Dokumentacja API **sender sms**

Wersja	Autor	Zakres zmian	Data utworzenia
1.0	Telein Sp. z o.o.	Wersja bazowa	30.07.2018
1.1	**Telein Sp. z o.o**	**Aktualizacja metod**	**19.11.2018**

Spis treści

Wstęp

Niniejsza dokumentacja prezentuje sposób wykorzystania API usługi **sender sms**.

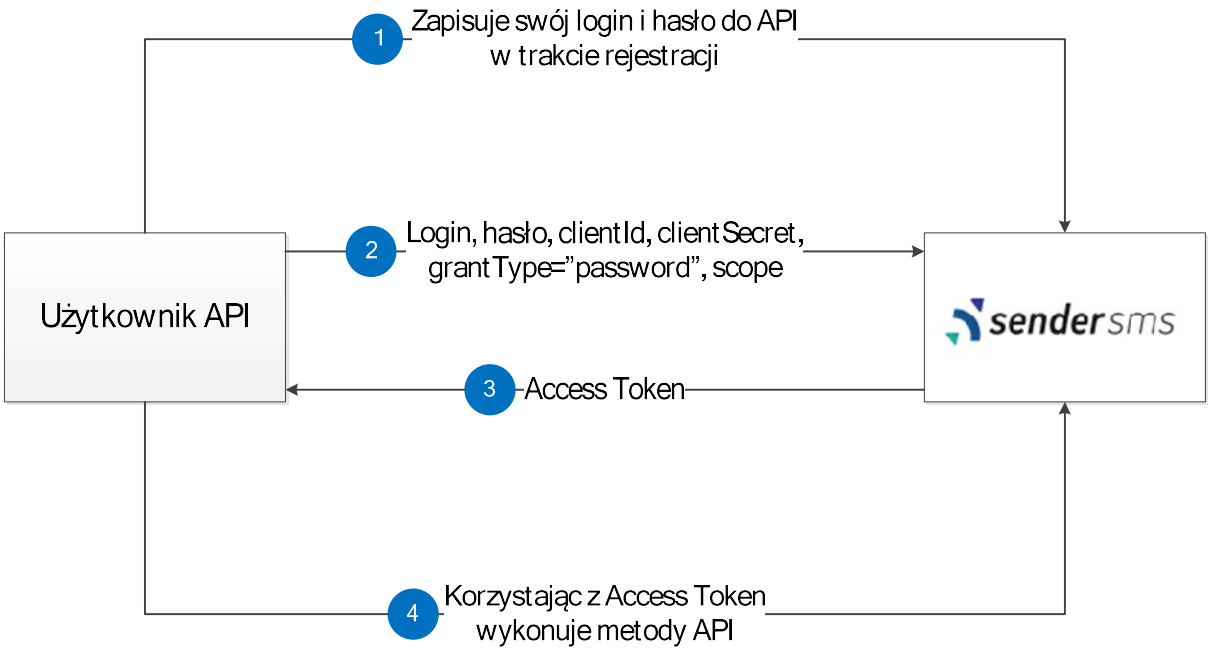
Poniżej zaprezentowano sposoby autoryzacji, przykłady podłączenia API, sposób dostępu do dokumentacji oraz statusy wysyłek i wiadomości SMS, MMS, VMS.

Autoryzacja

Informacje ogólne

Mechanizm autoryzacji API usługi **sender sms** został oparty o standard OAuth2 dla Grant Type = password

Schemat koncepcji autoryzacji OAuth2 oraz wywoływania metod API:



Co jest potrzebne

Aby połączyć się do API, powinny być spełnione następujące warunki:

1. użytkownik ma założone konto w systemie **sender sms**
2. użytkownik ma nadane uprawnienie zezwalające na dostęp do panelu administracyjnego i do API
3. użytkownik zna dane logowania (login, hasło) do panelu oraz API

4. użytkownik zna wartości parametrów clientId, clientSecret wykorzystywanych przez usługę **sender sms**

Jak spełnić w/w warunki? To proste:

1. Załóż konto w usłudze **sender sms** wchodząc na stronę https://sendersms.pl/, wybierając link „Rejestracja” i podążając za instrukcjami zakładania konta
2. Uprawnienie zezwalające na dostęp do panelu administracyjnego jest nadawane automatycznie po pomyślnym zakończeniu procesu rejestracji (pkt. 1). Aby otrzymać uprawnienie umożliwiające dostęp do API należy wystąpić o stosowne uprawnienia do administratora systemu.
3. Dane logowania do panelu administracyjnego (tworzone przez użytkownika w trakcie rejestracji) oraz do API są identyczne
4. Na potrzeby wykorzystania mechanizmu autoryzacji OAuth2 ustala się wartości parametrów:
    - AccessTokenUri = https://sendersms.pl:8082/oauth/token
    - clientId: sender_apiapp
    - clientSecret: gpa4RXc4wfbpSxLU

