Security (Authentication and Authorization) Mechanisms for Micro-Service Based Applications

BITS SEZG628T: Dissertation

by

ANKIT JAIN

2020MT93155

Dissertation work carried out at

PUBLICIS SAPIENT, NOIDA

**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN)**

APRIL, 2022

Security (Authentication and Authorization) Mechanisms for Micro-Service Based Applications

BITS SEZG628T: Dissertation

by

ANKIT JAIN
2020MT93155

Dissertation work carried out at

Publicis Sapient, Noida

Submitted in partial fulfilment of **M.Tech. Software Engineering** Degree Programme

Under the Supervision of

Kavita Missan

Sopra Steria, Noida

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI (RAJASTHAN)

APRIL, 2022

CERTIFICATE

This is to certify that the Dissertation entitled **Security (Authentication and Authorization) Mechanisms for Micro-Service Based Applications** and submitted by Ankit Jain having ID no. **2020MT93155** for the partial fulfilment of the requirements of M.Tech. Software Engineering Degree Programme of BITS, embodies the bonafide work done by him, under my supervision.

Kavita

Signature of Supervisor

Kavita Missan
Project Lead
Sopra Steria Ltd., Noida

Place: Noida

Date: 24 April, 2022

ACKNOWLEDGEMENT

The completion of this project report could not have been accomplished without the kind support and help of number of individuals. I would like to extend my sincere thanks to all of them.

First and foremost, I am thankful to BITS, PILANI (WILP) for giving me an opportunity to do this project study. I would like to acknowledge the supervision and professional assistance that Kavita Missan, Project Lead, Sopra Steria, Noida, provided me in completion of this project report. Her insights and expertise helped me a lot in understanding various concepts and issues related to topic of the report.

In continuation, I would like to thank Garima Jain, Technical Analyst, Coforge, Noida, for all the critical comments and professional assistance during progress report that helped me a lot during the project study.

Last but not least, I am grateful to my colleagues who guided and helped me at various stages of project study.

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

April, 2022

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
(RAJASTHAN)
WILP Division

Organization	**: PUBLICIS SAPIENT**
Location	**: NOIDA**
Duration	**: 5 MONTHS**
Date of Start	**: 15TH JANUARY, 2022**
Date of Submission	**: 25TH APRIL, 2022**
Title of the Project	**: Security (Authentication and Authorization) Mechanisms for Micro- Service Based Applications**
Name of the student	**: ANKIT JAIN**
ID of student	**: 2020MT93155**
Name of Supervisor	**: Kavita Missan**
Designation of Supervisor	**: Project Lead**
Name of Additional Examiner	**: Garima Jain**
Designation of Additional Examiner	**: Technical Analyst**
Name of the Faculty Mentor	**: Rajyalakshmi**

Key Words: Security, Microservices, JWT, OAuth2, Springboot

Project Areas: Software Engineering, Software Security, Microservice Security

[illegible]

Kavita

Signature of Student
Date: 24th April, 2022

Signature of your Supervisor
Date: 24th April, 2022

ABSTRACT

Application security is the process of making apps more secure by finding, fixing, and enhancing the security of apps. Much of this happens during the development phase, but it includes tools and methods to protect apps once they are deployed. This is becoming more important as hackers increasingly target applications with their attacks.

Application security is getting a lot of attention. Hundreds of tools are available to secure various elements of your applications portfolio, from locking down coding changes to assessing inadvertent coding threats, evaluating encryption options and auditing permissions and access rights. There are specialized tools for mobile apps, for network-based apps, and for firewalls designed especially for web applications.

Web application security is a branch of information security that deals specifically with security of websites, web applications and web services. At a high level, web application security draws on the principles of application security but applies them specifically to internet and web systems.

Web Application Security Tools are specialized tools for working with HTTP traffic, e.g., Web application firewalls.

Security threats:
The Open Web Application Security Project (OWASP) provides free and open resources. It is led by a non-profit called The OWASP Foundation. The OWASP Top 10 - 2017 is the published result of recent research based on comprehensive data compiled from over 40 partner organizations. From this data, approximately 2.3 million vulnerabilities were discovered across over 50,000 applications. According to the OWASP Top 10 - 2021, the ten most critical web application security risks include:

- Broken access control
- Cryptographic Failures
- Injection
- Insecure Design
- Security Misconfiguration
- Vulnerable and Outdated Components
- Identification and Authentication Failures
- Software and Data Integrity Failures
- Security Logging and Monitoring Failures
- Server-Side Request Forgery (SSRF)

[illegible]

Signature of the Student

Name: Ankit Jain

Date: 10th March, 2022

Place: Delhi

Kavita

Signature of the Supervisor

Name: Kavita Missan

Date: 10th March, 2022

Place: Delhi

Table of Contents

1. Advantages of Microservice Based Applications

Working with small components creates room to scale the service in separate parts.
Each microservice has its own autonomy & provides flexibility on the technology that will be used.
Productivity & velocity can increase by allowing different development teams to work on various components simultaneously without impacting each other.
Development teams are focused and organized around the business functionality.
Developers have the freedom to work more independently and autonomously without having a dependency on other teams.

2. WHAT ABOUT SECURITY WHEN MOVING TO MICROSERVICE BASED APPLICATION

A monolith needs to secure only itself, and that's manageable. Microservices have a bigger attack surface, which means a larger number of services comes with more significant risks. Each one needs to take care of all the weaknesses that might be exposed.

In a monolithic architecture, components invoke one another via a method call. In contrast, microservices may expose internal API's (synchronous calls) to communicate with each other. This requires more effort and attention to secure it.

In a monolith, all internal components share the same user session context. In a microservices architecture, nothing is shared between them, so sharing user context is harder and must be explicitly handled from one microservice to another.

3. AUTHENTICATIONS FLOWS

Authentication is the process of identifying users that request access to an application, enterprise, network, or device. The authentication flows are not new but it makes sense to reiterate them in the context of microservices.

- **Login**: This is the most critical and sensitive user flow in any system or application. An end-user is authenticated to a system based on this flow.

Typically the user identity is based on username and password but many other variations are in practice these days, like mobile-based OTPs, biometric & facial recognition, 3rd party authentication providers, multi-factor authentication, etc. A typical microservices setup will use *Identity Server/STS/Open-Id* to authenticate the user.

- **Subsequent Requests & User Session:** We need to ensure all the requests to our application are authenticated. We cannot ask the user for his password every time. In the monoliths, we used to maintain user state through session management but with the microservices, it's not possible. The typical characteristics of microservices, being independently deployed, restful, and stateless, have pushed the enterprises to adopt token-based access mechanisms such as *OAuth 2.0*.

- **Service to Service Communication:** This was not the case with monolith applications. Internal modules were bundled and deployed together so there was no need for additional security. With microservices, internal modules get deployed as independent services. Each of these services needs to authenticate itself to the other service(s).

- **Password Reset:** This flow is as crucial as the "Login" flow in terms of security risks. Similar to the "Login" flow, it also uses multiple variations including OTPs, email verification, security questions, biometric, multi-factor, etc. Implementation of it is mostly unchanged with microservices.

- **Logout:** In the typical monolith, once you click "logout" the user is supposed to be logged out from all its services, as its session was invalidated. This is not so straightforward with microservices, as the tokens are used to access them. Depending upon the configuration, they can be valid for a long time

4. THE DIFFERENCE BETWEEN AUTHENTICATION AND AUTHORIZATION

Both terms will come to mind when talking about securing applications. And yet, there might be people that confuse the meaning of these terms.

