LANTMÄTERIET

DOKUMENTVERSION: 1.0 2020-07-03

API-portalen guide

Innehållsförteckning

Inledning

Lantmäteriet använder sig idag av WSO2 API-Manager för att exponera några av sina API:er, på sikt är detta huvudsakliga plattformen för exponering av Lantmäteriets API:er.

Genom API-Portalen så kan användare själva lista och prenumerera på API:er av intresse, de kan ta del av dokumentation, testa ett API för ökad förståelse, ta del av statistik gällande deras användning av API:er mm.

För att anropa ett API så används en *Access Token* (även kallad API-nyckel) vilket är den enda autentiseringsmetoden.

Användarkontot och dess behörighet styr även vilka API:er som användaren kan se och prenumerera på i API-portalen.

Dokumentation för plattformen:

- WSO2 API-Manager KeyConcepts
- WSO2 API-Manager Wiki

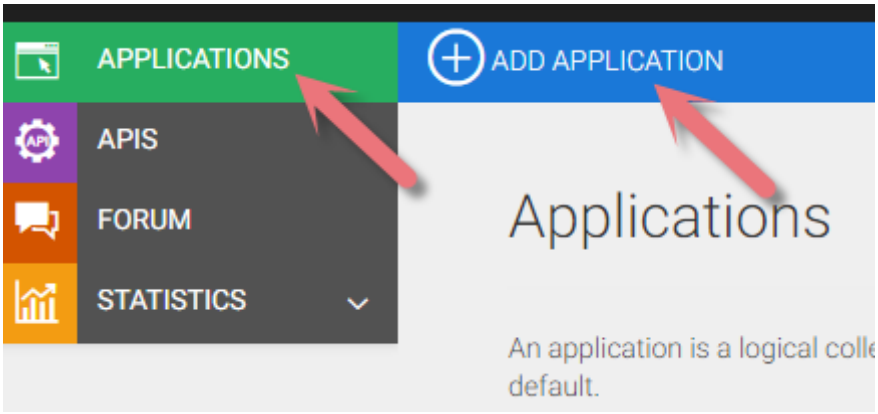
API-portalen finns tillgänglig i våra två miljöer:

- Verifieringsmiljön (VER)
- Produktionsmiljön (PRD)

Begrepp

Applikationer

Figur 1. Skärmbild över hur man lägger till en ”Application”.



Under "Applications" visas dina applikationer. Applikationer tillåter dig att gruppera en samling API:er och använda samma *Access Token* för alla API:er i den grupperingen.

Dina nycklar och dina prenumererade API:er är grupperade per applikation. Det finns alltid en *Default Application* skapad som du kan använda men det är lämpligt är om du skapar och namnger en egen ny applikation.

Observera: Om du som användare har applikationerna APP-X, APP-Y och APP-Z som använder Api:er(ett och samma api kan ingå i flera applikationer) så ska du skapa motsvarande logiska applikationer i API-Portalen dvs då hanteras nycklar och prenumerationer separat för resp. applikation X, Y, Z.

