Designing Information System for Private Network using RBAC, FGAC and Micro service Architecture

Arjit Mishra, Surendra Gupta, Swarnim Soni

***Abstract:** Microservice architecture is used in developing enterprise-level applications with the intent to modularise deployment of the application, this happens by creating an application as a collection of var-ious smaller applications known as microservices. An Information system is one such application that is ever-growing and therefore needs an architectural solution that addresses this issue. While microservice architecture addresses this issue by giving low coupling among microservices, future scalability of the system, and convenience in developing, deploying, and integrating new microservices.For all it‘s benefits, microservice architecture complicates the consistent implementation of security policies in this distributed system. Current industry standards are to use protocols that delegate the process of authentication and authorization to a third-party server, e.g. OAuth. Delegating these processes to be handled by the third party is not suitable for some web applications that are deployed in a less resourceful environment, e.g. organization with high internet downtime or an organization with high traffic of non working personnel e.g. people giving exams in college or workshops being held. This paper aims to research proposed solutions, existing frameworks, and technologies to implement security policies in an Information system which can be suitable for the above two scenarios.For this, we use authentication, Role-based access control (RBAC) on every request, and Fine-grained access control (FGAC) on the implementation method level, to achieve greater access control and flex-ibility of adding new microservice without changing whole security policies. We have also proposed a pre-registration condition in our system, which allows only certain people, whose data is already present in the system, to register themselves with the application. We also discuss the scenario where using a protocol like OAuth is not suitable. The solution is based on creating a central single entry point for authentication and implementing an RBAC policy that will filter every request based on access roles that the requesting user has. We further use FGAC on method level in microservices to enforce n even finer restrictions on resources to be accessed based on requirements. This solution will be implemented as apart of the Department Information System (DIS) in the following two-step:*

***Keywords:** (DIS),FGAC,RBAC.*

Manuscript received on April 15, 2021.
Revised Manuscript received on April 20, 2021.
Manuscript published on April 30, 2021.

* Correspondence Author

Arjit Mishra*, Research Scholar, Department of Computer Engineering, Shri G. S. Institute of Technology & Science, Indore.

Surendra Gupta, Associate Professor, Department of Computer Engineering, Shri G. S. Institute of Technology & Science, Indore.

Swarnim Soni, Assistant Professor, Department of Computer Engineering, Shri G. S. Institute of Technology & Science, Indore.

I. INTRODUCTION

Micro services are the need of the hour for developing enterprise-level applications. Businesses want ever scaling applications that are easy to develop, test, integrate, and deploy while allowing millions of users to access from different frontends i.e. mobile, computer, systems, etc. Applications based on monolithic architecture are painful in incremental development and release environments like agile but are great for developing policies for entire applications like security policies. Since most of the development community is moving towards microservice architecture, the system, although logically a single entity, is becoming more and more distributed. As a result, maintaining a single global policy is more difficult and increases code duplication. To ensure scalability of web applications and integration of new microservices with current security protocols implemented we need a robust way that can help us with authentication, authorization, and access control.

Traditionally authenticated users are logged in central databases that other microservices can access. This process of storing all login information in a central database is not recommended because it has a single point of failure upon which the whole application's operation depends.

Another alternative is to use OAuth and delegate authentication and authorization of an individual to a third party like Google or Facebook. This option is also not acceptable as:

- An organization that has high network downtime or doesn't want to expose its application to an outside network, will suffer greatly as the application is dependent on the third party of authentication and authorization.
- In an organization with high traffic of non-working people like a university, where many seminars, workshops, and exams are held, one needs a pre-registration policy to restrict who can register for this application.

The main objective of this project work is to design and implement Role-Based Access Control(RBAC) and Fine-Grained Access Control (FGAC) in an Information System which is Based on Microservice architecture and intended for a Private network.

To address our needs and solve shortcomings in the existing system we propose the following solution:

- Introduce a pre-registration condition to control registration to the system.
- Design a central gateway for authentication, authorization, and access control (RBAC) and reroute the request. For this, we will be using spring security and Netflix's zuul.
- To achieve resource(API) level access control we introduce FGAC.