In the **authentication process**, the identity of the user is checked to provide access to the system. This process verifies 'who you are?' so the user needs to supply login details to authenticate.

Authorization is the process of verifying if the authenticated user is authorized to access specific information or be allowed to execute a certain operation. This process determines which permissions the user has.

When talking about securing applications, the terms authentication and authorization will pop up. While the terms are used interchangeably, they represent different purposes in the spectrum of securing applications.

5. AUTHENTICATION STRATEGY IN A MICROSERVICE ARCHITECTURE

While moving from monolith to microservices architecture, it is important to manage security and access control by understanding how to implement authentication and authorization in the microservices world.
There are several approaches:

5.1 Authentication & Authorization on each service

Each microservice needs to implement its own independent security and enforce it on each entry-point. This approach gives the microservice team autonomy to decide how to implement their security solution. However, there are several downsides about this approach:

- The security logic needs to be implemented repeatedly in each microservice. This causes code duplication between the services.
- It distracts the development team from focusing on their domain main service. Each microservice depends on user authentication data, which it

doesn't own. It’s hard to maintain and monitor.

- Authentication should be a global solution and handle as a cross-cutting concern.
- One option to refine this solution would be to use a shared authentication library loaded on each microservice. This will prevent code duplication, and the development team will focus only on their business domain. However, there are still downsides that this refinement can’t solve.

5.2 Global Authentication & Authorization Service

In this strategy, a dedicated microservice will handle authentication and authorization concerns. Each business service must authenticate the request before processing it by down streaming it to the authentication service. However, there are several downsides about this approach:

- The authorization check is a business concern. What specific user roles are allowed to do on the service is governed by business rules. Therefore, the authorization concern should not be handled in the global authentication service.
- This strategy increases the latency of processing requests

5.3 Global Authentication (API Gateway) and authorization per service

When moving to a microservice architecture, one of the questions that need to be answered is how an application’s clients communicate with the microservices. One approach would be to use direct access between client and microservice. This approach suffers from a strong coupling between clients and microservices.

The API gateway is a single endpoint entry for all requests. It provides flexibility by acting as a central interface for clients using these microservices. Instead of having access to multiple services, a client sends a request to the API gateway responsible for routing it to the downstream service.

Because the API gateway is a single endpoint entry, it is an excellent candidate to enforce authentication concerns. It reduces the latency (call to Authentication service) and ensures the authentication process is consistent across the application. After successful authentication, the security component will enrich the request with the user/security context (identity details on the login user) and route the request to a downstream service that enforces the authorization check.

6. Microservices authentication and authorization technical solutions

6.1 Distributed Session Management

In order to make full use of benefits of the microservice architecture and to achieve the scalability and resiliency of the microservices, the microservices are preferably to be stateless.

This solution can be applied through different ways like:

- **Sticky session**
  Which ensures that all requests from a specific user will be sent to the same server who handled the first request corresponding to that user, thus ensuring that session data is always correct for a certain user. However, this solution depends on the load balancer, and it can only meet the horizontally expanded cluster scenario, but when the load balancer is forced suddenly for any reason to shift users to a different server, all of the user's session data will be lost.
- **Session replication**
  Means that each instance saves all session data, and synchronizes through the network. Synchronizing session data causes network bandwidth overhead. As long as the session data changes, the data needs to be synchronized to all other machines. The more instances, the more network bandwidth the synchronization brings.
- **Centralized session storage**
  Means that when a user accesses a microservice, user data can be obtained from shared session storage, ensuring that all microservices can read the same session data. In some scenarios, this scheme is very good, and the user login status is opaque. It is also a highly available and scalable solution. But the disadvantage of this solution is that shared session storage requires a certain protection mechanism and therefore needs to be accessed through a secure way.

6.2 Client Token

The traditional way is to use a session on the server side to save the user state. Because the server is stateful, it has an impact on the horizontal expansion of the server. It is recommended to use Token to record user login status in the microservice architecture.

The main difference between Token and Session is where the storage is different. Sessions are stored centrally in the server; Tokens are held by the user themselves and are typically stored in the browser in the form of cookies. The Token holds the user's identity information, and each time the request is sent to the server, the server can therefore determine the identity of the visitor and determine whether it has access to the requested resource.

The Token is used to indicate the user's identity. Therefore, the content of the Token needs to be encrypted to avoid falsification by the requester or the third party. JWT (Json Web Token) is an open standard (RFC 7519) that defines the Token format, defines the Token content, encrypts it, and provides lib for various languages.

6.3 Single sign-on

The idea of single sign-on is simple, that is, users only need to log in to the application once, then they can access all the microservices in the application. This solution means that each user-oriented service must interact with the authentication service. This can result in a lot of very trivial network traffic, repeated work, and it may cause single point of failure. When there are dozens of micro-applications, the drawbacks of this solution will become more apparent.

6.4 Client Token with API Gateway

The authentication process of the user is similar to the basic process of token authentication. The difference is that the API Gateway is added as the entrance of the external request. This scenario means that all requests go through the API gateway, effectively hiding the microservices. On request, the API gateway translates the original user token into an opaque token that only itself can resolve.

- Client Token with API Gateway solution

  In this case, logging off is not a problem because the API gateway can revoke the user’s token when it logs out and also it adds an extra protection to Auth Token from being decrypted by hiding it from the client.

6.5 Third-party application access

6.5.1 API Token

The third party uses an application-issued API Token to access the application’s data. The Token is generated by the user in the application and provided for use by third-party applications. In this case, generally only third-party applications are allowed to access the user’s own data of the Token, but not other users’ sensitive private data.

The advantage of using the API Token instead of using the username/password directly to access the API is to reduce the risk of exposing the user’s password, and to reclaim the token’s permissions at any time without having to change the password.

6.5.2 OAuth

Some third-party applications need to access data from different users, or integrate data from multiple users. You may consider using OAuth. With OAuth, when a third-party application accesses a service, the application prompts the user to authorize a third- party application to use the corresponding access authority and generates a token for access according to the user’s permissions.

6.5.3 OAuth authentication process

In the above example, the resource server and the authorization server are both Github, the client program is GitBook or Travis CI, and the user is a direct user of the client program.

Someone may wonder why an Authorization Code is used to request Access Token, rather than returning the Access Token to the client directly from the authorization server. The reason why OAuth is designed in this way is to pass through the user agent (browser) during the process of redirecting to the client’s Callback URL. If the Access Token is passed directly, there is a risk of being stolen.

By using the authorization code, the client directly interacts with the authorization server when applying for the access token, and the authorization server also authorize the client when processing the client’s token request, so it’s prevented others from forging the client's identity to use the authentication code.

When implementing user authentication of the microservice itself, OAuth may also be used to delegate user authentication of the microservice to a third-party authentication service provider.

The purpose of using OAuth for user authorization of third-party application access and microservices is different. The former is to authorize private data access rights of users in microservices to third-party applications. Microservices are authorization and resource servers in the OAuth architecture. The purpose of the latter is to integrate and utilize the OAuth authentication service provided by a well-known authentication provider, which simplifies the cumbersome registration operation, in this case the microservice act the role of the client in the OAuth architecture.

6.6 Mutual Authentication

In addition to vertical traffic from users and third parties, there is a large amount of horizontal traffic between microservices. This traffic may be in the same local area network or across different data centers. Traffic between these microservices exists by third parties. The danger of sniffing and attacking also requires security controls.