Nycklar

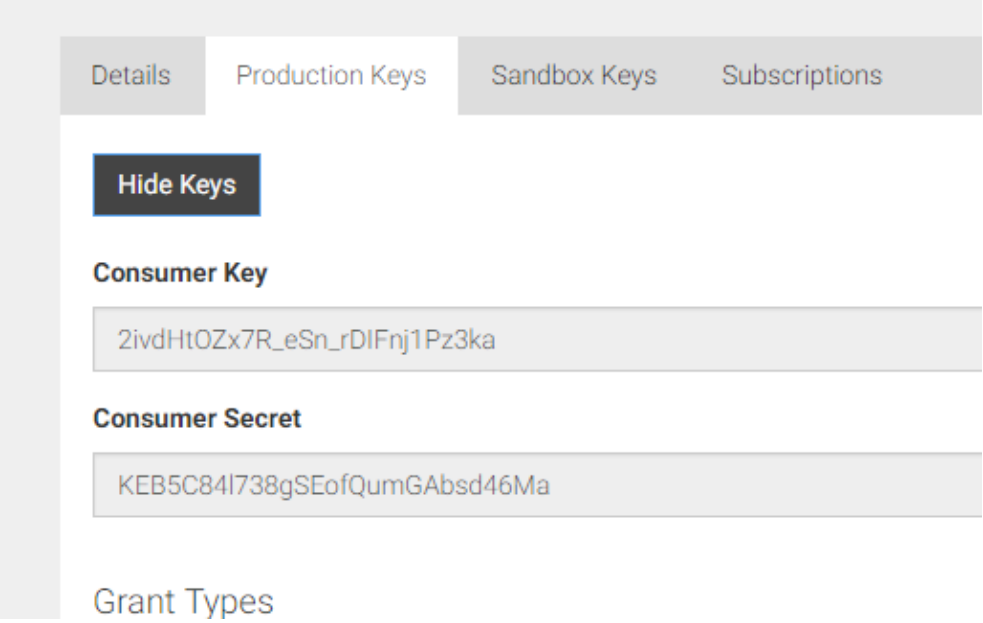
För varje logisk applikation (se ovan) genereras ett nyckelpar, *Consumer Key* och *Consumer Secret*. Dessa används i sin tur för att generera en *Access Token* för applikationen.

Dessa tre nycklar kan ses i API-Portalen, normalt används endast "Production Keys" men det går även att generera "Sandbox Keys" vilka används då ett API är kopplat mot två backend-tjänster och där "Sandbox" t.ex. pekar mot en test-miljö.

Dvs det går då att styra mot vilken miljö man går mot baserat på vilken *Access Token* som används, under förutsättning att ett API har en "Sandbox" miljö.

CONSUMER KEY OCH CONSUMER SECRET

Figur 1. Skärmbild över "Production Keys"-fälten.



Dessa nycklar är unika per applikation och används för att generera ny *Access Token*. Läs mer nedan under rubrik Exempel – Access Token generering.

Curl-exempel på generering av *Access Token* m.h.a. *Consumer key* och *Consumer secret*.

```
curl -k -d "grant_type=client_credentials" -H "Authorization: Basic
<Base64(consumer-key:consumer-secret)>" https://api-ver.lmv.lm.se/token
```

```
{"access_token":"5f996af4-afe6-3438-8e1e-7823018f6898","scope":"am_ap-
plication_scope default","token_type":"Bearer","expires_in":1373}
```

Curl-exempel på efterföljande anrop mot ett API med den genererade *Access Token*.

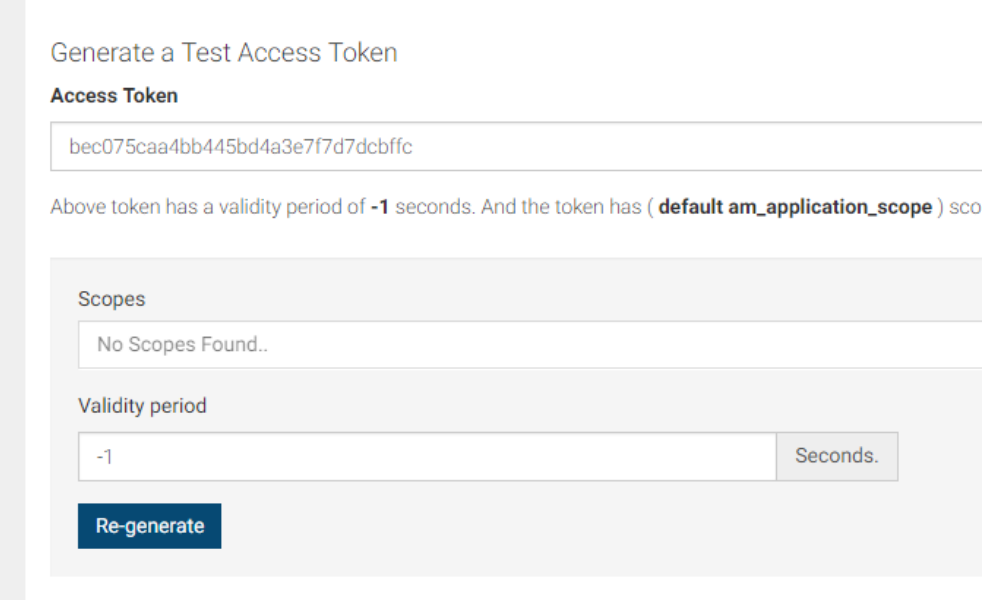
```
curl -X GET --header 'Accept: application/json' --header 'Authoriza-
tion: Bearer 5f996af4-afe6-3438-8e1e-7823018f6898' 'https://api-
ver.lantmateriet.se/hello/2.0/open/test'
```