Przykład podłączenia API w technologii JAVA Spring

Konfiguracja:

```
package pl.sender.portal.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
import org.springframework.security.oauth2.client.DefaultOAuth2ClientContext;
import org.springframework.security.oauth2.client.OAuth2RestTemplate;
import org.springframework.security.oauth2.client.token.grant.password.Resource
OwnerPasswordResourceDetails;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

/**
 * @author sender sms
```

```
 */
@Configuration
public class SmsSenderApiConfiguration {
    @Value("http://${sender.senderapi.host}:${sender.senderapi.port}")
    private String address;
    @Value("${sender.senderapi.user}")
    private String user;
    @Value("${sender.senderapi.password}")
    private String password;
    @Value("${sender.senderapi.clientId}")
    private String clientId;
    @Value("${sender.senderapi.clientSecret}")
    private String clientSecret;

    @Bean
    ResourceOwnerPasswordResourceDetails resourceOwnerPasswordResourceDetails(){
        ResourceOwnerPasswordResourceDetails resourceDetails = new
           ResourceOwnerPasswordResourceDetails();
        List<String> scope = new ArrayList<>(2);
        scope.add("read");
        scope.add("write");
        resourceDetails.setUsername(user);
        resourceDetails.setPassword(password);
        resourceDetails.setAccessTokenUri(address + "/oauth/token");
        resourceDetails.setClientId(clientId);
        resourceDetails.setClientSecret(clientSecret);
        resourceDetails.setGrantType("password");
        resourceDetails.setScope(scope);
        return  resourceDetails;
    }

    @Bean
    DefaultOAuth2ClientContext defaultOAuth2ClientContext(){
        return new DefaultOAuth2ClientContext();
    }

    @Bean
    OAuth2RestTemplate smsSenderApiRestTemplate() {
        OAuth2RestTemplate restTemplate =  new
            OAuth2RestTemplate(resourceOwnerPasswordResourceDetails(),
           defaultOAuth2ClientContext());
        List<HttpMessageConverter<?>> messageConverters = new
            LinkedList<HttpMessageConverter<?>>();
        messageConverters.add(new MappingJackson2HttpMessageConverter());
        restTemplate.setMessageConverters(messageConverters);
        return restTemplate;
    }
}
```

Prosta klasa wiadomości SMS:

```
package pl.sender.portal.service.sms;

/**
 * @author sender sms
 */
public class SimpleSmsRequest {
    private String recipientNumber;
    private String messageText;

    public String getRecipientNumber() {
        return recipientNumber;
    }

    public void setRecipientNumber(String recipientNumber) {
        this.recipientNumber = recipientNumber;
```

```
    }

    public String getMessageText() {
        return messageText;
    }

    public void setMessageText(String messageText) {
        this.messageText = messageText;
    }

    public SimpleSmsRequest(String recipientNumber, String messageText) {
        this.recipientNumber = recipientNumber;
        this.messageText = messageText;
    }
}
```

Użycie:

```
package pl.sender.portal.service;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.RequestEntity;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.context.SecurityContext;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.context.SecurityContextImpl;
import org.springframework.security.oauth2.client.OAuth2RestTemplate;
import org.springframework.stereotype.Service;
import pl.sender.portal.domain.SenderUser;
import pl.sender.portal.dto.SmsCenterDeliveryReceiptDTO;
import pl.sender.portal.service.sms.DifferentSmsToMany;
import pl.sender.portal.service.sms.SimpleSmsRequest;

import javax.inject.Inject;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.LinkedList;
import java.util.List;

/**
 * Sending sms service.
 * @author sender sms
 */
@Service
public class SmsService {
    private final Logger log = LoggerFactory.getLogger(SmsService.class);

    @Value("${sender.senderapi.host}:${sender.senderapi.port}")
    private String address;

    @Inject
    OAuth2RestTemplate smsSenderApiRestTemplate;

    public void sendMySms(){
        String message = "Mój sms testowy";
        String number = "48999888777";
        sendSms(message);
    }

    public void sendSms(String number, String content){
        DifferentSmsToMany smsToMany = new DifferentSmsToMany();
```

```
        smsToMany.setEscapeUnicodeCharacters(false);
        smsToMany.setFast(true);
        List<SimpleSmsRequest> smses = new LinkedList<SimpleSmsRequest>();
        smses.add(new SimpleSmsRequest(number, content));
        smsToMany.setSmses(smses);
        URI uri = null;
        try {
            uri = new URI(new StringBuilder()
                .append("http://")
                .append(address)
                .append("/sender_api/sendDifferentSmsToMany").toString());
        } catch (URISyntaxException e) {
            log.error("Nieprawidłowy adres serwera SenderSMS api", e);
        }
        RequestEntity request = RequestEntity
            .post(uri)
            .accept(MediaType.APPLICATION_JSON)
            .body(smsToMany);
        SecurityContextHolder.setContext(new SecurityContextImpl());
        ResponseEntity<?> response = smsSenderApiRestTemplate.exchange(request,
           Void.class);
        if(response.getStatusCode() != HttpStatus.OK)
            throw new RuntimeException("Błąd podczas wysyłania smsa");
    }
}
```