Through mutual SSL, mutual authentication between microservices can be achieved, and data transmission between microservices can be encrypted through TLS. A certificate needs to be generated for each microservice, and the microservices are authenticated with each other's certificates. In the microservice operating environment, there may be a large number of microservice instances, and the microservice instances often change dynamically, such as adding service instances as the level expands. In this case, creating and distributing certificates for each service becomes very difficult. We can create a private certificate center (Internal PKI/CA) to provide certificate management for various microservices such as issuing, revoking, and updating.

7. JSON Web Token (JWT)

A JWT is an open standard (RFC-7519) that defines a mechanism for securely transmitting information between two parties. The JWT token is a signed JSON object that contains a list of claims which allow the receiver to validate the sender's identity.

The purpose of using the JWT token is for a stateless authentication mechanism. Stateless authentication stores the user session on the client-side.

7.1 JWT Structure

The JSON Web token is composed of three parts separated by periods.

The header contains the algorithm used for signing.

The payload is the session data that also refers to ‘claims’. There are two types of claims:

- **Reserved Claims**

  The JWT specifications define reserved claims that are recommended to use while generating the JWT token.

- **Custom claims**

  The signature is the most critical part. The signature is calculated by encoding the header and the payload using Base64 encoded. Then the encode64 is signed using a secret key and cryptographic algorithms specified in the header section. The signature is used to verify the token has not changed or modified.

7.2 JWT Best Practices and Pitfalls

- Always use the HTTPS protocol to offer better protection. This way, all data sent between the client browser and a server are encrypted.
- Keep the token size as small as possible. The JWT can be either a signed token by using JSON Web Signature (JWS) or a more secure level of protection by using JSON Web Encryption (JWE). Either way, as a rule of thumb, the token should not contain sensitive data.
- Several attacks rely on ambiguity in the API of certain JWT libraries. Make sure the JWT library is protected against it by validating the algorithm name explicitly.
- Make sure to use a strong secret key as long as the length of the hash algorithm.
- Set the JWT token to a short period to reduce the probability of it being used maliciously.

- The JWT is automatically valid until it expires. If an attacker gets the token, the only way to "kill" the session is by a stateful solution that explicitly detects and rejects those tokens.
- Using JWT does not prevent CSRF attacks. A Cross-site request forgery, CSRF, is a web security vulnerability that allows the attacker to perform actions that they are not minted to perform. Use a synchronizer token pattern to prevent it.
- For additional JWT Best practices, read The JSON Web Token Current Best Practices
- Securing RESTful microservices involves not only the implementation of security mechanisms at the server side, but also web clients must be able to adjust its own requests to meet the security requirements configured by such microservice application.

7.3 Introduction to JWT, JWS and JWK

In general, JSON Web Tokens (JWT) consist on a security mechanism to transfer information between two parties by using JSON-format tokens that are self-signed. As such, a consumer of JWT can perform JWT signature verification, which is a cryptographic process to verify the authenticity of the information contained in the token, as well as validations of the information within the token payload.

JWT consists of three JSON format parts:

- **Javascript Object Signing and Encryption (JOSE) Header**
  This part of the token provides details about the encryption mechanisms that must be applied to verify the token signature to confirm the authenticity of the information (claims) contained in the token.

  When working with JWKS, JOSE header will specify the following attributes:

  - **encryption algorithm** – alg Declared with value "RS256" (asymmetric encryption algorithm, described at this specification)
  - **Key id –** kid This attribute contains the identifier for the public key that must be used to perform signature verification.

- **Token payload**
  It consists on a set of key-value attributes commonly referred to as "Claims", where

the attribute name and value are called “Claim Name” and “Claim value”. Claims are ultimately the lowest level of information contained in the JWT, and can be used by the resource owners to determine any kind of information about the user who owns the token (access level, identity, authorization, etc.)

- **Token signature -JWS**

  From the security standpoint, this is the most important part of a JWT, as it contains the token signature that must be used to perform the verification of the token. Token signature is the result of taking the token payload and apply RS256 encryption using the private key of the RSA key pair.

All three parts of the JWT are BASE64 encoded, which makes the JWT URL-encoded, this means it can be shared between two or more parts using HTTP.

