Information system development for restricting access to software tool built on microservice architecture

Olga Safaryan[1,*], *Elena* Pinevich[1], *Evgenia* Roshchina[1], *Larissa* Cherckesova[1], *Nadezhda* Kolennikova[1]

[1]Don state technical University Rostov–on–Don, 1, Gagarin square, Russia

Abstract. The article discusses issues related to improving the security of an information system by introducing an authentication system into distributed information systems, developing and implementing a secure software architecture built on a microservice architecture. That kind of architecture ensures the prevention of unauthorized access to confidential information processed in the application. Achieving this goal it is necessary to use authorization methods, the basics of building secure applications and the basics of database management systems.

1 Introduction

The promising direction in the development of the modern applications architecture is the construction of systems using microservice architecture, where the application is presented in the form of several small services, each of which works in the separate stream, and the modules communicate with each other via the HTTP protocol. This principle allows users to create projects whose modules are interconnected weakly, which provides good horizontal scalability.

Due to the fact, that communication between services is carried out through network interaction, it is necessary to secure each of the services so that a potential attacker could not gain access to data that does not belong to him.

Since microservice architecture is new principle in the sphere of applications construction, development of access control system and the use of access models for such software are poorly studied question that needs to be studied. Therefore, research of this issue is actual problem and an urgent task.

If to compare the server for monolithic application with microservice server, the difference is that in the approach associated with the principle of microservice architecture, the whole software application is constructed as set of independent services. Each of them works in isolation, and communicates with the others using special mechanisms. These services are deployed independently using fully automated environment.

The scientific novelty of the proposed research is the application of the principle of microservice architecture for the elaboration of the safe protected software application

*Corresponding author: safari_2006@mail.ru

architecture, including the development of the system of access control and the application of security models for access to the confidential information.

The research object is the network organization of the components interaction of an information system (IS).

The study subject is increasing the information system security by introducing authentication system in distributed IS.

The research goal is elaboration of the secure software architecture built on microservice architecture that ensures the prevention of unauthorized access to confidential information processed in the application and its implementation.

Achieving this goal it is necessary to use authorization methods, the basics of building secure applications and the basics of database management systems (DBMS). At the same time, it is necessary to solve a number of the following tasks:

– Study the classification of authorization algorithms.

– Identify their advantages and disadvantages.

– Analyze the available authentication algorithms and justify the choice of algorithm to protect data from unauthorized access.

– Develop the architecture of the security layer of the developed software that provides the solution to this goal.

Select programming language with used technology stack and realize the working program application that demonstrates the operating of the developed access control system.

2 Theoretical Basis

Identification and authentication are concepts used when it comes to restricting access of information system entities to the information system objects (to hardware and software resources). General procedure for user identifying and authenticating when accessing it in automated system (AS) is shown in the figure 1.

If the subject authenticity is established during the authentication process, the information protection system must provide this or that access right to this user, depending on which model of access control is most preferable in the current case.

According to system–controlled component, authentication methods can be divided into authentication of communication partners and authentication of data source. Main classification feature of authentication methods is that, which establish the subject identity. Consider the authentication types:

– Using secret information, that only subject knows – password.

– Using the unique material object – token, card, key, etc.

– Using biometric parameters: fingerprints, iris, face model.

– Using information associated with user (his coordinates).

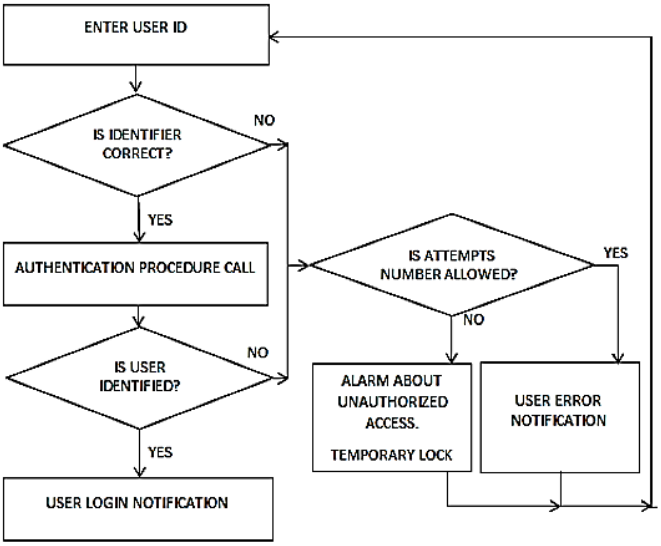


Fig. 1. Diagram of identification and authentication procedure.