II. RELATED WORK AND TECHNOLOGIES

A. **Role-Based Access Control(RBAC*)*:** RBAC is a mechanism to provide access control to different users based on their assigned role(s) [7]. In a large enterprise, an individual can have access to have different modules/tasks based on his/her position within the organization or competency for the task. RBAC allows developers to allow authorities to play a different role(s) without hardcoding the restrictions. Modules then can be allowed access to, based on the assigned role. RBAC allows us to set privileges, separation of duties, access control, and abstraction in the system.

B. **Fine-Grained Access Control(FGAC):** FGAC implementation depends largely on the requirement. The basic idea of FGAC is to define access control policy at a granular level like attribute, methods, object, etc, to achieve higher access control. FGAC is extensively used in cloud and database to achieve granularity in access control. The policy can be anything from allowing access to a particular time window to analyzing tokens by a cryptographic function [9][10]. FGAC is being used in cloud computing for enforcing access control on the instance level and also in database security by restricting the type of query a user can execute.

C. **OAuth:** OAuth is a protocol that enables developers to delegate the process of authentication and authorization to a third party [11]. In our case, due to unreliable network access, we cannot use OAuth as it will increase the downtime of our application. It also limits the organization's control over who can and cannot register to our system.

D. **Spring Security:** It's a powerful and highly customizable framework that allows developers to integrate authentication and access-control functionalities in the application [12]. A web application is usually developed with the use of a comprehensive framework and common business problems are implemented out of the box, just needing configuration from the user end [13].

E. **JSON Web Tokens:** It's a form of JSON strings used to perform authentication and information exchange in the system [14]. JWT can be signed by the issuing system and later can be verified while availing services from the system. Usually, a system with many microservices has an implementation for processing these JWT tokens and therefore verify that this user is authenticated or not [15].

JWT string consists of three parts which are separated by two dots, these three parts are:

i) Header*:* It typically consists of two parts:

- Which type of token it's going to be, which in this case, it's JWT.
- Signing algorithms like SHA256, RSA, etc.

```
{
"alg”:”HS256”,
"type”:”JWT”
}
```

ii) *Payload:* it's the second part of JWT and contains claims. Claims are information about an object or special data. Claims can be classified into three types:

- Registered claims: a group of predefined, recommended yet not compulsory claims. It provides metadata like **iss**, **exp**, **sub**, **aud,** etc.
- Public claims*:* those who will be using this JWT can define these claims as per their needs eg username, user type, authorities, etc. These are specific to the purpose for which JWT is being used.
- Private claims*:* these types of claims are special case assertions created as per custom need of application and can be shared among parties who agreed to use JWT e.g. privileges, access control, etc.

  Following is a demo payload:

```
{
"sub”:”1234567890”,
"name”:”Arjit Mishra”,
"Admin”: true
}
```

iii) Signature: Signature ensures that message is not altered during transmission of JWT over the network. To create a JWT we use base64 encoder to encode header and payload and then sign it with specified algorithm and secret key

```
HMACSHA256(base64UrlEncode(header) + "."
+ base64UrlEncode(payload),my-key)
```