Curl-exempel för att återkalla en *Access Token.*

```
curl -k -d "token=5f996af4-afe6-3438-8e1e-7823018f6898" -H "Authoriza-
tion: Basic Base64Encoded(Consumer key:consumer secret)" https://api-
ver.lmv.lm.se/revoke
```

ACCESS TOKEN

Figur 2. Skärmbild över hur man genererar ett ”Test Access Token”.



Detta är själva nyckeln som används vid API-anrop.

Default livstid för en genererad *Access Token* är 3600 s (1h).

I gränssnittet går det att generera en ny *Access Token* vilket kan vara bra i ett testskede men den ska helst genereras automatiskt från din applikation.

Om du vill använda en statisk nyckel så anger du *Access token validity period* = -1, vi rekommenderar inte det men kan vara bra i testsyfte och ev. i legacy applikationer där det är begränsade möjligheter till dynamisk nyckelgenerering.

SCOPE

I nedan curl-exempel för generering av *Access Token* så kan parametern *scope* anges, den måste inte vara med men är API:ets resurser skyddade med ett *scope* (läs skyddad med extra behörighet/roller) så måste man begära en *Access Token* för det aktuella *scopet*. I responset syns för vilka *scope* som *Access Token* är giltig för.

```
curl -k -d "grant_type=client_credentials&scope=write" -H "Authoriza-
tion: Basic <Base64(consumer-key:consumer-secret)>" https://api-
ver.lmv.lm.se/token


{"access_token":"5f996af4-afe6-3438-8e1e-
7823018f6898","scope":"write","token_type":"Bearer","expires_in":1373}
```

Scope kan också användas för att göra skillnad på olika klient-instanser som använder samma credentials (*Consumer key* och *Consumer secret*). Det kan behövas ibland då det kan uppstå problem vid generering av *Access Token* från olika instanser samtidigt.

OBS! Om man använder *Scope* för att skilja på klienter och så att de får olika *Access Token* fast de använder samma credentials så ska *scopet* ha prefixet: **device_** t.ex. device_instance1.

Det går att ange flera *scope* och då ska de vara separerade med ett blanksteg t.ex.

```
curl -k -d "grant_type=client_credentials&scope=write device_instance1"
-H "Authorization: Basic <Base64(consumer-key:consumer-secret)>"
https://api-ver.lmv.lm.se/token
```

Exempel - skapa applikation och prenumeration

1. Logga in med ditt användarkonto i API-Portalen.
2. Skapa en applikation för att gruppera Api:er, läs mer under rubrik 2.1 Applikationer.
3. Klicka på det API du vill prenumerera på och välj "subscribe", viktigt att ange vilken applikation prenumerationen avser.

Figur 3. Skärmbild över hur man prenumererar på en ”Application”.

Gå till dina prenumerationer under "Applications" och välj applikation för vilken du vill lista nycklar och API:er. Om inte *Access Token* är genererad för din applikation så kan du generera den i gränssnittet. Själva *Access Token* kan kopieras och användas direkt men helst bara för testning.
Det rekommenderas starkt att *Access token* genereras från din kod och från dina applikationsnycklar, *Consumer key* och *Consumer secret*, läs mer om detta under kapitel Nycklar.

