Authentication and Authorization of End User in Microservice Architecture

IOP Conf. Series: Journal of Physics: Conf. Series **910** (2017) 012060 doi:10.1088/1742-6596/910/1/012060

Authentication and Authorization of End User in Microservice Architecture

Xiuyu He and Xudong Yang

Department of Computer Science and Technology, Beijing University of Posts of Telecommunications, Beijing, China
Email: xyhe@bupt.edu.com, xdyang@bupt.edu.com

Abstract. As the market and business continues to expand; the traditional single monolithic architecture is facing more and more challenges. The development of cloud computing and container technology promote microservice architecture became more popular. While the low coupling, fine granularity, scalability, flexibility and independence of the microservice architecture bring convenience, the inherent complexity of the distributed system make the security of microservice architecture important and difficult. This paper aims to study the authentication and authorization of the end user under the microservice architecture. By comparing with the traditional measures and researching on existing technology, this paper put forward a set of authentication and authorization strategies suitable for microservice architecture, such as distributed session, SSO solutions, client-side JSON web token and JWT + API Gateway, and summarize the advantages and disadvantages of each method.

1. Introduction

For any mature system, a complete authentication and authorization of end user module is essential. It protects the boundaries of the system. what is the difference between authentication and authorization?

Authentication is the process of verifying who you are. When you log into a PC with a user name and password you are authenticating. Authorization is the process of verifying that you have access to something. For example, some resources only allow the administrator to view but not the ordinary users.

1.1 Auth in Monolithic Application

In a traditional monolithic architecture project, User's requests are handled within a single process in the backend. Put a filter in the system boundary to verify the identity and access, and determine the response or distribution of the request. Figure1 is a typical monolithic architecture. As HTTP is a stateless protocol, it is usually based on the session that the server generates for the client to manage the user's status.

Here is the session control process:

1. The client provides authentication credentials.
2. The server valid the credentials
   - Re-certification if verification failed.
   - Generate session if verification successful.
3. The client requests the resource with session
4. The server gets session of the user by session and response resource if the user has access

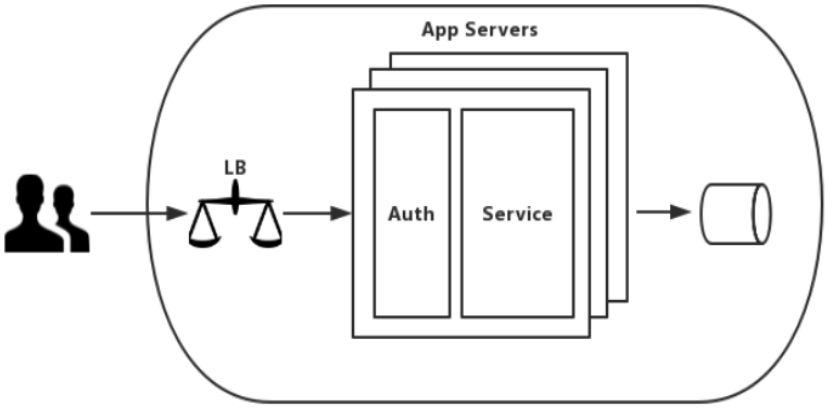

Figure 1 Monolithic auth architecture

The advantages of the session-based system are simple easy to implement and exposed less entrance to the system. But there are many challenges as well. The server needs to save the session in memory, which may cause high memory usage and reduce performance. On the other hand, the authentication feature and other system's features mix together, high coupled results in reducing scalability and flexibility of the system. As the traffic increases, the system needs to deploy multiple nodes to load balance. Share a session in multiple nodes is a problem. Besides, the session-based system uses cookie at most time, so it should deal with some cookie-based attacks from client.

1.2 Distributed Session Management

As the feature of the system becomes more complex and the number of user increases, application deployed in single machine is unable to handle the pressure. From a single node to a cluster, the multiple nodes must share a session when they use session-based auth. There are some solutions about distributed like sticky session, session replication, session centralized management and so on.

- Sticky session ensures that all the subsequent request will be send to the server who handled the first request corresponding to that request.
- Session replication means each server saves session data, and synchronizes through the network. So it will be affected by the network situation.
- Centralized management adds a specific server to manage session. Every service request session from the session server.

In any case, the distributed session is complex to design and difficult to maintain.