Consider these groups. Password authentication, the most common simple and familiar are authentication methods based on passwords – secret identifiers of subjects. Entering his password, the subject sends it to the security module, which, in turn, compares it with the password that is stored inside its database. To ensure enhanced security, the database does not store the password itself in explicit form, but its hash calculated by one of the common methods. It is not the password itself that comes to the module, but the hash of the password, which is calculated on the client side, so that it is not sent explicitly via network information transfer channels.

If the hashes match, the security module either provides access to information system in accordance with access control model used, or records the fact of unauthorized access attempt.

The advantage of this method is the fact that the transfer of secret data in the communication channel is minimized, which reduces the likelihood of information leakage or interception by an attacker.

Authorization by unique identifier, combined identification methods have recently spread, requiring, in addition to knowing the password, the presence of a token, a special material device that confirms the subject authenticity gaining access to the information system [1].

Cards are divided into 2 types: passive (memory cards) and active (smart cards). The most common are passive magnetic stripe cards, which are read by a special device that has a keyboard and processor. When using the specified card, the user enters his identification number. If it coincides with the electronic version encoded in the card, the user gets access to the system. This allows system to identify reliably the person who gained access to the system and exclude unauthorized use of the card by an attacker (for example, if it is lost). This method is referred often to as two–component authentication.

Biometric authentication, authentication methods based on the use of human biometric parameters provide a sufficiently high reliability of object protection, solving the problem of losing passwords and personal identifiers, since these parameters are tied to a person and copying this data is a very difficult task. However, such methods cannot be used to identify processes or data (data objects), because the technology is new, and specialists have not developed a single standard for distributed systems.

Relative high cost of equipment, and its implementation in information systems, also affects. This conditions their use so far only on particularly important objects and systems.

For subject identification, such parameters of the human body as the eye iris, fingerprint or palm, auricle pattern, infrared image of capillary vessels, handwriting, as well as its parameters, sweating secretions, voice timbre (which is widespread in mobile phones) can be used devices and begins to spread in banking systems), and even DNA code.

Authentication by location, new means of authentication is authentication by geo–information, subject's characteristic. This method is realized on the ground of space navigation system – GPS or GLONASS. User having equipment capable of contacting satellites repeatedly sends the coordinates of certain satellites that are for the subject in the reception area. The authentication subsystem, the satellite trajectory, can determine the subject location with fairly high accuracy. High authentication reliability is determined by the fact that satellite orbits are subject to fluctuations, which are difficult to predict. In addition, the coordinates of satellites and users change over time, which reduces the likelihood of interception [2].

Access control, after the successful completion of identification and authentication procedures, it is necessary to provide the subject with access to information system objects in accordance with the access control policy that was adopted and established for specific user. Typically, user privileges are presented in the form of a table, where the list of objects to which access is granted is listed, as well as the rights that the user is endowed with respect to an object. The main types of authorization, identification and authentication algorithms are considered, their strengths and weaknesses are highlighted. The basic models of access control are indicated, thanks to which it is possible to manage access rights for different users.

The main advantages of applications built on microservice architecture are good horizontal scalability and low connectivity of individual modules, which entails convenience in maintaining the software in the future and the predictability of the software behavior as a whole in failure case of one of the modules.

Thus, in view of the prospects of this architecture in the future, it is necessary to develop and implement an authorization system to protect data processed by individual services from unauthorized access.