4. Du kan även testa ett API via fliken "API Console". Mer om det går att läsa på WSO2’s wiki under Invoke an API using the Integrated API Console.

Information om att prenumerera på ett API finns att hitta på WSO2’s wiki under Subscribe to an API.

Exempel – Access Token generering

När du anropar ett API så måste en *Access Token* användas och helst ska den genereras dynamiskt från din applikation, gemensamt är att OAuth2 specifikationen används för detta.

Det finns olika sätt enligt OAuth2 att göra det på och nedan finns två beskrivningar när flödet *ClientCredentials* används via java-baserade klienter.

Adresser som används för att generera ny *Access Token*

- VER: https://api-ver.lantmateriet.se/token
- PRD: https://api.lantmateriet.se/token

Exempel 1 - generera Access Token med Spring OAuth2

Spring har tagit fram ett bibliotek för att stödja bl.a. OAuth2 klienter. Använder du det så genereras *Access Token* automatiskt baserat på livslängden på token.

```
<!-- https://mvnrepository.com/artifact/org.springframework.security.oauth/spring-security-oauth2 -->
<dependency>
   <groupId>org.springframework.security.oauth</groupId>
   <artifactId>spring-security-oauth2</artifactId>
   <version>2.1.0.RELEASE</version>
</dependency>
```

Vanligt när man anropar ett API via maskin-till-maskin så används ett OAuth2-flöde som kalls *ClientCredentials*.

1. Skapa upp en Spring-bean av typen *OAuth2RestTemplate* med en bean av typen *ClientCredentialsResourceDetails*, ange din applikations *Consumer key* och *Consumer Secret* som Credentials, läs mer om det ovan under kapitel Nycklar.

```
@Bean
public OAuth2RestTemplate createRestTemplate(ClientCredentialsResourceDetails resource) {
   return new OAuth2RestTemplate(resource);
}

@Bean
public ClientCredentialsResourceDetails createClientCredentialsResourceDetails() {
   ClientCredentialsResourceDetails resource = new ClientCredentialsResourceDetails();
   resource.setClientId(clientId);
   resource.setClientSecret(clientSecret);
   resource.setAccessTokenUri(accessTokenUri);
   resource.setScope(scope); // Om scope används
   return resource;
}
```

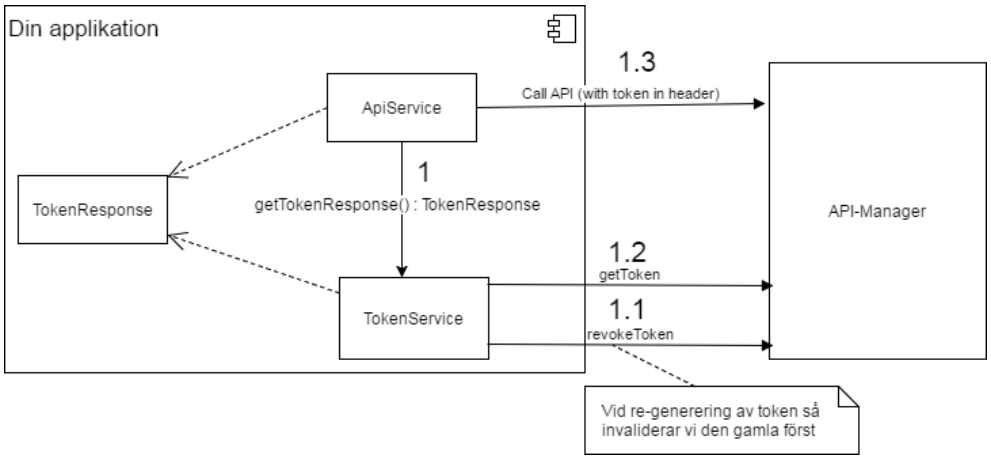
2. Autowire den skapade *OAuth2RestTemplate* och börja anropa API:er, *Access Token* re-genereras per automatik när dess livslängd går ut.

```
@Autowired
private OAuth2RestTemplate restTemplate;

public void callAPI() {
   try {
      apiCall();
   }
   // Fallback if unauthorized token, should only happen if accesstoken
   is
   // by misstake manually re-generated from other places e.g API-Store
   GUI
   catch (HttpClientErrorException e) {
      if (HttpStatus.UNAUTHORIZED.equals(e.getStatusCode())) {
         logger.warn("Unauthorized, genererar ny nyckel");
         restTemplate.getOAuth2ClientContext().setAccessToken(null);
         apiCall();
      }
   }
}

private void apiCall() {
   logger.info("AccessToken: " + restTemplate.getAccessToken());
   logger.info("AccessToken scope: " + restTemplate.getAc-
   cessToken().getScope());
   logger.info("AccessToken expires at: " + restTemplate.getAc-
   cessToken().getExpiration());

   String result = restTemplate.getForObject(API_URL, String.class);
   logger.info("Svar från API-anrop: " + result);
}
```