Final JWT generated form above configuration is:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJz
dWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkFyaml0I
E1pc2hyYSIsImlhdCI6MTUxNjIzOTAyMn0.eUqJs9
8gELCCwoy-u1E7k0W8eJKw6XRruElukrJT2Ow
```

F. **Netflix Zuul:** Zuul is the first checkpoint for all the requests from different frontend to backend services. Its purpose is to enable dynamic routing for incoming requests, security, and monitoring. Since we have a single entry gateway for all other microservices, we need zuul only for dynamic routing to redirect requests to microservices in our Information system. It can be configured in the gateway along with spring security so we can redirect the requests after authentication.

III. DESIGN AND ANALYSIS

We consider four microservices namely Administration Microservice, Academics Microservice, User Microservice, and Infrastructure Microservice with their respective databases. Although there was no freezing of requirements, the microservices were still discovered and fixed. The collected requirements were not fixed, and even after implementation began, they continued to change. Later, any minor change in requirements or responsibilities was handled easily but a major change in it forced to redesign microservices and therefore should be avoided. Overall components in the Department Information System(DIS) are shown in a block diagram below:

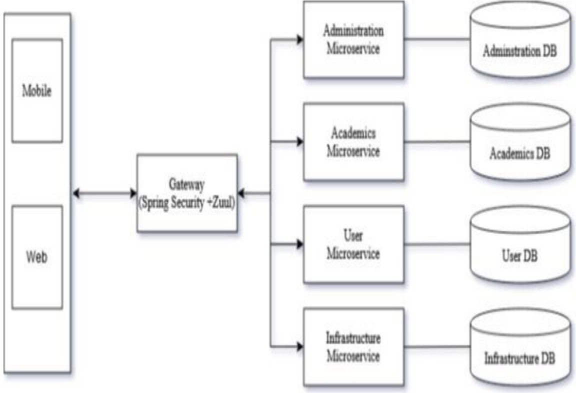


Fig 1: System Block Diagram

Considering the above block diagram, we have two scenarios, first is designing pre-registration flow, Fig 2(a), and the second is designing resource request flow based on RBAC and FGAC, Fig 2(b).

Pre-registration conditions are fulfilled by checking for user data in the database prior to registration, a fraudulent user's data will not be available in the database. If we have used Oauth then anyone with an account of the authenticating service will be able to log in, also the system will be out of service if the internet infrastructure is not optimum and frequent breakdowns, or doesn't exist at all.

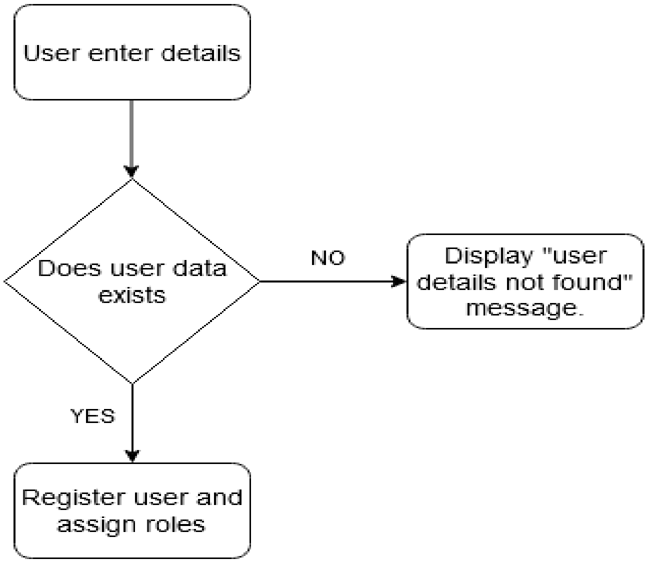

User enter details
Does user data exists
NO
Display "user details not found" message.
YES
Register user and assign roles

Fig 2(a): Flow Diagram: Registration

Resource request flow is designed by implementing RBAC using spring security. To implement FGAC we hard code specific conditions particular to that resource, as every resource will have unique restrictions.

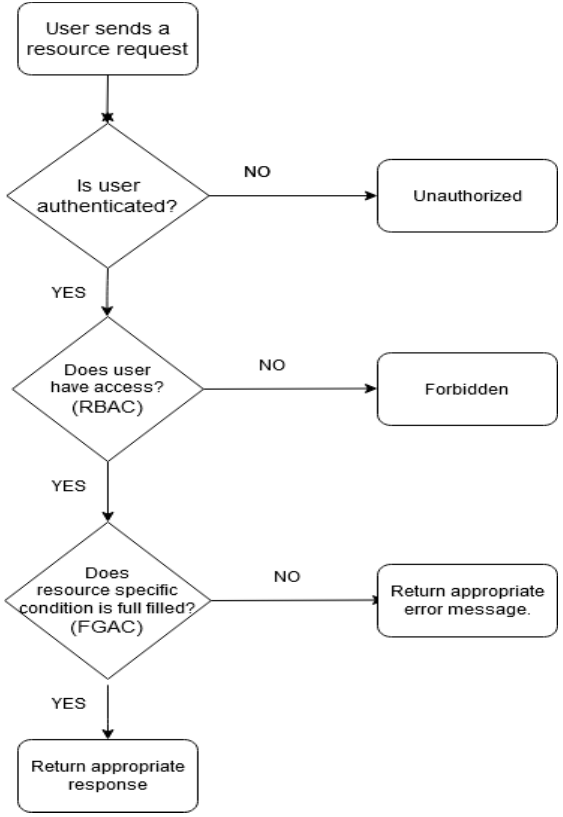

User sends a resource request
Is user authenticated?
NO
Unauthorized
YES
Does user have access? (RBAC)
NO
Forbidden
YES
Does resource specific condition is full filled? (FGAC)
NO