8. PROJECT WORK

8.1 DESIGN AND ARCHITECTURE

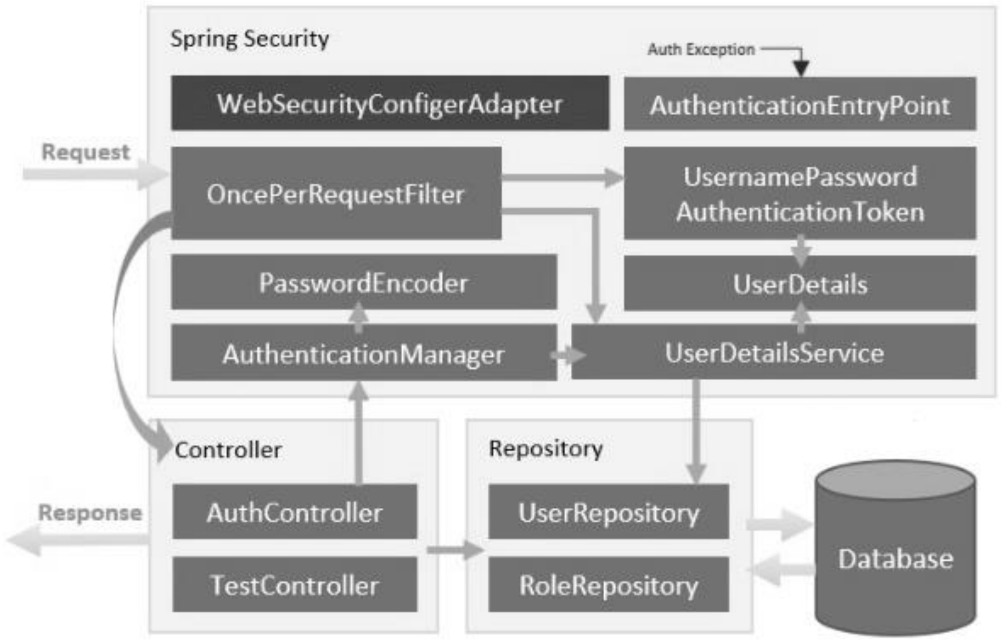


Figure 1: SPRING SECURITY ARCHITECTURE

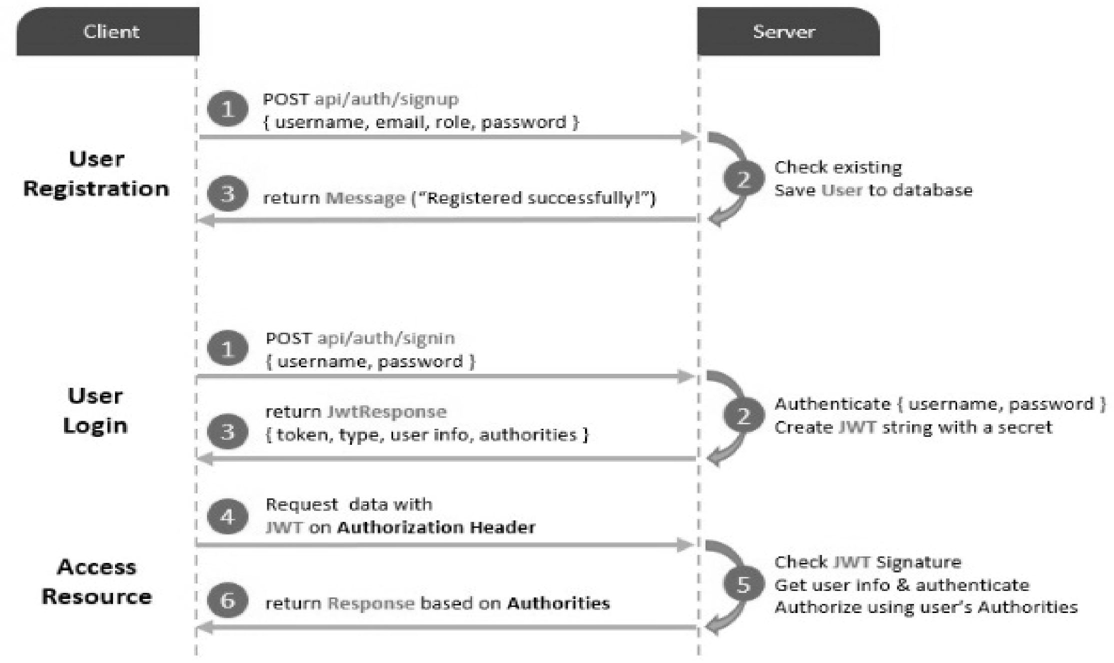

Client
Server
User Registration
1 POST api/auth/signup { username, email, role, password }
2 Check existing Save User to database
3 return Message ("Registered successfully!")
User Login
1 POST api/auth/signin { username, password }
2 Authenticate { username, password } Create JWT string with a secret
3 return JwtResponse { token, type, user info, authorities }
Access Resource
4 Request data with JWT on Authorization Header
5 Check JWT Signature Get user info & authenticate Authorize using user's Authorities
6 return Response based on Authorities

Figure 2:SEQUENCE DIAGRAM GRANT TOKEN

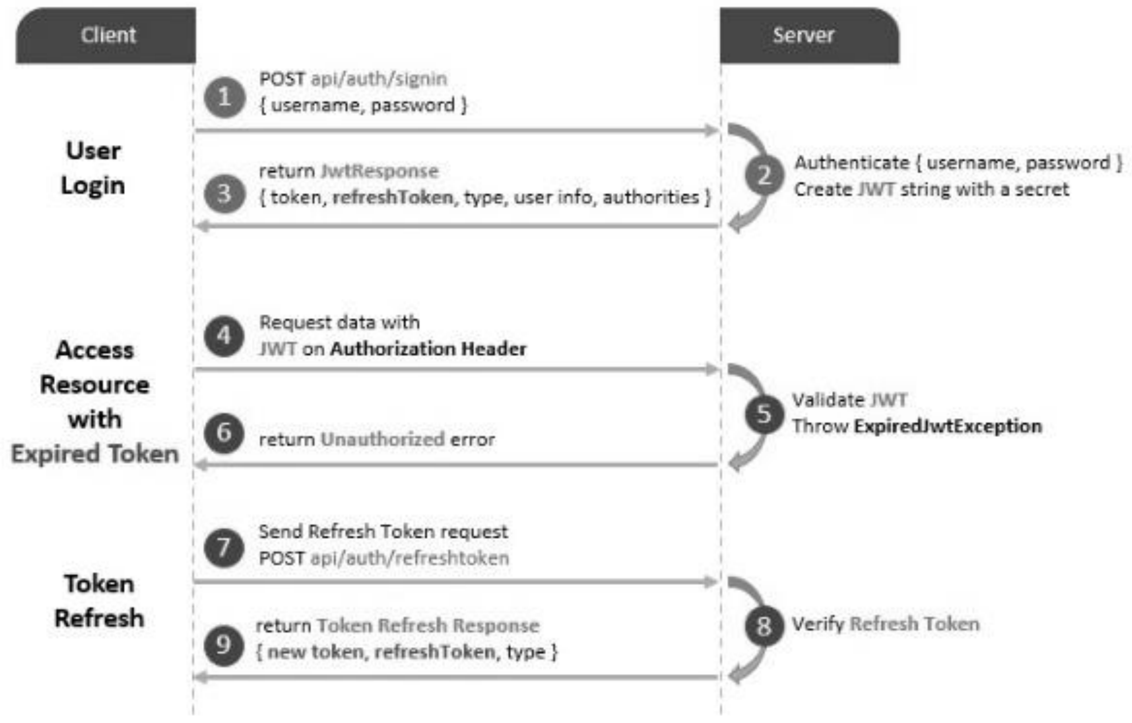
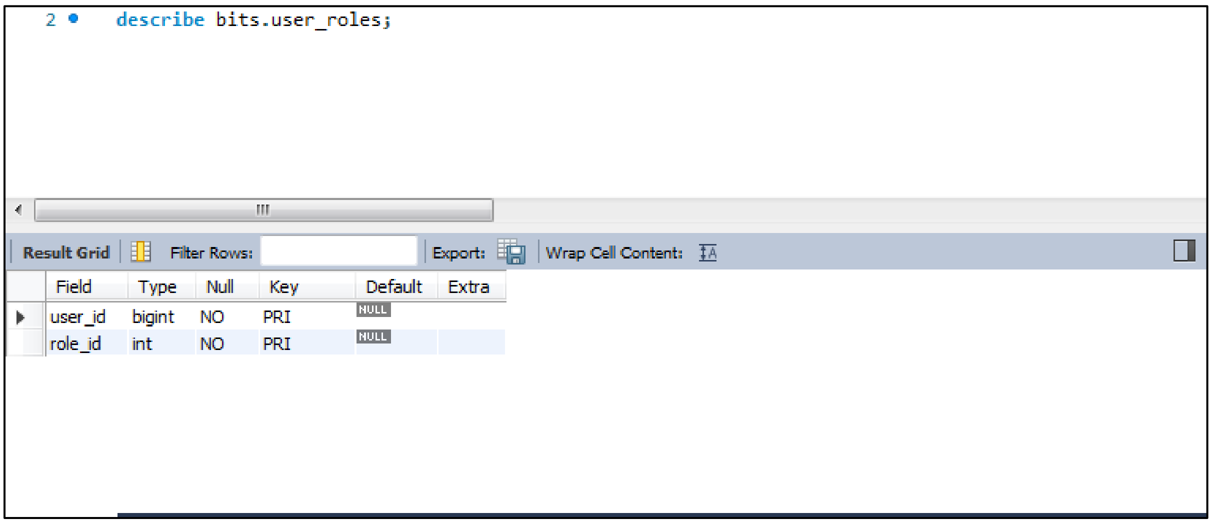

Client
Server
User Login
1 POST api/auth/signin { username, password }
2 Authenticate { username, password } Create JWT string with a secret
3 return JwtResponse { token, refreshToken, type, user info, authorities }
Access Resource with Expired Token
4 Request data with JWT on Authorization Header
5 Validate JWT Throw ExpiredJwtException
6 return Unauthorized error
Token Refresh
7 Send Refresh Token request POST api/auth/refreshtoken
8 Verify Refresh Token
9 return Token Refresh Response { new token, refreshToken, type }