1.3 Token-Based Auth

The token-based authentication system allows users to enter their username and password in order to obtain a token which allows them to fetch a specific resource without using their username and password. Once their token has been obtained, the user can offer the token which offers access to a specific resource for a time period to the remote site. The Figure 2 simply shows the process of token-based authentication.

By using token, there is no need to keep a session store; the token is a self-contained entity that conveys all the user information. In addition, the token is stateless so it’s easy to deal with server side scalability. The token is adapted to different client like browser and mobile. In the meantime, it can effectively avoid CORS and CSRF attack.

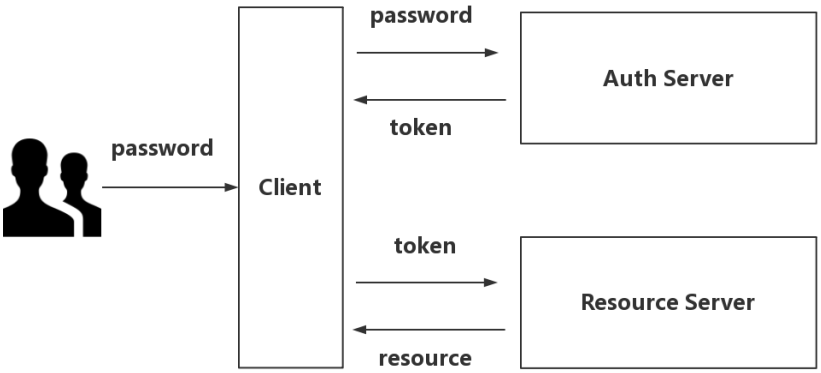

Figure 2 Token-based auth

2. Auth Challenges in Microservice

2.1 What is Microservice Architecture

While there is no standard formal definition of micro service. In the words of Fowler & Lewis [25], a micro service is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. [1]. Figure 3 is example architecture of microservice.

We like microservice for a bunch of reasons.

- Composing functionality
- Self-contained services
- "Bounded context"
- Independent scaling
- Independent deployment
  - Containers
  - Kubernetes, Mesos
- Localized failures
- Prefer statelessness
  - Not rely on HTTP sessions

2.2 Auth Service in Microservice

There is a simple way to deal with the authentication and authorization in microservice that is to imitate the way of monolithic structure. Each service uses own database or shared database that stores credential data and implements own function to verify the the user independently. This way is easy to understand but has several deficiencies. First, when joining a new service in the system every time, we must re-implement the auth function for the new service. Second, when we use a single shared user database, it will have single point of failure problem. Also, this way is against to the single responsibility principle of micro service.

In order to improve the above architecture, and to adapt to the microservice design principles, we put a separate auth services in the system like Figure 4. The auth service focuses on the authentication and authorization of the user. Other services authenticate the user's identity and authority by interacting with the auth service. As a result, each service is focused on its own business, while improving the scalability and loosely coupled of the system.