Return appropriate error message.
YES
Return appropriate response

Fig 2(b): Flow Diagram: Resource request

When someone attempts to register themselves with the system, their credentials are first checked by the systems. These credentials are uploaded when the user joins the institute. If the credential is found then only that person is registered in the system else an error message is sent.

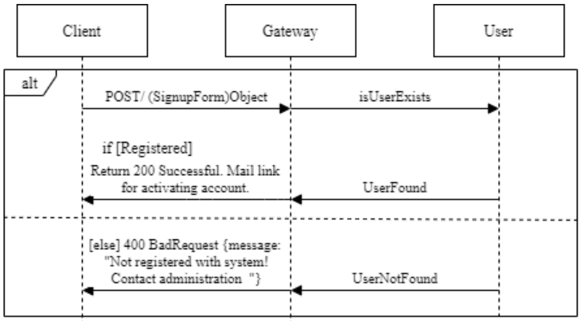

Client
Gateway
User
alt
POST/ (SignupForm)Object
isUserExists
if [Registered]
Return 200 Successful. Mail link for activating account.
UserFound
[else] 400 BadRequest {message: "Not registered with system! Contact administration "}
UserNotFound

Fig 3 (a): Sequence diagram for registration.

When a client logs in with a username and password, after successful authentication a JSON payload along with roles/authorities and JWT is returned by the gateway. After receiving the jwt token, we can pass it into the authorization header in the HTTP request.

This JWT can then be used to authenticate yourself during future requests. When a request comes, role-based access control (**RBAC)** can be achieved by filtering URLs based on roles/authorities in JSON payload in incoming requests. This is achieved by the spring security framework which is configured by users as per their requirement.Below is a sequence diagram showing the authentication process:

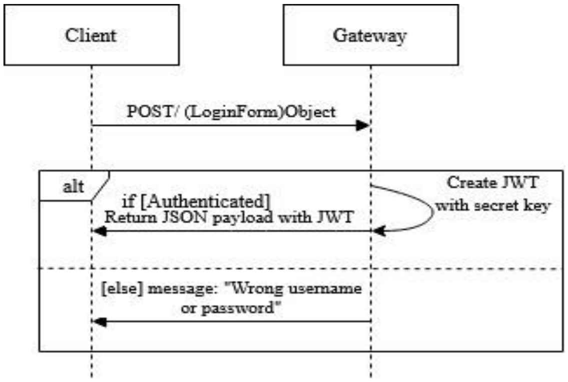


Fig 3 (b): Sequence diagram for authentication.

The authorized requests are then rerouted to respective microservices and an appropriate response is returned. Following is the zuul configuration for API gateway which is configured using Netflix's Zuul, and is to be mentioned in the application properties file. All the microservices are configured to accept requests only from the gateway, so when a request originates (redirected) from the gateway, access is granted by these microservices. Furthermore, we implemented fine-grained access control (**FGAC)** in method, say `/user/addFaculty,` which can only be accessed during a certain time period, to achieve granularity in access control. Now there can be any condition that can be used to achieve granular control over a resource, the one mentioned above is just for reference.Below sequence diagram demonstrates service request from other microservice:

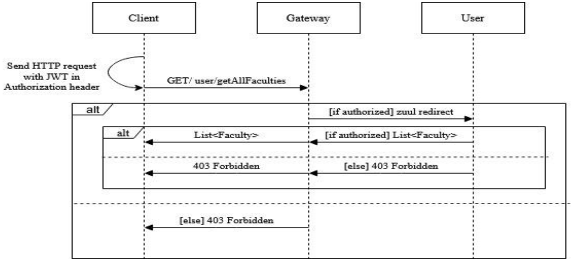
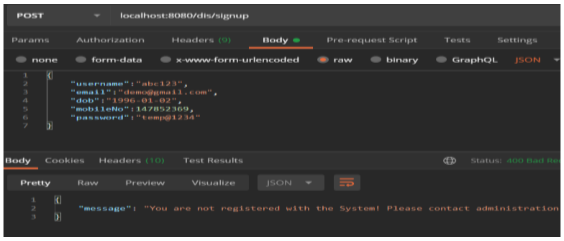


Fig 4: Sequence diagram for the service request.

IV. RESULT AND DISCUSSION

All the tests shown below are a part of the **Department Information System** being developed for the Department of Computer Engineering, Shri Govindram Seksaria Institute of Technology and Science, Indore.

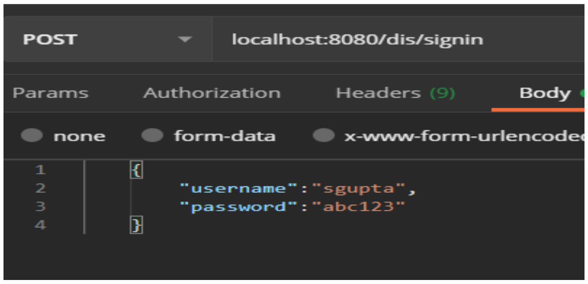