Figure 3: SEQUENCE DIAGRAM REFRESH TOKEN

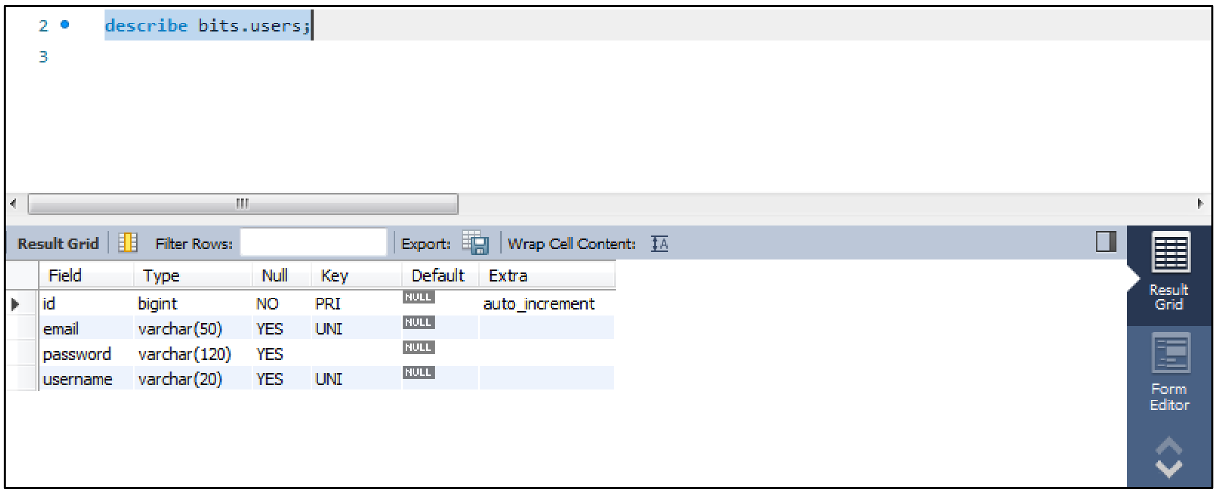
8.2 DATABASE DESIGN



```
describe bits.user_roles;
```

Field	Type	Null	Key	Default	Extra
user_id	bigint	NO	PRI	NULL	
role_id	int	NO	PRI	NULL	

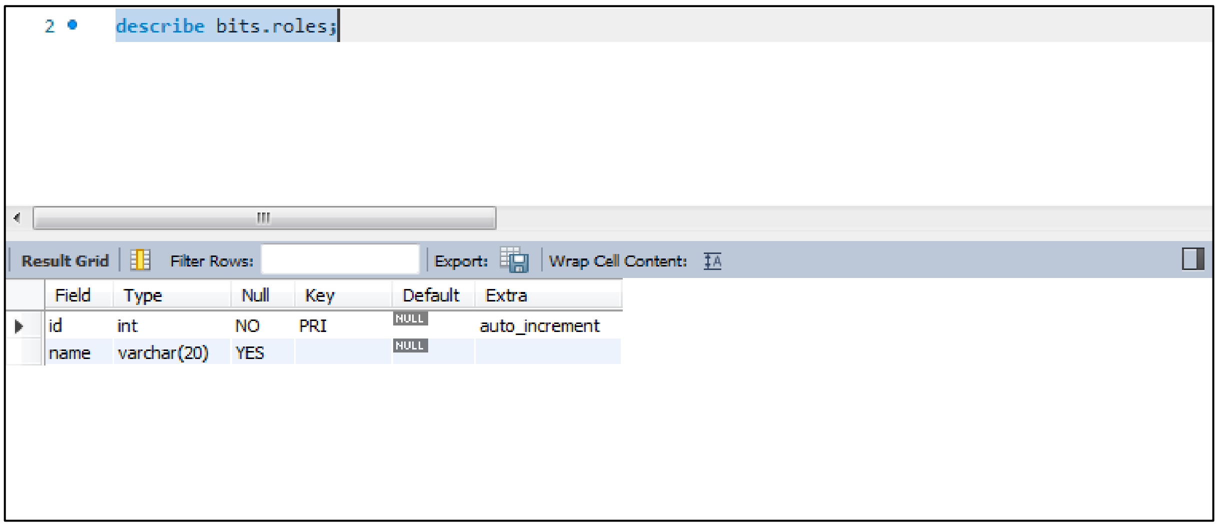
Figure 4: TABLE SCHEMA USER_ROLES



```
describe bits.users;
```

Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	auto_increment
email	varchar(50)	YES	UNI	NULL	
password	varchar(120)	YES		NULL	
username	varchar(20)	YES	UNI	NULL	

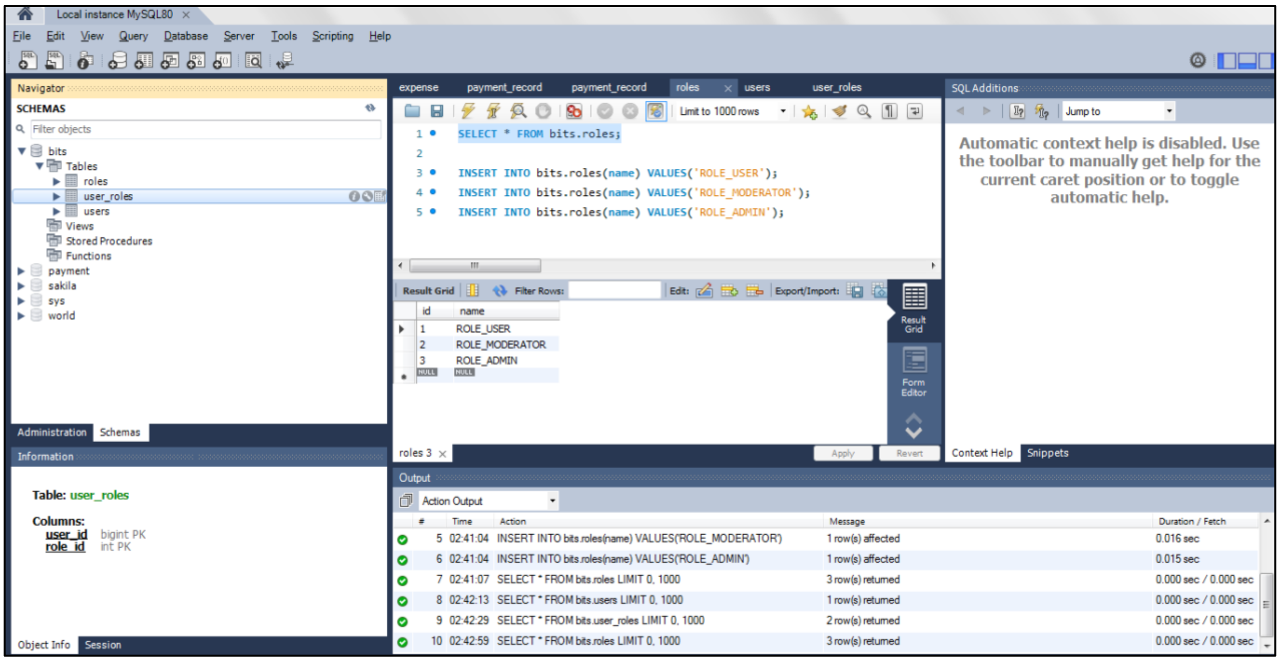
Figure 5: TABLE SCHEMA- USERS



```
describe bits.roles;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar(20)	YES		NULL	

Figure 6: TABLE SCHEMA- ROLES



```
SELECT * FROM bits.roles;