3 Principles of Microservice Architecture

Basic principles of microservice architecture:

– As program interfaces for interacting with microservices, as a rule, they use the RESTful API and asynchronous messaging via queues (RabbitMQ, ZeroMQ). Microservices allow system to manage information resources using a narrow set of operations: reading and updating its individual parts. All interactions between services are built on the client side without storing on the server. In this regard, it is necessary to adhere to the design for failure pattern [3]. This pattern is extremely necessary, because the network is very unreliable in nature: packets may not reach either due to poorly configured routing, or because of a broken communication channel between endpoints. In this regard, each of the services should be built in such a way that it takes into account these fluctuations in network bandwidth, and it must be able to restore its performance if it was either isolated from all or one of the services with which it interacted due to its architecture was unavailable;

– Unlike monolithic applications that are deployed in one process, in microservice architecture, each standalone unit is deployed in isolated environment, as shown in figure 2. This allows independent development teams involved in different services, providing the opportunity to choose the development environment and stack used technologies for each

of the microservices. This is especially convenient for those cases when different DBMS can work out better in special cases. Oracle can be a great fit for storing a large array of text data, PostgreSQL with its lightweight Postgis extension is suitable for storing geotypes, and NoSQL databases Berkeley DB or Apache Cassandra exist for storing key–value pairs, which is well suited for storing ordinary files;

– Microservices implement the specific need; they provide specific functionality for each particular case. If for different tasks slightly different behavior of the microservice is required, several microservices are created that implement the same functionality and have the same interaction interface, but each of them will have different behavior. One service can implement data transfer without using secure data transfer protocols, while others can use it. This is an excellent tool for implementing error and exception handlers, expanding the typical behavior of algorithms by adding additional behavior scenarios, and supporting new functions [4]. This is extremely important in fast–growing application. In dynamic application, this is crucial.

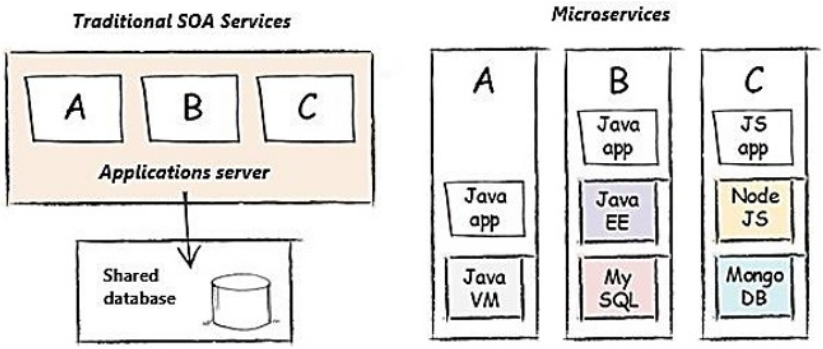


Fig. 2. The structure of the microservice application.

The technical aspects of working with microservices, especially when working with microservices, it must be taken into account that for their full functioning a well-developed network infrastructure is needed in which these services are contained, since it is through network interaction that one service communicates with others [4]. To do this, it is necessary to provide the fault–tolerant topology for the local network between all services: have a backup channel or devices that will replicate the main critical nodes of the system [5]. The second important aspect in working with such an architecture is the simplest horizontal scaling of the application. It should be possible to run several identical services at different hardware capacities so that they replicate each other, thereby fulfilling two goals: increasing the fault tolerance of the hardware and software complex as a whole and increasing reliability due to the possible load distribution between different instances of the same microservices [5, 6]. Together with the possible redistribution of network traffic to different network resources, this can give a significant result in an increase in reliability.

At the formulation of the development task, in order to create an application that meets the above features of this architecture and supports role–based access control, it is necessary to design the application architecture. In such a way, it can replace one service with another without recompiling the source code, or replace the current running service with another one without stopping the work of separate functional.

It was constructed hierarchically so that it was possible to connect security module that restricts access to unauthorized users, and implements load balancing between different service instances that represent and implement the same business logic. This architecture will allow:

– To avoid the unauthorized access by the side of illegal entities to the confidential data of the users;

– To avoid losses associated with both the maintenance of the equipment on which the services are deployed and the software components updating themselves to newer versions;

– Increase the software reliability by providing sufficient resistance to high loads due to the possibility of functional replication without complex manipulations [7–9].

