PUT, POST).

5xx (błędy serwera 🡪 fail)

* 500 Internal Server Error: Wywołanie jest poprawne ale nie jest możliwa jego obsługa po stronie serwera

Powyższe kody błędów HTTP uzupełniane są informacją szczegółową przekazywaną w body response (code, message). Kody błędów szczegółowych zdefiniowane są w Rejestrze kodów błędów.