Wysyłki i wiadomości

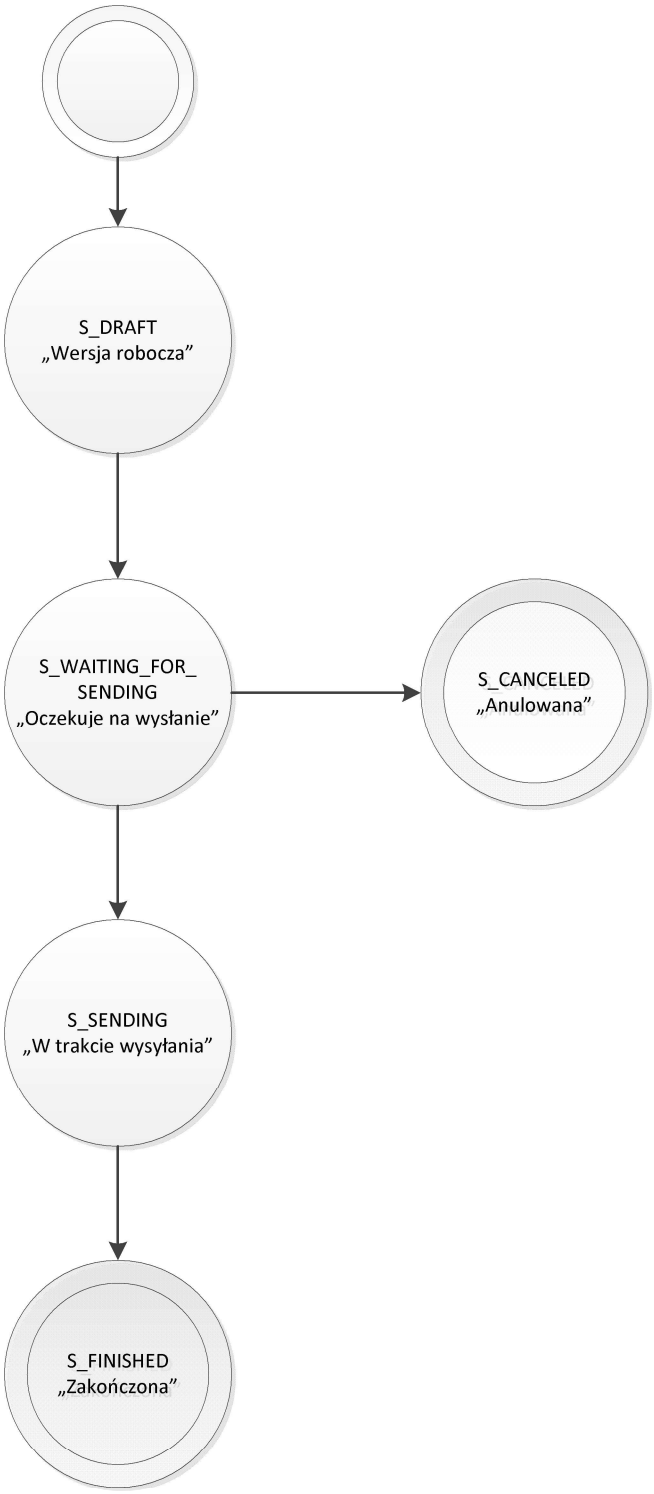
Definicje

Usługa **sender sms** opiera się na koncepcji wysyłki i wiadomości.