```
{
    "username":"abc123",
    "email":"demo@gmail.com",
    "dob":"1996-01-02",
    "mobileNo":147852369,
    "password":"temp@1234"
}
```

```
{
    "message": "You are not registered with the System! Please contact administration
}
```

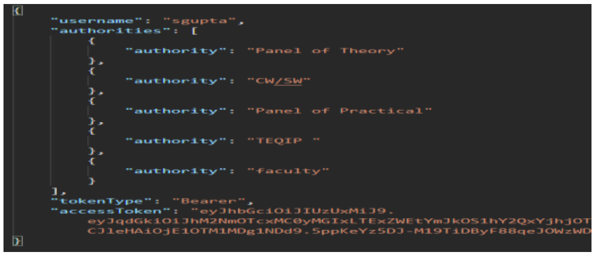
Fig 5: Failed registration.

If a user whose data is not present in the system beforehand then their registration will fail as shown in Fig 4.We can see the authentication process in Fig 6(a) and Fig 6(b). A POST request is sent and after validation, a JSON response along with an access token is returned from the server. This access token will be sent in an authorization header to avail of further services. Any change in this token will result in an Unauthorized(401) response from the server.



```
{
    "username":"sgupta",
    "password":"abc123"
}
```

Fig 6(a): LoginForm object

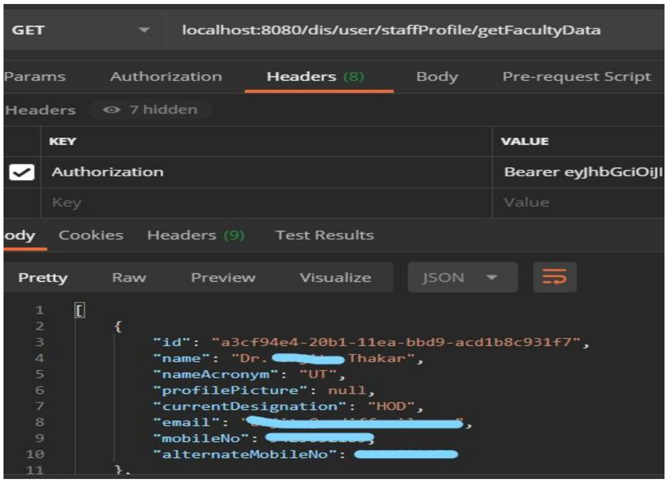