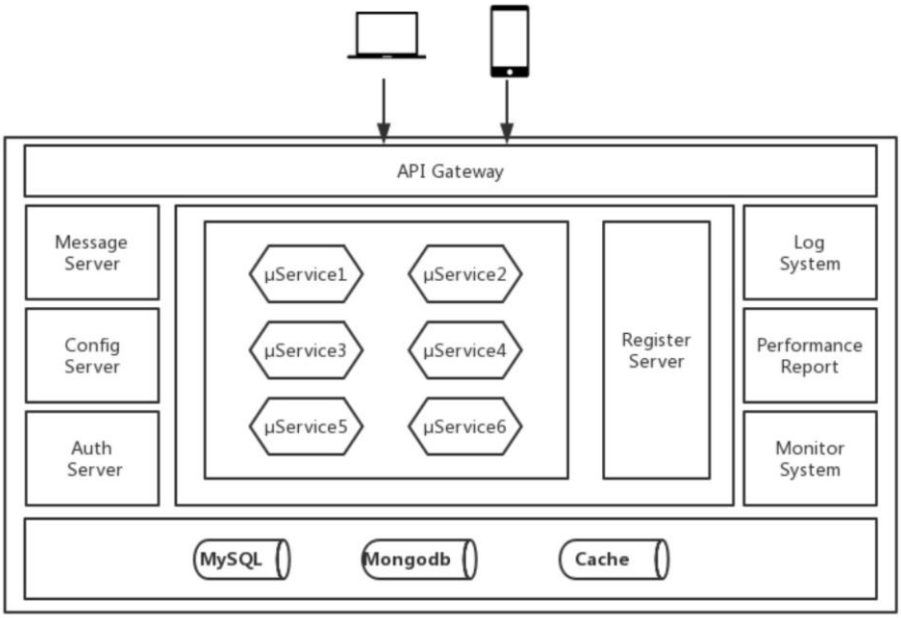

Figure 3 Microservice architecture

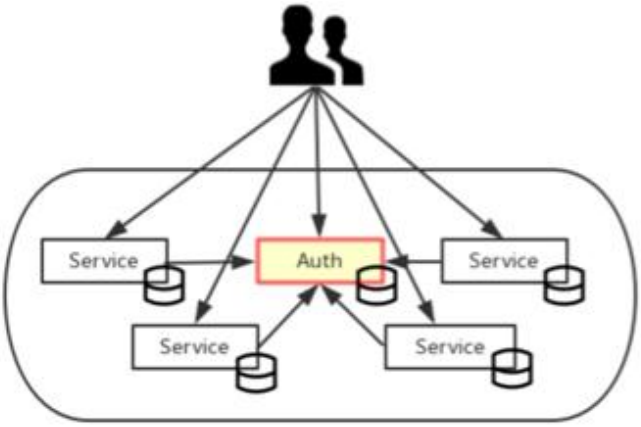

Figure 4 Auth service in microservice system

2.3 What do We Want

- Security

  Security means that users are secured identified. The sharing of information within microservice could be a concern depending on how we actually deploy the system.

- Stateless services

  We want stateless services with multiple instances in running because we are deploying in the cloud and anything can fail at any time.

- No single point of failure

  As mentioned in the previous section, shared databases solution has the problem of SPOF.

- No inherent bottlenecks
- Ability to log out
- Integration with microservices
- Simple to implement

3. Using Distribute Session

In the first part we mentioned the traditional session-based authentication and authorization method and

distributed session management. It is well known that microservice is distributed architecture. This section explains how to implement the distribution session solution in micro services. Figure 6 is the architecture.

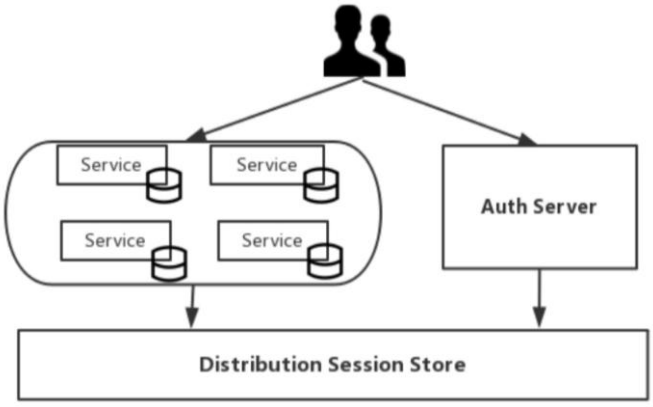

Figure 5 Distribution session solution architecture

User requests access for some services, service redirects to auth server if there are no sessionid in the request. Auth requires the user to provide credentials, generates session for the user and store it in session store, then return sessionid to user. User re-requests access for services with sessionid. Services query session store with sessionid to get user's status and make an appropriate response.

Distribution session solution is feasible but is little complicated to implement as mentioned in first part. The next will introduce several token-based solutions about authentication and authorization in microservice.

4. Using SSO Server

Single sign-on (SSO) is an authentication process that allows a user to access multiple applications with one set of login credentials. In the microservice, each service can be seen as an application. When entering the system, only need to certify the user once.