Exempel 2 - generera Access Token DIY

Nedan exempel visar hur du själv kan generera en *Access Token* dynamiskt från din applikation (server-side) med ett OAuth2-flöde som kalls ClientCredentials.

Figur 4. Flödesschema över hur man dynamiskt genererar en ”Access Token”.



Exemplet baseras på java och spring framework.

- Vi har en class *Token Service* som hanterar regenerering av *Access Token* mha *Consumer key*, *Consumer secret* (dessa bör hållas skyddade i applikationen).
- I *Token Service* injectas *Consumer key*, *Consumer secret* från properties-fil.
- Ett hjälpobjekt *Token Response* används som informationsbärare.
- *Api Service* gör själva anropet mot valfritt api med *Access Token* från *Token Service* och *Token Response.*

TOKEN RESPONSE

Informationsbärare av *Access Token* information.

- *access_token* - själva api-nyckeln
- *expires_in* - anger i sek. hur länge *access_token* är gällande
- *expiresAsDate* - anger i datumformat hur länge nyckeln gäller

```
public class TokenResponse {
   private Long expires_in;
   private String access_token;
   private Calendar expiresAsDate;

   public void setExpiresAsDate() {
      expiresAsDate = Calendar.getInstance();
      // Om livstid är satt till oändligt, lägg på 50 år...
      if (expires_in > Integer.MAX_VALUE) {
         expiresAsDate.add(Calendar.YEAR, 50);
      } else {
         expiresAsDate.add(Calendar.SECOND, expires_in.intValue());
      }
   }
...
   // Dra bort 3 sek för att få lite marginal
   public void setExpires_in(Long expires_in) {
      this.expires_in = expires_in - 3;
   }
   public Calendar getExpiresAsDate(){
      return expiresAsDate;
   }
...
}
```

TOKEN SERVICE

Genererar ny *Token Response* via REST-anrop till API-manager. *Token Response* cachas och genereras endast om den har timat ut. Innan en re-generering av ny *Token Response* så invaliderar vi gamla *Access Token*.