INSERT INTO bits.roles(name) VALUES('ROLE_USER');
INSERT INTO bits.roles(name) VALUES('ROLE_MODERATOR');
INSERT INTO bits.roles(name) VALUES('ROLE_ADMIN');
```

id	name
1	ROLE_USER
2	ROLE_MODERATOR
3	ROLE_ADMIN
NULL	NULL

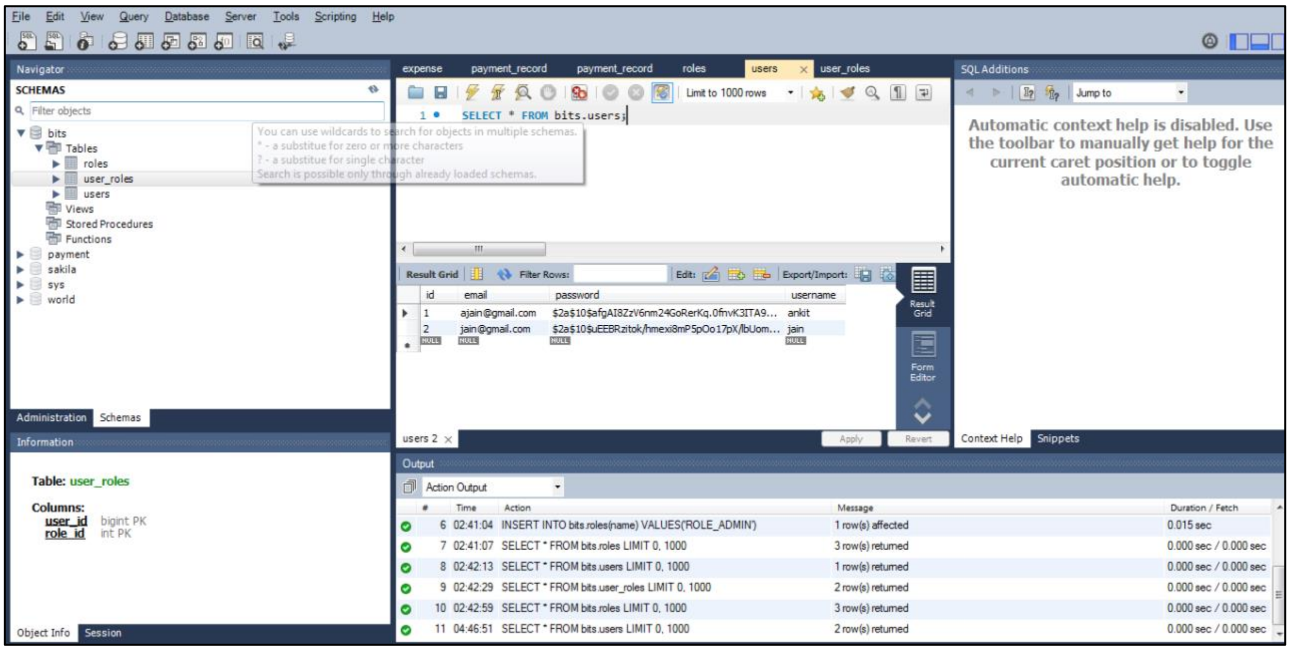
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Table: user_roles

Columns:
user_id bigint PK
role_id int PK

#	Time	Action	Message	Duration / Fetch
5	02:41:04	INSERT INTO bits.roles(name) VALUES('ROLE_MODERATOR')	1 row(s) affected	0.016 sec
6	02:41:04	INSERT INTO bits.roles(name) VALUES('ROLE_ADMIN')	1 row(s) affected	0.015 sec
7	02:41:07	SELECT * FROM bits.roles LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
8	02:42:13	SELECT * FROM bits.users LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
9	02:42:29	SELECT * FROM bits.user_roles LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
10	02:42:59	SELECT * FROM bits.roles LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

Figure 7: TABLE CONTENT ROLES



```
SELECT * FROM bits.users;
```

id	email	password	username
1	ajain@gmail.com	$2a$10$afgAI8ZzV6nm24GoRerKq.0fnvK3ITA9...	ankit
2	jain@gmail.com	$2a$10$uEEBRzitok/hmexi8mP5pOo17pX/lbUom...	jain
NULL	NULL	NULL	NULL

You can use wildcards to search for objects in multiple schemas.
* - a substitute for zero or more characters
? - a substitute for single character
Search is possible only through already loaded schemas.

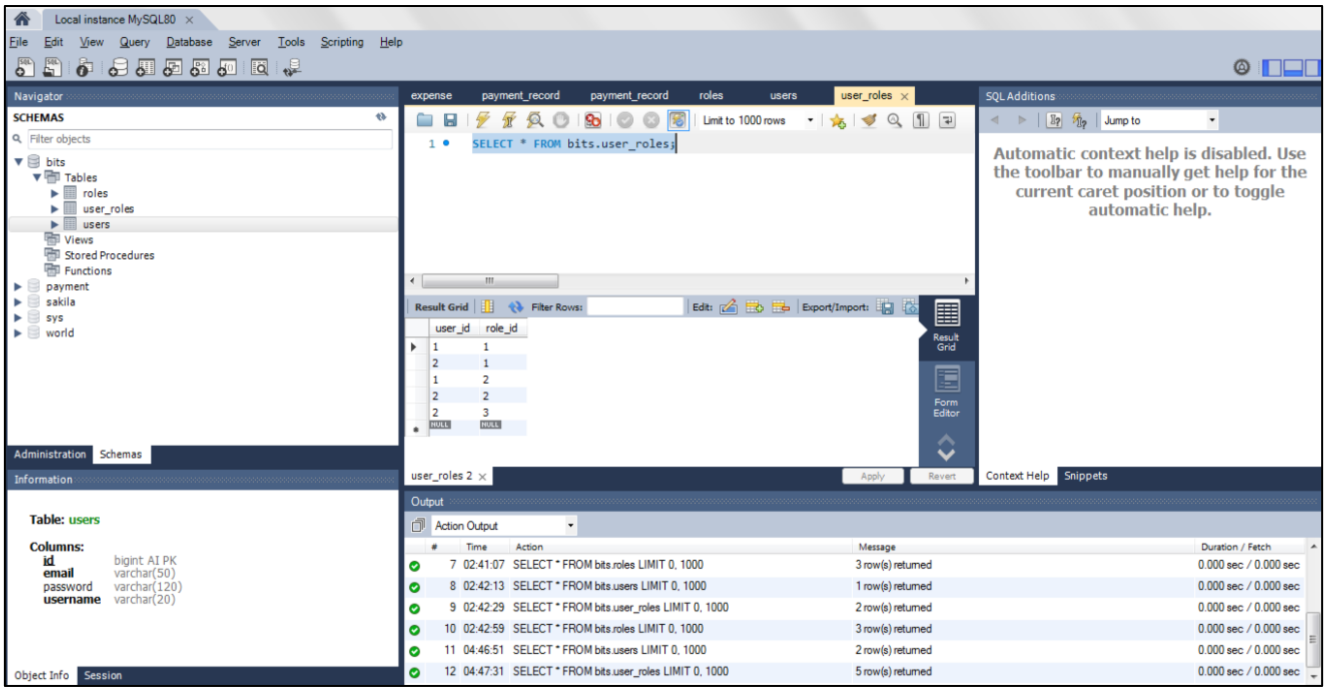
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Table: user_roles

Columns:
user_id bigint PK
role_id int PK

#	Time	Action	Message	Duration / Fetch
6	02:41:04	INSERT INTO bits.roles(name) VALUES('ROLE_ADMIN')	1 row(s) affected	0.015 sec
7	02:41:07	SELECT * FROM bits.roles LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
8	02:42:13	SELECT * FROM bits.users LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
9	02:42:29	SELECT * FROM bits.user_roles LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
10	02:42:59	SELECT * FROM bits.roles LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
11	04:46:51	SELECT * FROM bits.users LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec

Figure 8: TABLE CONTENT USERS



```
SELECT * FROM bits.user_roles;
```

user_id	role_id
1	1
2	1
1	2
2	2
2	3
NULL	NULL

Table: users

Columns:
id	bigint AI PK
email	varchar(50)
password	varchar(120)
username	varchar(20)