```
{
    "username": "sgupta",
    "authorities": [
        {
            "authority": "Panel of Theory"
        },
        {
            "authority": "CW/SW"
        },
        {
            "authority": "Panel of Practical"
        },
        {
            "authority": "TEQIP "
        },
        {
            "authority": "faculty"
        }
    ],
    "tokenType": "Bearer",
    "accessToken": "eyJhbGciOiJIUzUxMiJ9.
        eyJqdGkiOiJhM2NmOTcxMC0yMGIxLTExZWEtYmJkOS1hY2QxYjhjOT
        CJleHAiOjE1OTM1MDg1NDd9.5ppKeYz5DJ-M19TiDByF88qeJOWzWD
}
```

Fig 6(b): Login Response

Users can access the APIs available to them as per security policies and then the appropriate response is received (Fig 7(a)), if the authorization header is empty then the request is unauthorized as shown below (Fig 7(b)). In the example shown in Fig 7(a) the resource /dis/user/staffProfile/getFacultyData is available to all authenticated users and therefore data is returned when the get request is made.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication
© Copyright: All Rights Reserved

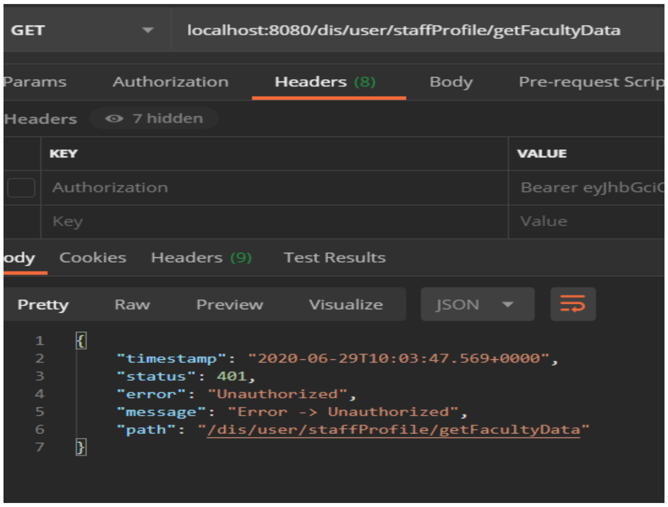


GET localhost:8080/dis/user/staffProfile/getFacultyData

	KEY	VALUE
☑	Authorization	Bearer eyJhbGciOiJI
	Key	Value

```
[
    {
        "id": "a3cf94e4-20b1-11ea-bbd9-acd1b8c931f7",
        "name": "Dr. [redacted] Thakar",
        "nameAcronym": "UT",
        "profilePicture": null,
        "currentDesignation": "HOD",
        "email": [redacted],
        "mobileNo": [redacted],
        "alternateMobileNo": [redacted]
    },
```

Fig 7(a): Request with authorization header.



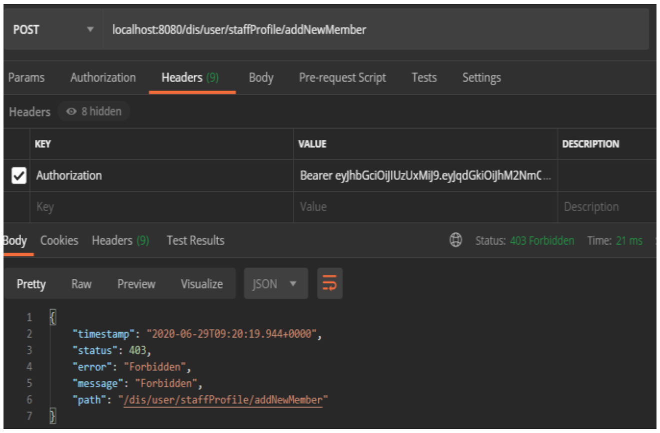
GET localhost:8080/dis/user/staffProfile/getFacultyData

	KEY	VALUE
□	Authorization	Bearer eyJhbGciC
	Key	Value