Vi injectar properties och skapar upp *client Credentials*, en base64-string baserad på *consumer Key* och *comsumer Secret*.

```
@Service
public class TokenService  {
   @Value("${host}")
   private String host;

   @Value("${consumerKey}")
   private String consumerKey;

   @Value("${consumerSecret}")
   private String consumerSecret;

   @Autowired
   protected RestTemplate restTemplate;

   private String clientCredentials;
   private TokenResponse tokenResponse = null;

   @PostConstruct
   public void initCredentials() throws Exception {
      clientCredentials = "Basic " + Base64Utils.encodeToString((con-
      sumerKey + ":" + consumerSecret).getBytes());
   }

   public AccessToken getAccessTokenObject() {
      if (accessToken != null && Calendar.getInstance().after(ac-
      cessToken.getExpiresAsDate())) {
         revokeToken(accessToken.getAccess_token());
         accessToken = getTokenByClientCredentials();
      } else if (accessToken == null) {
         accessToken = getTokenByClientCredentials();
      }
      return accessToken;
   }

   /**
```

```
* curl -k -d "grant_type=client_credentials" -H "Authorization:
Basic Base64Encoded(Consumer key:consumer secret)"
https://api.lmv.lm.se/token

*/

private TokenResponse getTokenByClientCredentials() {

   // Set header Authorization = "ConsumerKey:ConsumerSecret" as
   base64

   HttpHeaders headers = new HttpHeaders();

   headers.add("Authorization", clientCredentials);

   // Set params "grant_type=client_credentials"

   MultiValueMap<String, String> vars = new LinkedMulti-
   ValueMap<String, String>();

   vars.add("grant_type", "client_credentials");

   HttpEntity<MultiValueMap<String, String>> request = new
   HttpEntity<MultiValueMap<String, String>>(vars, headers);

   // Call endpoint

   ResponseEntity<TokenResponse> response = restTem-
   plate.postForEntity(host + "/token", request, Token-
   Response.class);

   return response.getBody();

}


/**

* curl -k -d "token=<ACCESS_TOKEN_TO_BE_REVOKED>" -H "Authorization:
Basic Base64Encoded(Consumer key:consumer secret)"
https://api.lmv.lm.se/revoke

*/

public boolean revokeToken(String tokenToBeRevoked) {

   // Set header Authorization = "ConsumerKey:ConsumerSecret" as
   base64

   HttpHeaders headers = new HttpHeaders();

   headers.add("Authorization", clientCredentials);

   // Set params "token=<ACCESS_TOKEN_TO_BE_REVOKED>"

   MultiValueMap<String, String> vars = new LinkedMulti-
   ValueMap<String, String>();

   vars.add("token", tokenToBeRevoked);

   HttpEntity<MultiValueMap<String, String>> request = new
   HttpEntity<MultiValueMap<String, String>>(vars, headers);

   // Call endpoint

   ResponseEntity<String> response = restTemplate.postForEntity(host
   + "/revoke", request, String.class);

   if (response.getStatusCode().is2xxSuccessful()) {

      log.debug("revoke of accessToken: {} was successful", tokenTo-
      BeRevoked);

      return true;
```

```
        }
        log.warn("revoke of accessToken: {} was NOT successful", tokenTo-
        BeRevoked);
        return false;
    }
}
```