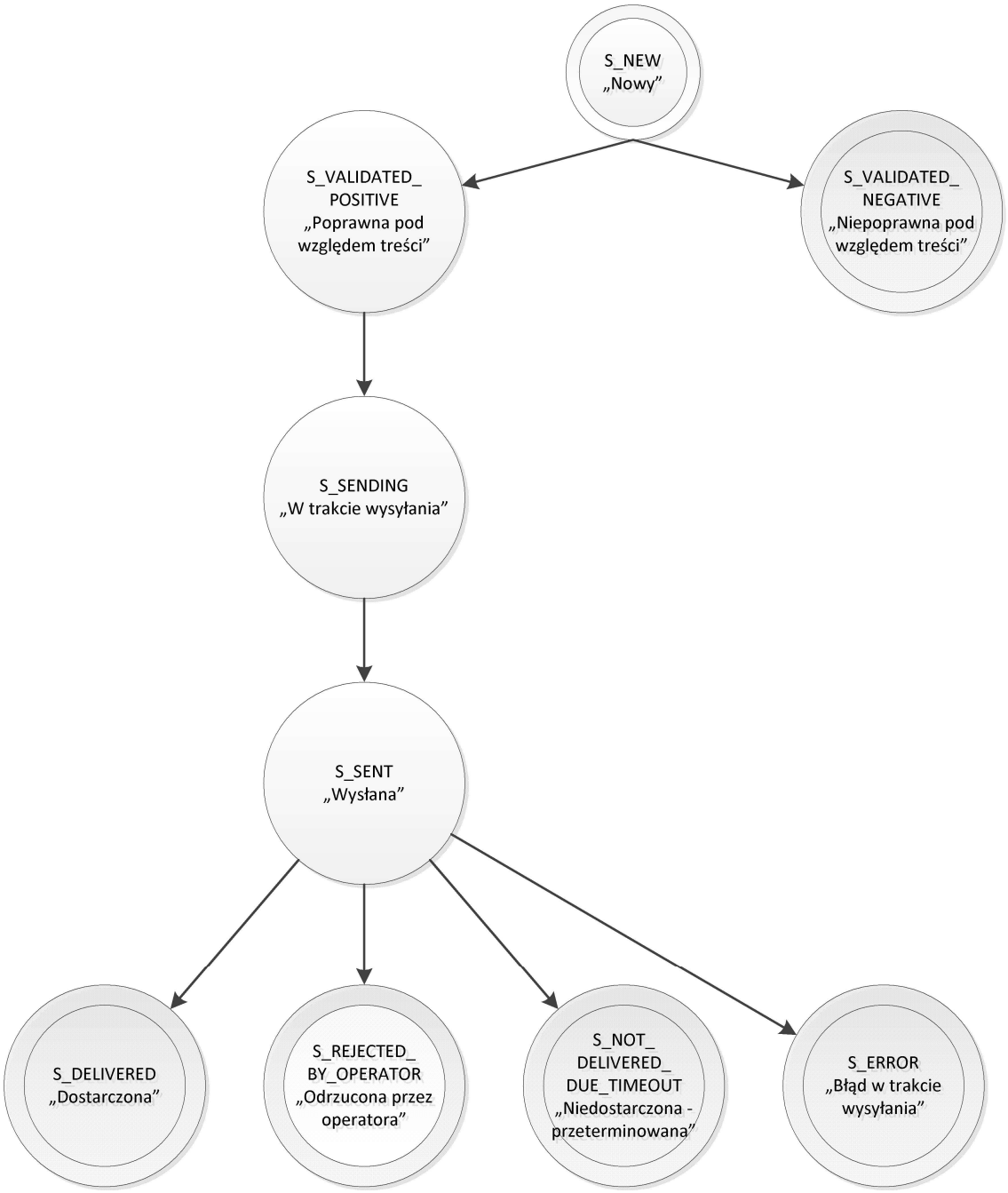
Wysyłka – zbiór wiadomości, o wspólnych właściwościach. Dodana do systemu w ramach wykonania jednej metody API. Może składać się z jednego lub wielu SMS. Posiada własny status.

Wiadomość – pojedynczy SMS. Posiada własny status.