```
{
    "timestamp": "2020-06-29T10:03:47.569+0000",
    "status": 401,
    "error": "Unauthorized",
    "message": "Error -> Unauthorized",
    "path": "/dis/user/staffProfile/getFacultyData"
}
```

Fig 7(b): Request with empty authorization header.

If a user is not authorized to access any resource as per the policy, then the server responds with Forbidden(403). In the below figure (Fig 8) we see that the server forbids access to the resource at `/dis/user/staffProfile/addNewMember` based on role (RBAC).



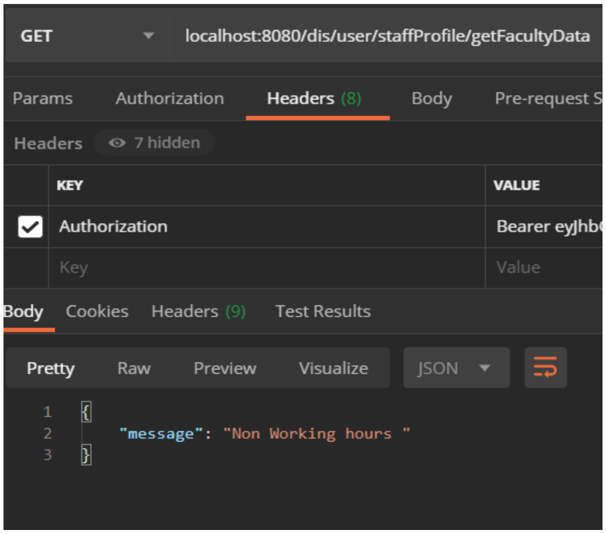
POST localhost:8080/dis/user/staffProfile/addNewMember

	KEY	VALUE	DESCRIPTION
☑	Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJqdGkiOiJhM2NmC...	
	Key	Value	Description

Status: 403 Forbidden Time: 21 ms

```
{
    "timestamp": "2020-06-29T09:20:19.944+0000",
    "status": 403,
    "error": "Forbidden",
    "message": "Forbidden",
    "path": "/dis/user/staffProfile/addNewMember"
}
```

Fig 8: Forbidding access to a resource based on the role (RBAC)



GET localhost:8080/dis/user/staffProfile/getFacultyData

	KEY	VALUE
☑	Authorization	Bearer eyJhb
	Key	Value