APISERVICE

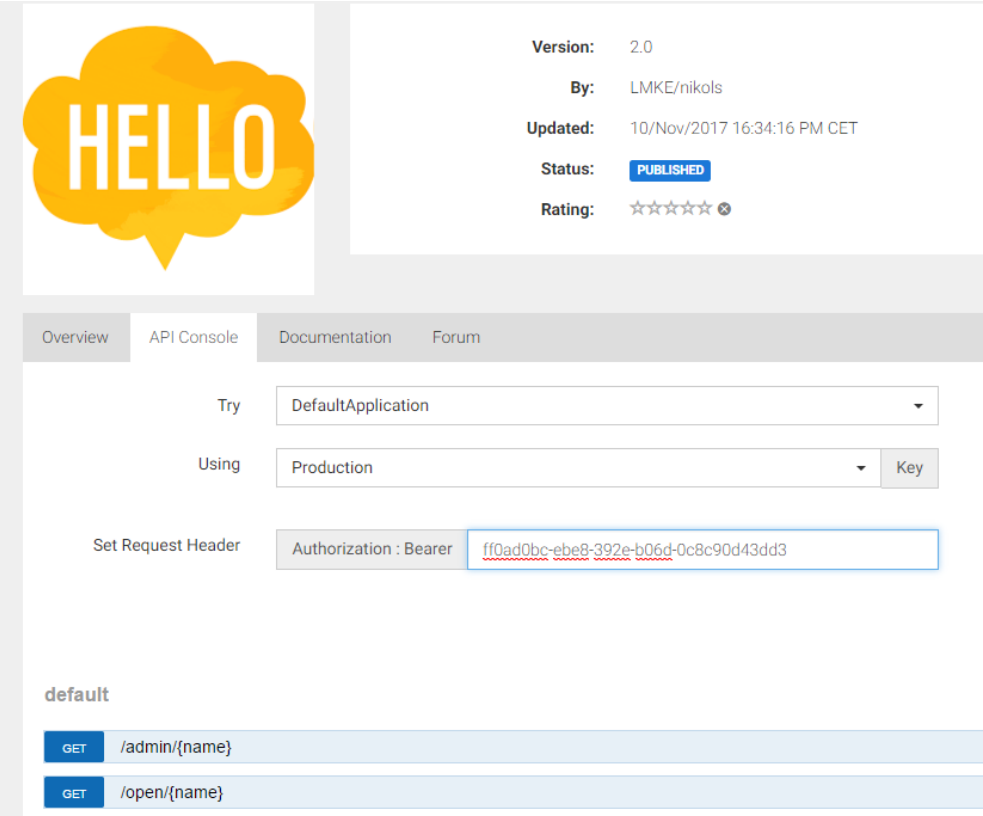
Själva service som anropar aktuellt api med Token från *Token Service*.

```
@Service
public class ApiService {
    private static final Logger log = LoggerFactory.getLog-
    ger(ApiService.class);
    @Value("${host}")
    private String host;

    @Autowired
    protected RestTemplate restTemplate;

    @Autowired
    private TokenService tokenService;

    public void callApi() {
        ResponseEntity<String> apiResponse = null;

        // Get accessToken
        String accessToken = tokenService.getAccessTokenObject().getAc-
        cess_token();

        try {
            // Call API
            apiResponse = callApi(accessToken);
        }
        // Fallback if unauthorized token, should only happen if ac-
        cesstoken is
        // by misstake manually re-generated from e.g API-Store GUI
        catch (HttpClientErrorException e) {
            if (HttpStatus.UNAUTHORIZED.equals(e.getStatusCode())) {
                log.warn("Unauthorized, genererar ny nyckel");
                accessTokenUtils.revokeToken(accessToken);
                accessToken = accessTokenUtils.getAccessTokenString();
                apiResponse = callApi(accessToken);
```

```
            }
        }

        log.info(apiResponse.getBody());
    }

    private ResponseEntity<String> callApi(String accessToken) {
        HttpHeaders headers = new HttpHeaders();
        headers.add("Authorization", "Bearer " + accessToken);
        HttpEntity<String> entity = new HttpEntity<String>(headers);
        return restTemplate.exchange(host + "/hello/2.0/open/Kalle",
        HttpMethod.GET, entity, String.class);
    }
}
```

Demo API – Hello

Figur 5. Skärmbild över ett publikt Demo-API.



Det finns ett publikt Demo API (Hello) publicerat som man kan laborera med.

Testa att skapa en prenumeration på det och gå in i API Console för att test-anropa API:et, eller testa att anropa det från din applikation samt testa att

generera nycklar från din applikation först för att se om din nyckelgenerering fungerar.

Notera att resursen */admin* är skyddat med s.k. *scope* dvs för att anropa den resursen så måste *Access Token* ha genererats för det aktuella *scopet*, med *scope* kan man alltså styra extra behörighet på resursnivå.

Curl-exempel på generering av *Access Token* m.h.a. *Consumer key*, *Consumer secret* och *scope*.

```
curl -k -d "grant_type=client_credentials&scope=hello_admin" -H "Authorization: Basic <Base64(consumer-key:consumer-secret)>" https://api-ver.lmv.lm.se/token
```

```
{"access_token":"5f996af4-afe6-3438-8e1e-7823018f6898","scope":"hello_admin","token_type":"Bearer","expires_in":1373}
```

Exempel på att ange *scope* i gränssnittet vid generering av *Access Token*.

Figur 6. Skärmbild över scope.