```
 #  Time      Action                                     Message              Duration / Fetch
 7  02:41:07  SELECT * FROM bits.roles LIMIT 0, 1000       3 row(s) returned    0.000 sec / 0.000 sec
 8  02:42:13  SELECT * FROM bits.users LIMIT 0, 1000       1 row(s) returned    0.000 sec / 0.000 sec
 9  02:42:29  SELECT * FROM bits.user_roles LIMIT 0, 1000  2 row(s) returned    0.000 sec / 0.000 sec
10  02:42:59  SELECT * FROM bits.roles LIMIT 0, 1000       3 row(s) returned    0.000 sec / 0.000 sec
11  04:46:51  SELECT * FROM bits.users LIMIT 0, 1000       2 row(s) returned    0.000 sec / 0.000 sec
12  04:47:31  SELECT * FROM bits.user_roles LIMIT 0, 1000  5 row(s) returned    0.000 sec / 0.000 sec
```

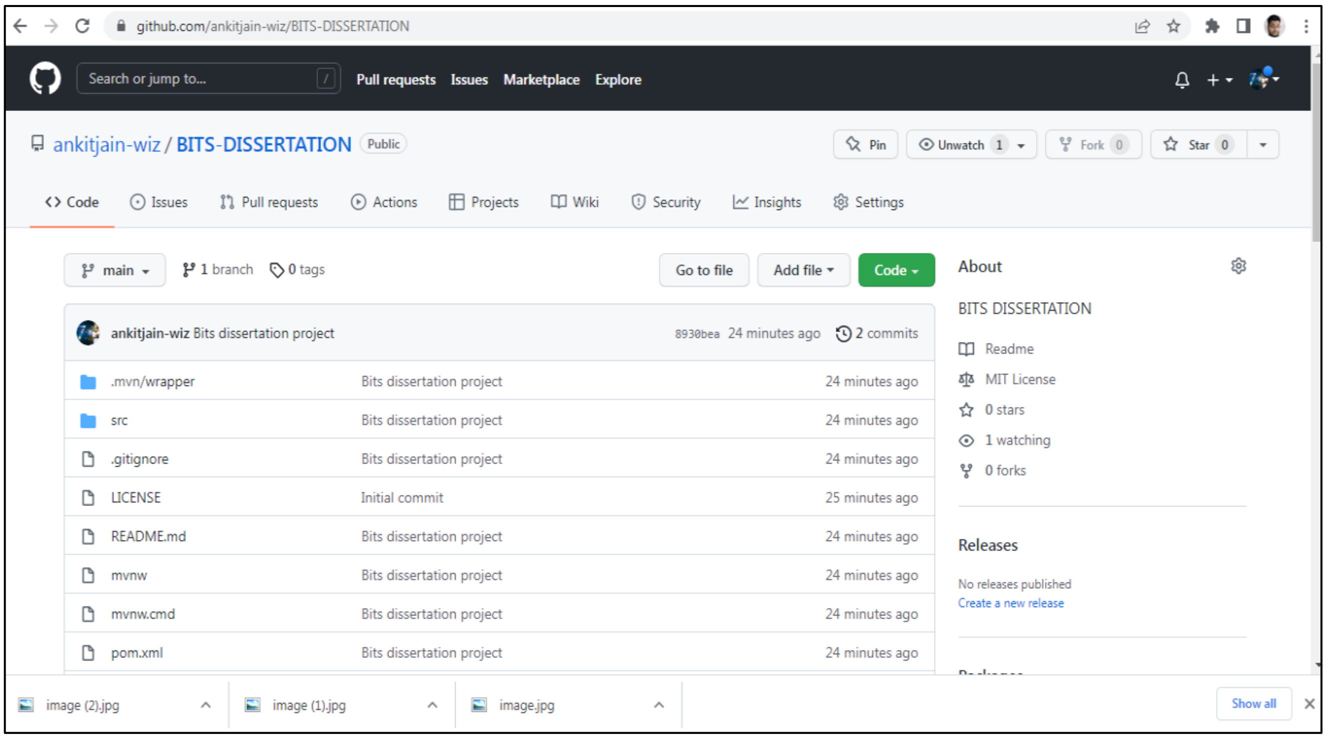
Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Figure 9: TABLE CONTENT USERS_ROLES

8.3 DEVELOPMENT AND CODEBASE

8.3.1 GITHUB

https://github.com/ankitjain-wiz/BITS-DISSERTATION



github.com/ankitjain-wiz/BITS-DISSERTATION

ankitjain-wiz / BITS-DISSERTATION Public

main · 1 branch · 0 tags

ankitjain-wiz Bits dissertation project · 8930bea 24 minutes ago · 2 commits

.mvn/wrapper	Bits dissertation project	24 minutes ago
src	Bits dissertation project	24 minutes ago
.gitignore	Bits dissertation project	24 minutes ago
LICENSE	Initial commit	25 minutes ago
README.md	Bits dissertation project	24 minutes ago
mvnw	Bits dissertation project	24 minutes ago
mvnw.cmd	Bits dissertation project	24 minutes ago
pom.xml	Bits dissertation project	24 minutes ago

About

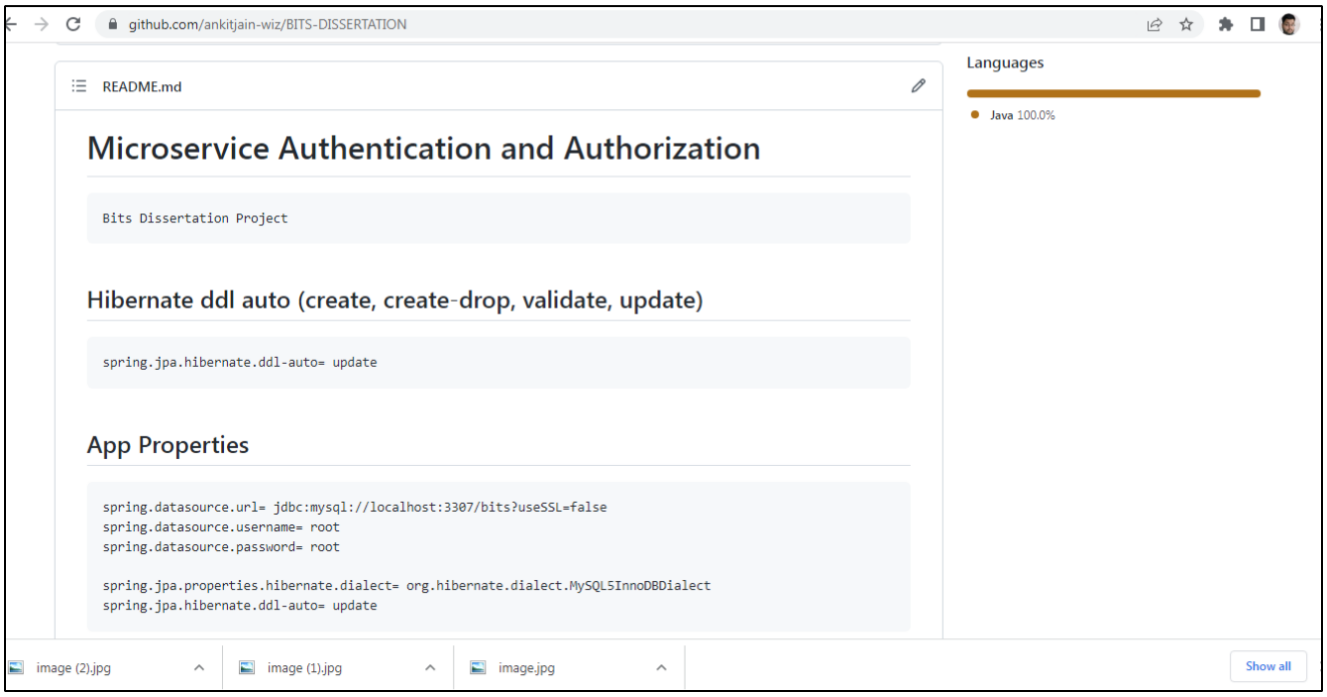
BITS DISSERTATION

Readme
MIT License
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Figure 10: GITHUB REPOSITORY



Microservice Authentication and Authorization