Statusy wysyłek



Statusy wiadomości

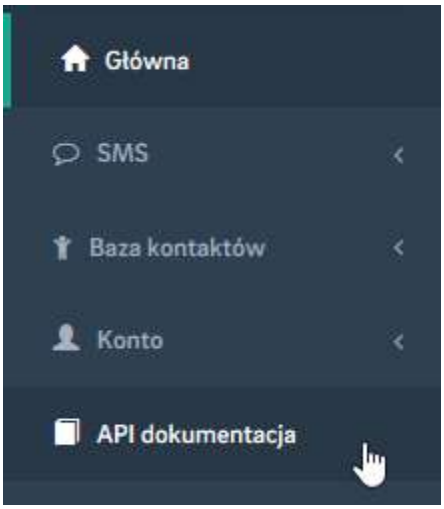


API

Dokumentacja i testowe wywołania

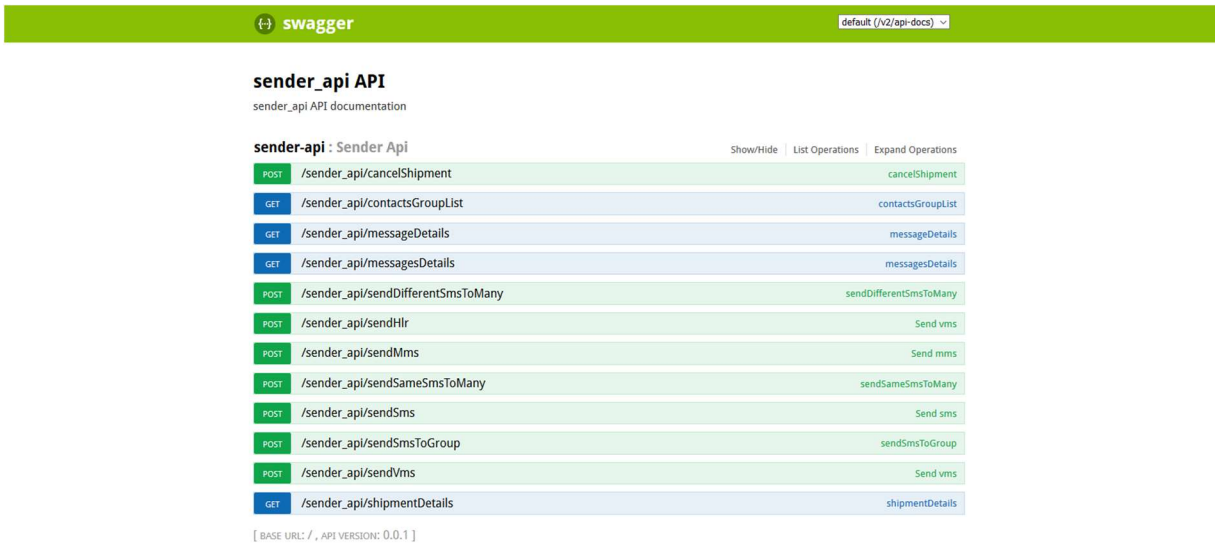
Po zalogowaniu się do panelu administracyjnego **sender sms**, użytkownik ma możliwość zapoznania się z metodami API i wypróbowania każdej z nich przy pomocy środowiska pomocniczego Swagger.

W tym celu należy z menu wybrać pozycję „API dokumentacja”:



po czym w celu weryfikacji uprawnień ponownie wpisać dane logowania do panelu administracyjnego.

Po poprawnym zalogowaniu prezentowane są wszystkie dostępne metody API.



Metody

- cancelShipment – anulowanie wysyłki
- contactsGroupList – pobranie listy grup kontaktów (baza kontaktów)
- messageDetails – pobranie szczegółów wiadomości SMS
- sendDifferentSmsToMany – wysłanie wysyłki składającej się z różnych wiadomości dla różnych numerów
- sendSameSmsToMany – wysłanie wysyłki zawierającej wielu odbiorców wiadomości i jedną treść, która zostanie do nich wysłana
- sendSms – wysłanie pojedynczego SMS
- sendSmsToGroup – wysłanie SMS do grupy
- shipmentDetails – pobranie szczegółów wysyłki
- templateList – pobranie listy szablonów wiadomości
- sendHlr – sprawdzenie numeru w bazie HLR
- sendMms – wysłanie wiadomości MMS
- sendVms – wysłanie wiadomości VMS

Webhook

Po każdej zmianie statusu wiadomości na adres podany przy tworzeniu wysyłki (parametr webhookAddress) za pomocą metody POST wysyłana jest struktura SentMessageInfo o polach:

```
public class SentMessageInfo {
    private UUID id;
    private String recipientNumber;
    private String currentStateName;
    private UUID shipmentId;
}
```

Kontakt techniczny

W razie jakichkolwiek problemów technicznych prosimy o kontakt e-mail:

admin@sendersms.pl