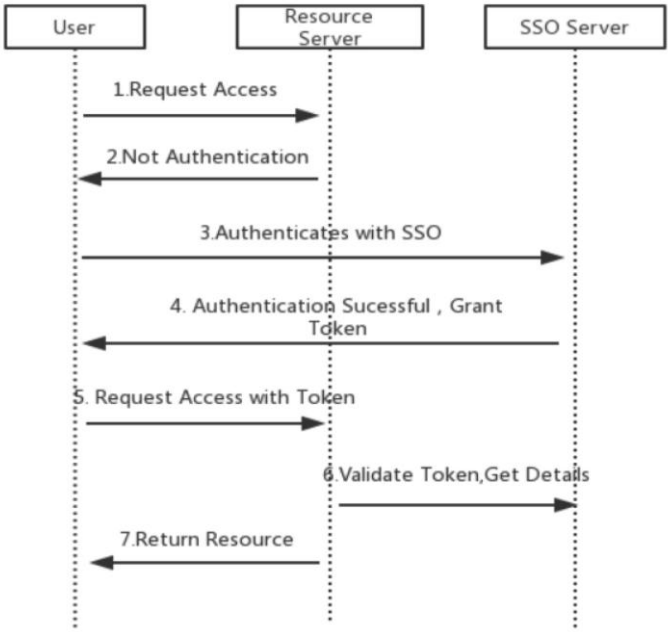

Figure 6 SSO solution timing diagram

Let's take a look at the process of a user want to get resources from microservice. User request

access for the resources server. Resource server check if the request carrying a token. If there is a token, valid token from SSO server, and get user details. Judge whether the user has access to the resource and make appropriate response. If the token is not existed nor verified, redirect the request to SSO server. User provides the username and password to get the token and re-visit the resource with token. Figure 5 is the SSO solution timing diagram.

In the SSO solution, "login" state is usually opaque, all requests will finally arrive SSO server to valid the token. This will cause chatty traffic and single point of failure (SPOF) problem in SSO server. when switch the service inside the system, it need to valid the user again with SSO server. We can optimise with local "login" caching to solve the problem.

5. Using JSON Web Token

5.1. JSON Web Token(JWT)

JSON Web Token (JWT)[2] is an information transfer protocol based on an open standard (RFC 7519) that consists of Header, Payload and Signature.

1. Header: The header typically consists of two parts, one of them is token type, namely JWT, and another one is hashing algorithm used, such as HMAC SHA256 or RSA. The format of JSON header is as follow:

```
{
      "alg": "HS256",
     "typ": "JWT"
 }
```

Then, this JSON is Base64Url encoded to form the first part of the JWT.

2. Payload: Second part of token which consists of claim. Claim is the statement about an entity (usually users) and supplementary metadata.

```
{
  "exp": 1494428560,
  "name": "John Doe",
  "authorities": [
    "ROLE_ADMIN",
    "VIEW_RESOURCE"]
    …
}
```

The payload is then Base64Url encoded to form the second part of the JSON Web Token.

3. Signature: This section is to signature the Header and Payload sections according to the algorithm specified in the Header section to prevent modification maliciously.

For example, if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

5.2. Client-Side JWT Solution

When the user sends request to the resources server, the resource server redirect to auth server if there is no token or token is expired. User sends credentials to auth server. Auth server valid the credentials and generate JWT token for the user which contains user's authority, expired time and so on. User re-requests resource server with JWT token this time. Resource server decodes the token and verifies user permission. Then make an appropriate response. Figure 7 is JWT token solution timing diagram.

Due to the signature, JWT can validation itself rather than request auth server. So this will reduce auth server pressure and chatty traffic. Another benefit is that system is easy to expand for the stateless of the token.