In order to hide the application internal structure from the user, it is necessary to create a single interface for user interaction with the final product. It is proposed also to implement the ability to create multiple instances of the same service to maintain the scalability of the application. This will solve the following problems: software that has such mechanism under the hood will be more faults tolerant, because if one of several services fails, the rest will take over the work.

Directly on the service gateway, it is most advisable to put authentication and authorization means, because it will know the routes to all the final local nodes.

To achieve these goals, it will be necessary to use password authentication and confirmation during any request, since the developed architecture and access to services will only have limited trusted circle of people. Authorization will be necessary for the most part in order not to inadvertently damage the data of users who use this distributed system. Ultimately, the final application architecture will look as shown in the figure 3.

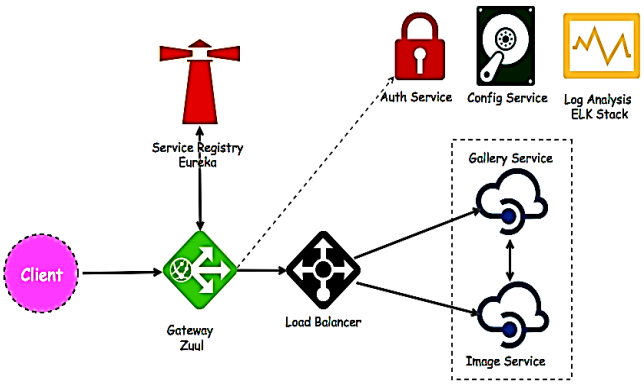


Fig. 3. The final product architecture.

The following functionality will be assigned to the developed service of registration services:

– To register services that appear in the environment;

– To assign the symbolic names for registered services and store them in memory as long as they are alive;

– To send registry of active services to all registered services.

This service will allow being the layer between interacting services, which will talk about which address which service is deployed. In this case, user will only have to set the symbolic service name in the dependencies, which will be used during the operation of another service, and the service registrar will give the address of the desired system object by this name [10].

Service gateway assumes the role of router: according to registry collected by the registration service, it will forward requests to actual addresses that are available in environment. Registration service will send this registry to gateway, which will be stored and updated according to current information.

This service will be the only entry point into the system; it will accept all requests in order to forward them to the final necessary service. There is possibility to supplement additional security functionality to this service: authentication, logging action, requests

distribution between different instances of the same service in order to distribute the load, and requests filtering.

In general, the interaction structure of the two services is shown in figure 4, where each of the services informs itself of the registration service, which, in turn, sends this information to the gateway to correctly address requests between services.

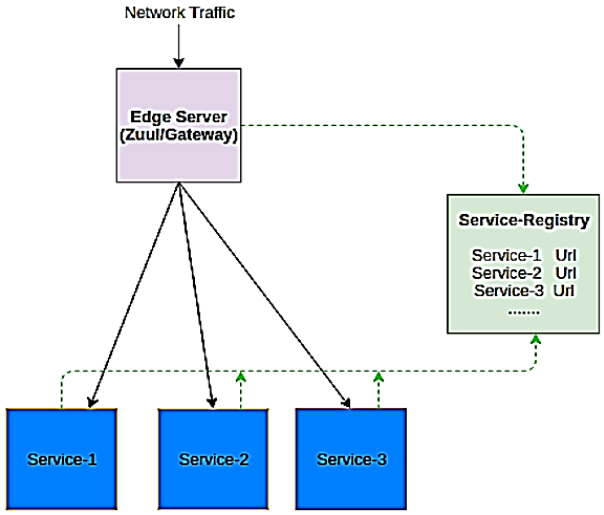


Fig. 4. Scheme of services interaction.

Authentication service will be attached to service gateway and verify the validity of connection established by user.

For these purposes, it will be enough to use confirmation using the token. Authentication on service will occur as follows:

– User sends request for token confirming his identity.

– Server checks username / password pair and sends token.

– User attaches the token every time, thanks to which the server provides the access to the service.

4 Realization of the Functionality

There are two ways to implement the functionality, which restricts access to services of developed architecture.

