Security in OpenLMIS

We suggest that the best option would be to implement **Token Based Authentication** in OpenLMIS. Our suggestion is to create new repository “openlmis-auth” with a microservice called AuthService, that would handle both authentication and authorization.
Additionally, there will be an API Gateway (ticket “*OLMIS-761: Reverse proxy server deployment (nginx)”*) that will forward all requests to an appropriate microservice.

Front end clients authentication
When front-end client wants to log in by sending request with his credential, reverse proxy forwards the request to AuthService and if the credentials are correct a token is returned. Client passes his token as a header in every request sent to API Gateway. API Gateway validates this token by calling AuthService. If it is valid, API Gateway routes request to appropriate service.

Authentication between microservices
Microservices authenticate each other also by using tokens. When one service is calling another, it passes a token that he received from client. The other service validates the token by calling AuthService.

Authorization
User can have one of two roles in OpenLMIS - User or Admin. When he is authenticated and his request gets passed to a microservice, this microservice can ask AuthService whether user has rights to perform given action.

Tokens format
We suggest using **JSON Web Tokens** (JWT) as a format of tokens. They are signed, so they can be transported between client and service with the guarantee that its content is not changed. When using shared key to sign token, we can easily verify the authenticity of the token. Shared key should be saved in configuration file.

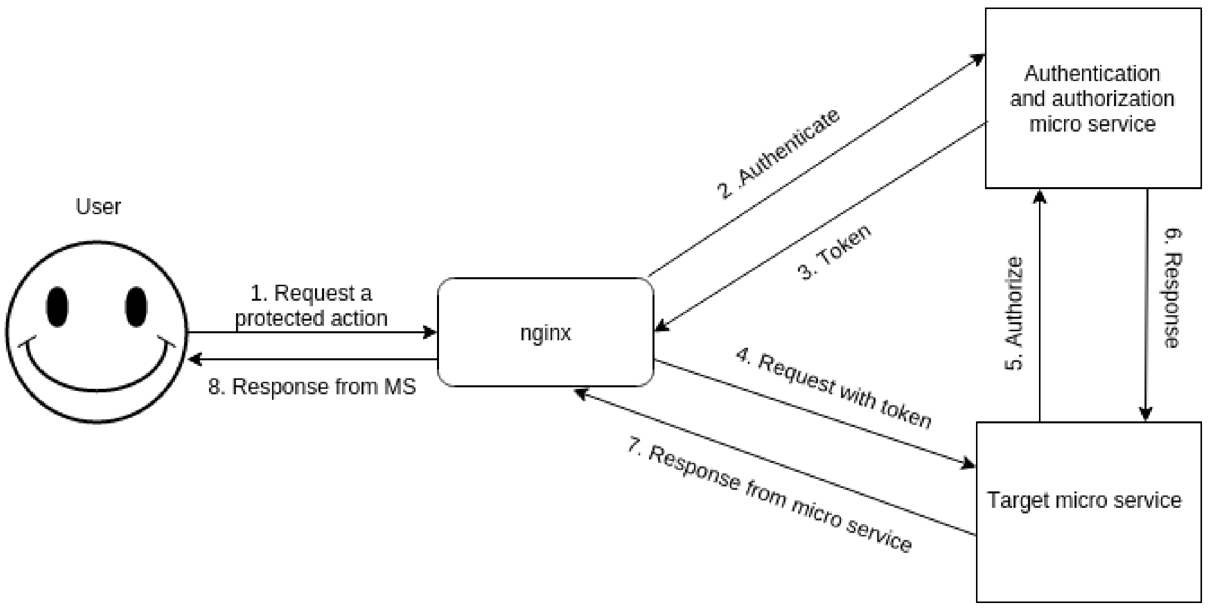
OAuth2
OAuth 2.0 is a standard that applications can use to provide client applications with a ‘secure delegate access’. OAuth works over HTTP and authorizes Devices, APIs, Servers and Applications with access tokens. In our implementation we suggest to use Spring Security OAuth2 with JWT as the format of the OAuth2 token.

For our point of view OAuth 2.0 defines the following roles of users and applications:

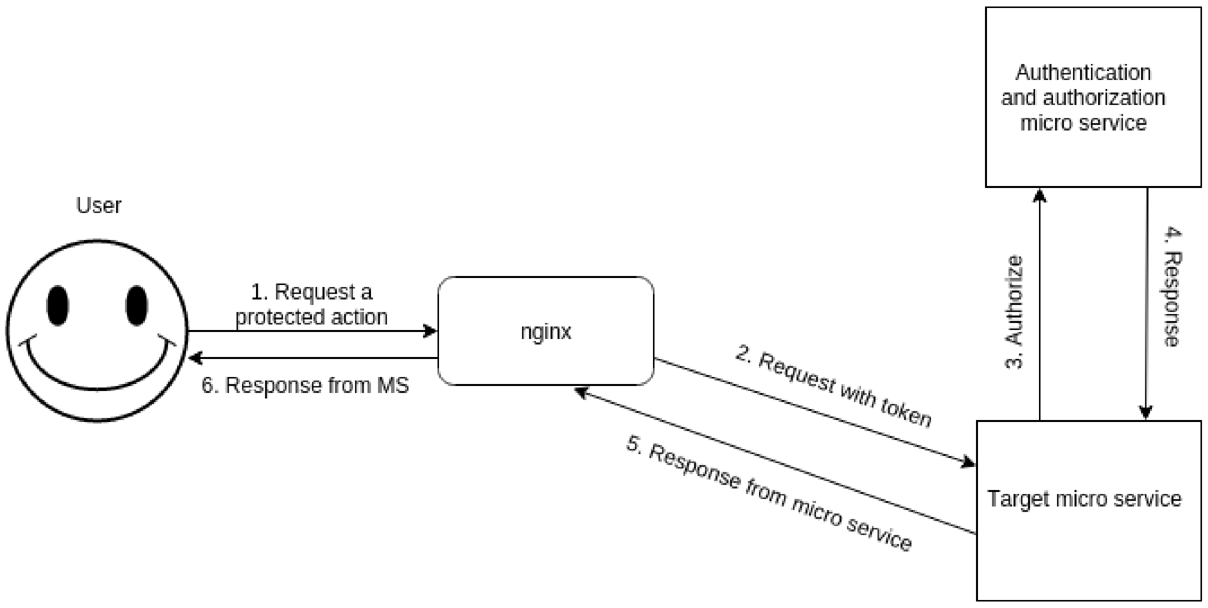
- Client
- API Gateway (Nginx)
- AuthService
- Target Service

We can define authorization flow for two cases:

1. Unauthenticated Client request a protected action.



2. Authenticated Client request a protected action.


User
1. Request a protected action
6. Response from MS
nginx
Authentication and authorization micro service
3. Authorize
4. Response
2. Request with token
5. Response from micro service
Target micro service

As you might notice the only difference is that user was or was not authenticated. It means that User already has (or doesn’t have) an access token stored.

Spring Security OAuth

Spring Security OAuth provides support for using Spring Security with OAuth1 or OAuth2 using standard Spring and Spring Security programming models and configuration idioms. (API documentation: http://docs.spring.io/spring-security/oauth/apidocs/index.html).

Recommended way to get started using Spring Security OAuth is with a dependency management system. We just need to add this line to build.gradle:

```
dependencies {
    compile
'org.springframework.security.oauth:spring-security-oauth2:2.0.10.RELEASE'
}
```

For more information about Spring Security OAuth please read the presentation about security for microservices written by Dave Syer, the lead of Spring Security OAuth:
https://speakerdeck.com/dsyer/security-for-microservices-with-spring

JSON Web Tokens
JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. The information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret or a public/private key pair using RSA.

JSON Web Tokens consist of three part separated by dots (.), which are:

- Header
- Payload
- Signature

Header typically consists of two parts: the type of the token, which is JWT and the hashing algorithm being used, such as HMAC SHA256 or RSA.
In OpenLMIS we suggest using the following header:

```
{
      “alg”: ”HS256”,
      “typ”: “JWT”
}
```

Payload is the second part of the token, which contains the claims. Claims are statements about the entity (for now the user) and additional metadata. There are three types of claims: reserved, public and private claims.

- Reserved claims: These is a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: iss (issuer), exp (expiration time), sub (subject), aud (audience) and others.
- Public claims: These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.
- Private claims: These are the custom claims created to share information between parties that agree on using them.

For our needs we suggest the following payload example:

```
{
      "iss": "OLMIS",
      “exp”: 1300819380,
      “name”: “Username”
}
```

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header and sign that. The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn’t changed along the way.

For OLMIS, we suggest to creating signature in the following way:

```
HMACSHA256(
      base64UrlEncode(header) + “.” +
      base64UrlEncode(payload),
      secret)
```

The output is three Base64 strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact when compared to XML-based standards such as SAML.

The following shows a JWT that has the previous header and payload encoded and it is signed with a “*qwertyuiopasdfghjklzxcvbnm123456*” key:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiIiLCJpYXQiOm51bGwsImV4cC
I6bnVsbCwiYXVkIjoiIiwic3ViIjoiT0xNSVMiLCJuYW1lIjoiVXNlcm5hbWUiLCJhZG1pb
iI6ImZhbHNlIiwicm9sZXMiOiJ7IFwiVXNlclwiIH0ifQ.MFlVg6Fm5bJluTx89YXs11ojH
kizsYEXYiYWZjKbREU
```