```
{
    "message": "Non Working hours "
}
```

Fig 9: Forbidding access based on an attribute's condition (FGAC)

All users have access to `/getFacultydata` resource, but let's say we implement a condition on an attribute that denies access outside working hours, then that access can be revoked. Figure (Fig 9) shows the server forbids access to resources based on security policy on a time attribute that it should be within working hours.

V. CONCLUSION

Spring security provides a way to configure a central gateway for authentication, and role-based access control in an ever-growing and complex web application. A new microservice can be easily integrated with the existing system by doing only minimal configuration as the changes are to be done only in one place. If there are special requirements for authorization then these constraints can be coded in method level, this is called fine-grained access control. We also successfully implemented pre-registration conditions so that only intended people can register themselves with the system.

This approach of authentication and authorization also removes the problem with central logging systems as there is no need to maintain a log of the authenticated user, which will result in a single point of failure. By using JWT, the party who is requesting a service is going to provide a JWT token, these parties can simply be verified by the gateway and then redirected to the appropriate microservice. If a token is expired or changed in any way these requests will not be authorized. This helps in maintaining a stateless session between client and server.

REFERENCES

1. FU Yue, "A Study of Student Information Management Software", Chongqing Institute of Technology, 2016 IEEE.

2. Software architecture, [Online] https://www.tutorialspoint.com/software_architecture_design/index.htm Accessed on : Nov 29, 2020
3. L. Bass, P. Clements, and R. Kazman. “Software Architecture in Practice”, 2nd ed.Reading, MA: Addison Wesley, 2003. [E-book] Available: Safari e-book.
4. N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin,and L. Safina. “Microservices: yesterday, today, and tomorrow”, arXiv preprintarXiv:1606.04036, 2016.
5. M. Villamizar et al.,“Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” 2015 10thComputing Colombian Conference (10CCC), Bogota, 2015, pp. 583-590. doi:10.1109/ColumbianCC.2015.7333476.
6. P. Siriwardena, “Advanced API Security: OAuth 2.0 and Beyond”, Second Edition, Apress, Berkeley, CA, 2020 Springer.
7. Tetiana Yarygina, Anya Helene Bagge, “Overcoming Security Challenges inMicroservice Architectures”, Department of Informatics, University of Bergen, Norway,2018 IEEE Symposium on Service-Oriented System Engineering.
8. Ravi S. Sandhu, Edward J. Cope, Hal L. Feinstein, Charles E. Youman, “Role-Based Access Control Models, SETA Corporation, 1996 IEEE.
9. Antonio Nehme, Vitor Jesus, Khaled Mahbub, and Ali Abdallah, “Fine-Grained Access Control for Microservices”, School of Computing and Digital Technologies, Birmingham City University, Birmingham, UK, Springer Nature Switzerland AG 2019.
10. S. Newman, “Building Microservices: Designing Fine-Grained Systems", O'Reilly Media (2015), ISBN: 978-1491950357.
11. OAuth 2.0, [Online] https://oauth.net/. Accessed on : Dec 1, 2020
12. Spring projects, “Spring Security”, [Online] https://spring.io/projects/spring-security#overview. Accessed on : Dec 1, 2020
13. Tetiana Yarygina, Anya Helene Bagge, “Overcoming Security Challenges in Microservice Architectures”, Department of Informatics, University of Bergen, Norway, 2018 IEEE Symposium on Service-Oriented System Engineering.
14. jwt.io, “Introduction to JSON Web Tokens”, [Online] https://jwt.io/introduction/. Accessed on: Accessed on : Dec 2, 2020
15. RCBJ-ADMIN, "JWT Use Cases," 7 2017. [Online]. Available: http://rcbj.net/blog01/2017/07/14/jwt-use-cases/. Accessed on :Dec 2, 2020
16. I. I, P. M. R. Anand and V. Bhaskar, "Encrypted Token-based Authentication with Adapted SAML Technology for Cloud Web Services," Journal of Network and Computer Applications 99, 2017.
17. Xiuyu He, Xudong Yang,” Authentication and Authorization of End User in Microservice Architecture”, Department of Computer Science and Technology, Beijing University of Posts of Telecommunications, Beijing, China, IOP Conf. Series: Journal of Physics, CTCE2017

AUTHOR’S PROFILE

Arjit Mishra, Research Scholar, Department of Computer Engineering, Shri G. S. Institute Of Technology & Science, Indore. Completed Bachelor of Engineering in Information Technology (Hons) from the Department of Information Technology, Technocrats Institute of Technology, Excellence, Bhopal in 2016 . arjitm786@gmail.com

Surendra Gupta, Associate Professor, Department of Computer Engineering, Shri G. S. Institute Of Technology & Science, Indore. Pursuing Ph.D. from DAVV, Indore. Completed Bachelor of Engineering in Computer Engineering from Samrat Ashok Technological Institute, Vidisha 1997. Completed Masters of Engineering in Computer Engineering from Shri G. S. Institute of Technology & Science, Indore 2001. sgupta@sgsits.ac.in

Swarnim Soni, Assistant Professor, Department of Computer Engineering, Shri G. S. Institute Of Technology & Science, Indore. Completed B.Tech in Computer Science and Engineering from the Institute of Engineering, DAVV, Indore 2009. Completed M.Tech in Computer Science and Engineering from the Indian Institute of Technology, Madras 2013. swarnimsoni@gmail.com