The first way is that the token can be checked every time in the authentication service. To do this, the gateway must call the third–party service each time before allowing requests to go to any other service. In this case, the security logic lies entirely on the authentication service, which is not bad. Nevertheless, it will be bad, if the system is heavily loaded: unnecessary token verification requests can adversely affect the “useful” channel bandwidth between services: service requests that do not carry the payload from the point of view of the end user will become larger [11].

The second way can be implemented by checking the tokens on the gateway side, while the authentication service will check the credentials and issue tokens. However, from security point of view, both methods are good: neither one nor the other will give access to internal services without authentication.

It is proposed to select the JSON Web Token (JWT) standard as the generated token. These tokens are used typically to transmit authentication data in client – service

applications. They are created by the server, signed with a secret key and sent to the client, who subsequently uses it to confirm their authenticity.

The token consists of three parts: header, payload, and signature. The first two objects are JSON objects of a certain structure, and the third one is calculated based on the first two and the selected algorithm. The first two parts are often encoded in Base64–URL for compact presentation.

This architecture will allow unauthorized users to restrict access to application, which will avoid the following problems:

– Information leak due to the threat of unauthorized access;

– Elimination of damage to information that is not required to achieve goals by issuing the access only to the data that is needed to achieve the set goal.

Program structure should consist of following components:

– Registration service is the service whose functionality consists in registering active services and maintaining the registry of the user's connections.

– Service gateway, which, relying on the registration service, should lay the route to both intermediate nodes, such as the authentication service, and to the final nodes representing the business logic of the application.

– Authentication service that verifies the user's identity and generates a token confirming the user identity.

– Services that represent the business logic of application.

Task of software tool development was solved in the Java programming language. As the main framework used in the conceived architecture, we will use Spring. For ease of development and testing, we will implement each of the services through the Spring Boot framework, which takes care of configuring and launching the container with the service through the development environment. For this reason, we will write the following lines for all modules used in the project:

```
<dependency>
groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter web</ artifactId>
</dependency>
<dependency>
groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>

<dependency>
groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-devtools</artifactId>
<optional>true</optional>
</dependency>

<dependency>
groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
```

In addition, it is necessary to indicate for Maven, which plug–in the assembly should go through:

```
<build>
<plugins>
<plugin>
```

<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven plugin</artifactId>
</plugin>
</plugins>
</build>

Registering individual services can be implemented using Spring Cloud. Inside this class package there is realized class SpringEurekaServer. To connect to service of this class and deploying the Eureka registration server, one should specify the following dependency in the Maven file.

<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-netflix-
eureka-server</artifactId>
</dependency>

The following is class that implements loading the entry point into the Spring application context. After that, in the file, which is a list of key–value strings, we indicate the main class parameters, such as the service name; the port on which it will work, as well as flags indicating that the service should not register itself and declare itself in the registry.

spring.application.name=eureka-server
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false

Eureka service customers will declare themselves for the registration service. Subsequently, the service will send hash of this registry to the final nodes. If the hash, stored on the final services differs, then new version of this registry will be sent to them. The Spring Cloud package has preform for the service gateway – Zuul, which has ability to accept requests, and then to redirect them to the final service nodes.

To start such service, several annotations need to be specified in the main class of the application: EnableEurekaClient and EnableZuulProxy. The first annotation includes Eureka client: after starting the instance, the service will send periodic echo requests to the registration service, thereby declaring itself.

The receiving side will remember where the echo request came from, and will add the record about it in its registry. The lifetime of such record is one and half minutes. The entry in the registry will be erased after that no signals have been received for 90 seconds. This will mean that the node has become unavailable. Specify the main settings for this service in the configuration file:

zuul.routes.auth-service.sensitive-headers=
Cookie,Set-Cookie
zuul.routes.auth-service.strip-prefix=false
zuul.routes.auth-service.path=/auth/**
zuul.routes.auth-service.service-id=AUTH-SERVICE
zuul.routes.gallery-service.path=/gallery/**
zuul.routes.gallery-service.service-id=gallery-service
zuul.ignored-services=*
server.port=8762
spring.application.name=zuul-server
eureka.client.service-url.default-zone=http://localhost:8761/eureka/

In these lines it is specified the service name, the port on which it should be deployed, indicate the address of the registrar service, indicate the services available through this gateway (parameter zuul.routes.auth–service.service–id) and the path (parameter zuul.routes.gallery–service.path), making request with which user can get response from these services.