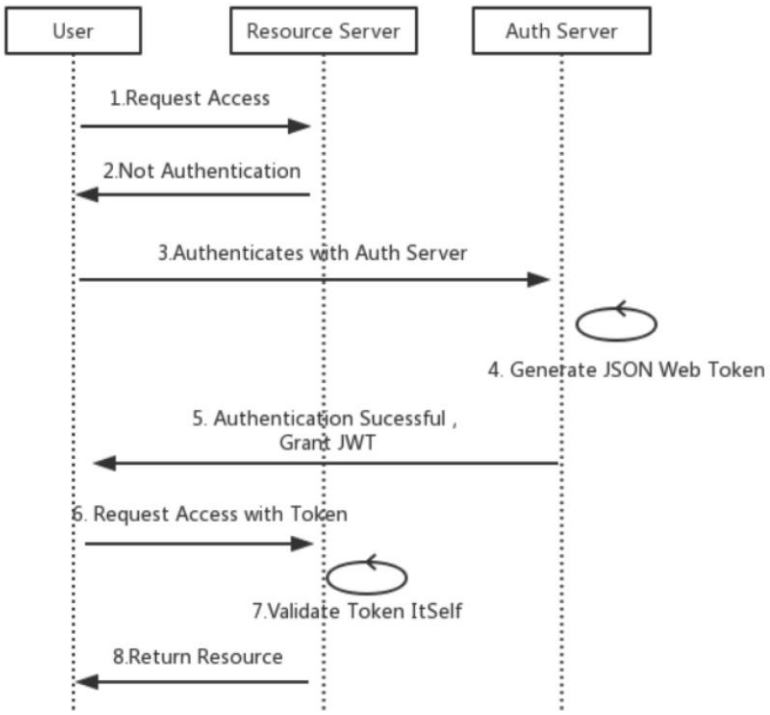

Figure 7 JWT token solution timing diagram

It's a good solution so far. But how to logout. We know that JWT contains an expire time. Before the expire time, the token is always valid. When we want logout, the simple way is to remove the token from client side. But this is not safe, because the server side has not been aware. Another way is periodically check with auth service in the server side. This will cause additional network traffic.

6. Using JWT+API Gateway

In the microservice architecture of Figure 3, we can found there is an API gateway in the boundary of the system. An API gateway provides a single, unified API entry point across one or more internal APIs.

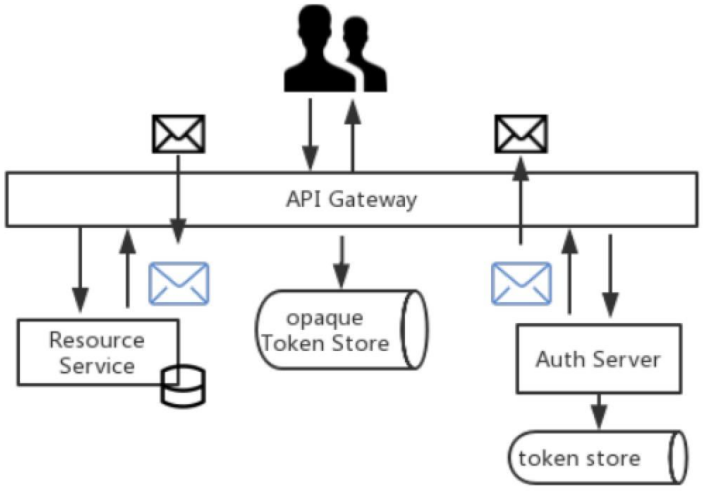

Figure 8 JWT+API gateway solution architecture

The most important thing about logout is to invalidate the token on server side. As every request will go through API gateway, so we can use it to solve the logout problem. The key step is that API gateway

translates JWT to an opaque token. Figure 7 is the JWT+ API gate solution architecture and Figure 8 is the timing diagram.

Most steps of JWT+API gateway solution are the same as the solution in V. The only difference is that the gateway has transferred the token. Auth server generate a JSON web token for the user and deliver it to gateway. Gateway transfer the token to an opaque token that only itself can resolve. Then store the relationship of the opaque token and origin token in a token store and send the opaque token to client. When the gateway receives an opaque token from client, transfer it to the origin token, then forward the origin token to the resource service. With that transfer step, we can remove the token relationship in the API gateway token store, So the client need to re-request a token. This is a brilliant solution for the logout issue.

By comparing with the traditional monolithic architecture, this paper explores several solutions of authentication and authorization in microservice architecture. Distributed session solution is similar to the traditional auth way but its complex to implement and maintain. SSO server will cause SPOF and traffic problem. Using API gateway to improve the client-side JWT solution can solve the logout problem. So there is no absolute right solution but only suitable options for your system.