The implementation of the authorization service, in addition to the main class, which is the starting point of the application, will require three classes more that will expand the library interfaces and implement methods directly responsible for security: generate and sign access tokens.

The UserDetailsServiceImpl class is an implementation of the UserDetailsService interface, which is used to check whether the user that was entered during the authorization request exists or not. If the user does not exist, then a UsernameNotFoundException will be thrown.

For demonstration purposes, several users will be hardcoded in the code. In the case of really working applications, the source of users can be either a query to the database, or some list stored in the file system.

The SecurityCredentialsConfig class extends the WebSecurityConfigurerAdapter library class and represents the class construction with the parameters necessary for the current task. Inside the class, we add filters, set request masks, and initialize the objects responsible for checking the correctness of the data and for calculating the password hash.

The JwtUsernameAndPasswordAuthenticationFilter class is one of those filters that were initialized in the class SecurityCredentialsConfig. The standard parameters and settings are redefined in the body of this filter. After successful authentication, the token is returned to the user in the header, which he must present at each request [12].

To work with services integrated into this architecture, it is necessary to send requests to the gateway. The gateway will check the authorization data and, depending on this, provide access or deny access. In case of failure of this node, the entire application will be inaccessible.

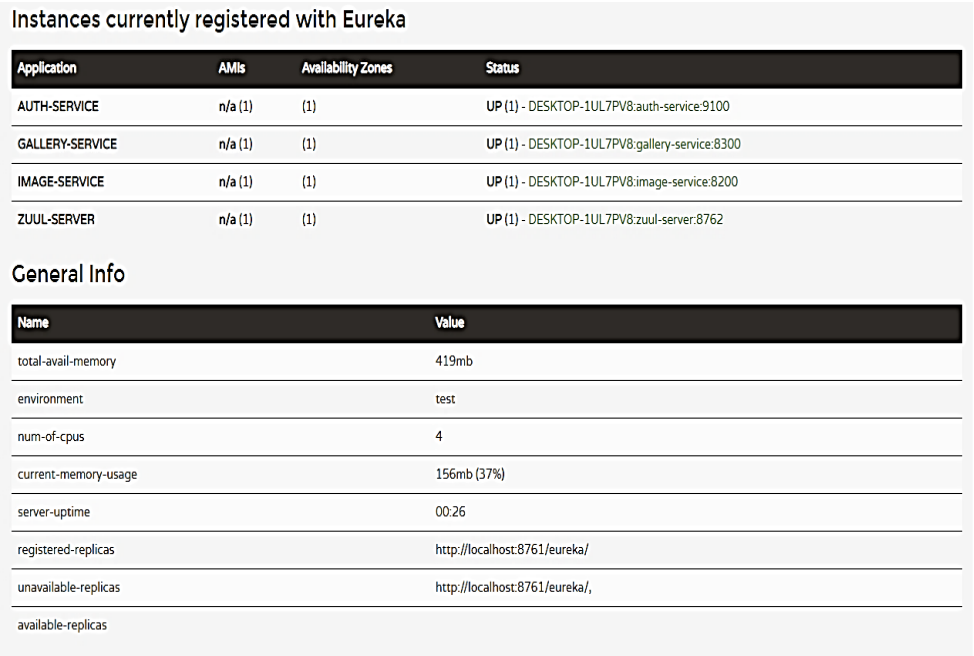
Any services and any number of them can be located on the local network, while access to the public network will have only a gateway. Thus, from the outside it will not be possible to access directly to individual services, they will be available only through the gateway. Run the services that represent the architecture, as well as a couple of services that implement the business logic of the application. On the Eureka service, we will see that they are all registered and ready to work, as shown in the figure 5.

For debugging, we will use Postman, which allows us to construct http requests using convenient graphical interface. In this program tool, when creating the request, there is possibility to fill in the header, indicate its type, and pass the necessary parameters. The transmitted data can be represented either explicitly or in binary form also. After sending a request, it is possible to parse the answer to this request and see its structure in order to understand whether the application is configured correctly. To do this, you must to parse the header with the response body, decode them and present them in a readable form for the user, in order to understand how correctly the backend part works out the applications: in what form and what information is given to the user. Since the user's token is returned in the request header, it will be quite problematic to test the browser.

Let us make a request to the service of gallery–service through the created gateway. We will leave an empty request header, and we will not attach any tokens for authentication. As we can see in the figure 5, the error message is displayed: access was denied (HTTP error code: 401 Unauthorized). This suggests that filter that was configured on Zuul service intercepted this request and threw exception that led to error.

5 Conclusion

Thus, because of the software implementation of the conceived architecture, the set of isolated services was obtained, access to which is restricted to unauthorized users. By assigning roles to each of the users and issuing labels to the pages that are responsible for the necessary user permissions, the task of role – sharing the user access to services was solved.



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
AUTH-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-1UL7PV8:auth-service:9100
GALLERY-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-1UL7PV8:gallery-service:8300
IMAGE-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-1UL7PV8:image-service:8200
ZUUL-SERVER	n/a (1)	(1)	UP (1) - DESKTOP-1UL7PV8:zuul-server:8762

General Info

Name	Value
total-avail-memory	419mb
environment	test
num-of-cpus	4
current-memory-usage	156mb (37%)
server-uptime	00:26
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/,
available-replicas	

Fig. 5. List of registered service.

Scientific novelty of the elaboration is also that applied microservices are relatively new direction in programming engineering and has enough large number of advantages, one of the main ones is modularity, which was used by authors in the proposed software application. Such models can be developed in completely different programming languages and can be replaced easily at any time.

Comparison with analogues in this case is quite difficult to perform, since the authors did not meet similar systems developed using microservice architecture. Many foreign systems that have such functionality are paid and commercial, and it is quite problematic for authors to get access to them for research of their program code and possible functionality.

Principal difference between the proposed development and other software applications that provide protection against unauthorized access to confidential information is precisely the application of microservice architecture principle.

Microservices provide great opportunities for scalability of projects and allow developing individual projects.

The resulting architecture will be resistant to increasing loads on this system over time. Horizontal scaling is done by simply adding duplicate service to the overall application structure, which has an increased load. With the automatic registration of this service, the load balancing system will automatically reduce the load.

The implemented microservice architecture allows users to protect reliably against unauthorized access the software tool that is distributed across several computers, and

allows them to issue or restrict rights to individual information objects using discretionary or credential model.

References

1. Babash A V, Baranova E K, Larin D A 2015 *Information Security History of Information Security in Russia* (M: KDU, Pub Book House MGU) p 736
2. Bondarev V V 2016 *Introduction to Information Security of Automated Systems* (M: MSTU NE Bauman) p 252
3. Craig W 2015 *Spring in action* (M: DMK Press) p 754
4. Rambo J, Blaha M 2007 *UML 20 Object–oriented modeling and development* (SPb: Peter) p 544 p
5. *Microservices (Microservices)* https://habrcom/en/post/249183/
6. Partyka T L, Popov I I 2012 *Information Security* (M: Forum) p 432
7. Brandón Á, Solé M, Huélamo A, et al 2020 *Journal of Systems and Software* **159** 110432
8. Baboi M, Iftene A, Gîfu D 2019 *Procedia Computer Science* **159** 1035–1044
9. Wang R, Imran M, Saleemc K 2020 *Journal of Network and Computer Applications* **152** 102510
10. Schäffer E, Mayr, F, Fuchs J, et al 2019 *Procedia CIRP* **86** 86 – 91
11. Di Francesco P, Lago P, and Malavolta I 2019 *Journal of Systems and Software* **150** 77 – 97
12. Götz B, Schel D, Bauer D, et al 2018 *Procedia CIRP* **67** 167–172