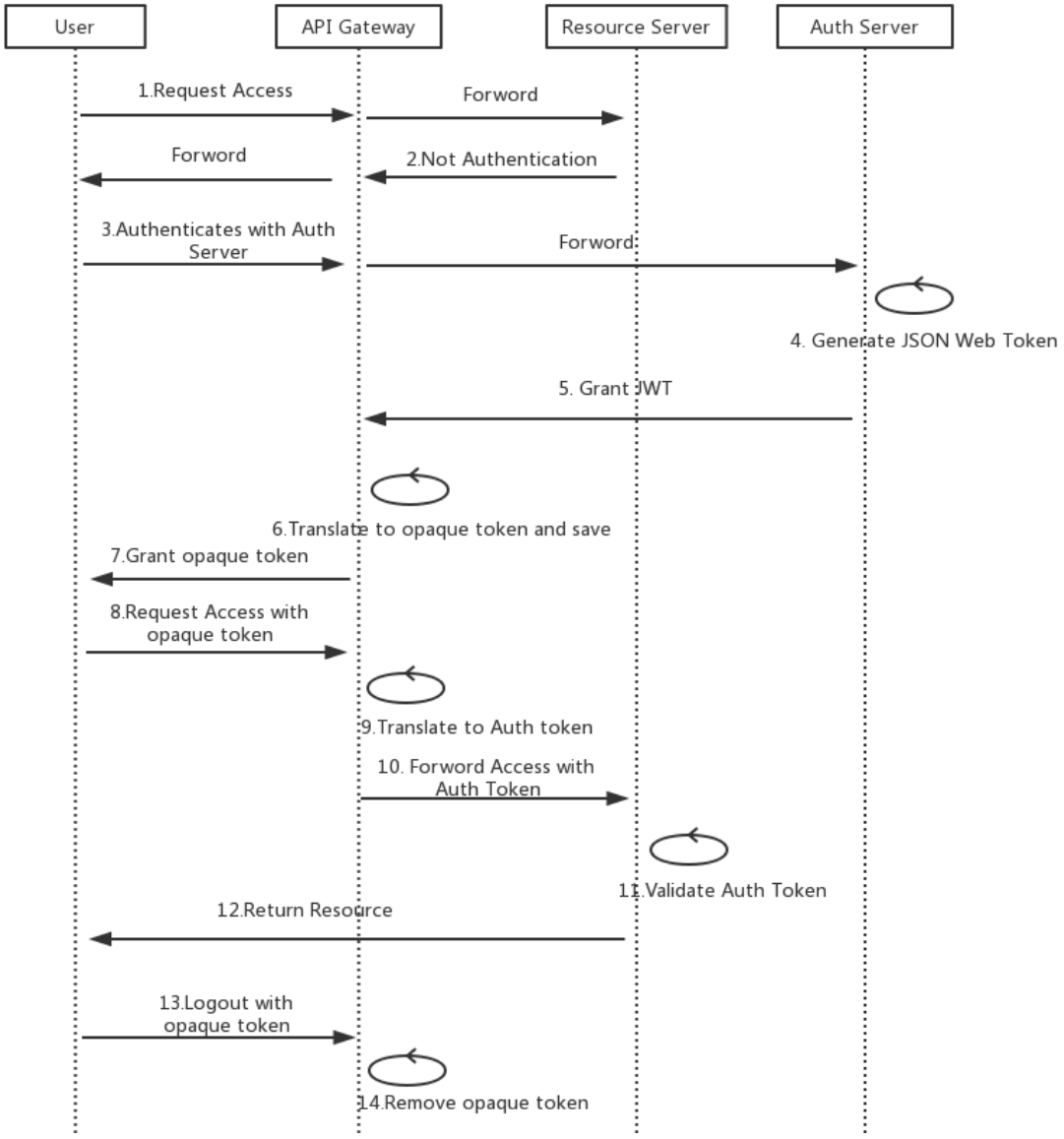

Figure 9 JWT+API gateway solution timing diagrConclusions

7. References

[1] Fowler, M. & Lewis, J. Microservices, 2014 Retrieved From Martin Fowler: http://martinfowler.com/articles/microservices.html.
[2] Bradley J,Sakimura N,Jones M.JSON Web Token(JWT)[Jl,2015.
[3] J. Phys.: Conf. Ser. Flexible session management in a distributed environment, 2010
[4] Peng D,Li C,Huo H.An Extended Username Token-based Approach for REST-style Web Service Security Authentication[C]. Computer SCience and Information Technology,2009.ICCSIT 2009.2nd IEEE International Conference on.IEEE,2009:582—586.
[5] Dmitry Namiot, Manfred Sneps-Sneppe. "On Microservices Architecture".International Journal of Open Information Technologies ISSN: 2307-8 162 vol. 2, no. 9, 20 14. p24-27
[6] Chris Schmidt. “Token Based Authentication - Impelementation Demonstration”. Workshop Friend of a Friend, Social Networking and the Semantic Web, 1st-2nd September 2004.
[7] X. Li. “The design of digital campus unified identity authentication system based on web services”, Applied Mechanics and Materials Vols. 427-429 pp. 2301-2304, 2013.