```
Bits Dissertation Project
```

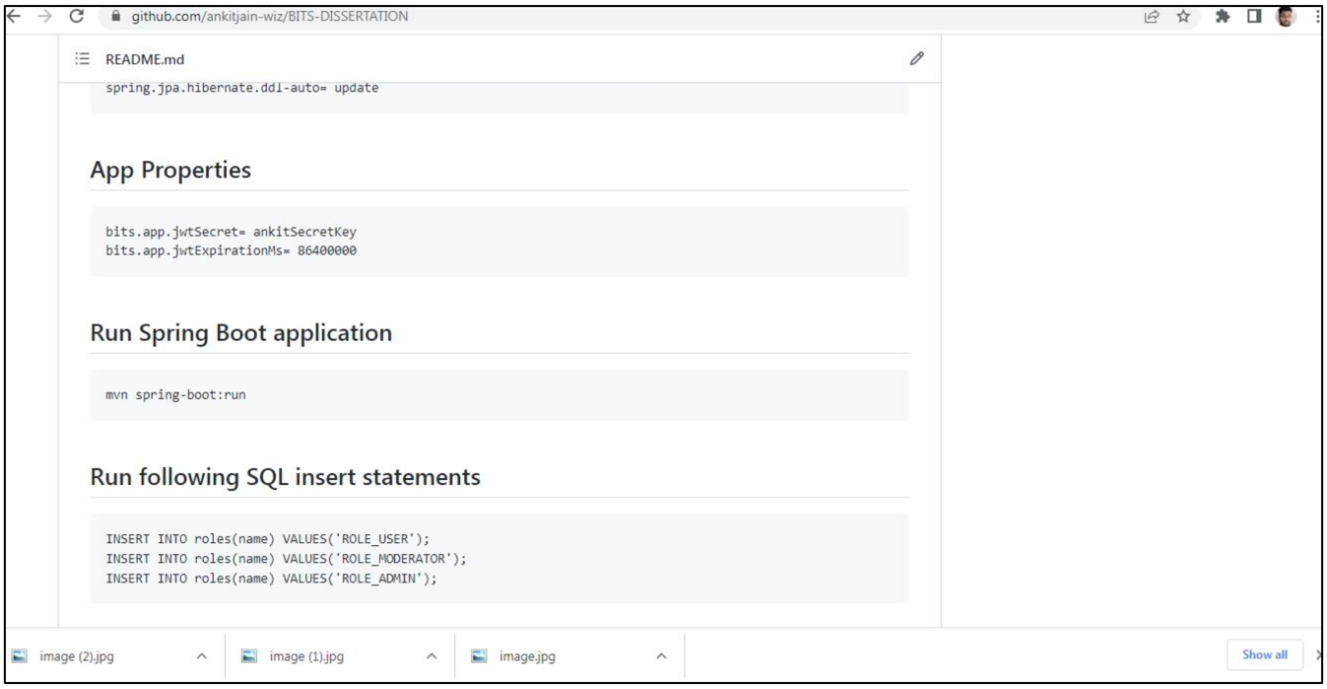
Hibernate ddl auto (create, create-drop, validate, update)

```
spring.jpa.hibernate.ddl-auto= update
```

App Properties

```
spring.datasource.url= jdbc:mysql://localhost:3307/bits?useSSL=false
spring.datasource.username= root
spring.datasource.password= root

spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto= update
```

Figure 11:README.MD



```
spring.jpa.hibernate.ddl-auto= update
```

App Properties

```
bits.app.jwtSecret= ankitSecretKey
bits.app.jwtExpirationMs= 86400000
```

Run Spring Boot application

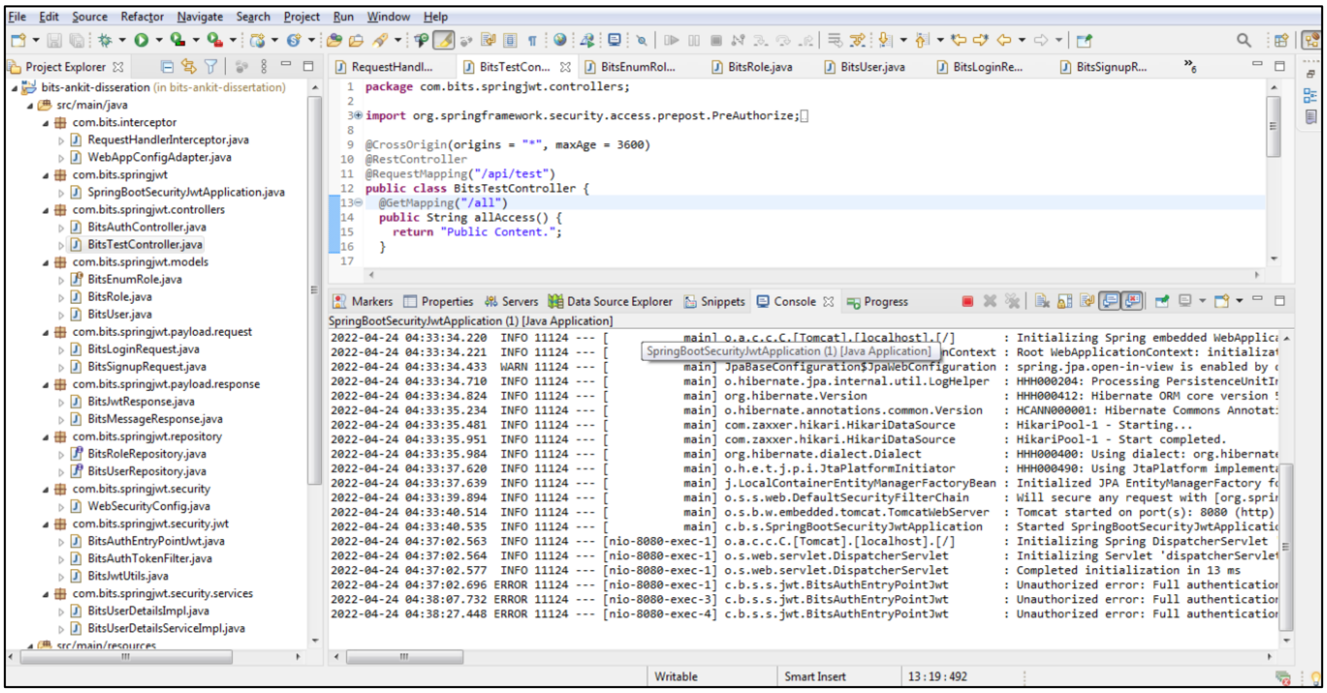
```
mvn spring-boot:run
```

Run following SQL insert statements

```
INSERT INTO roles(name) VALUES('ROLE_USER');
INSERT INTO roles(name) VALUES('ROLE_MODERATOR');
INSERT INTO roles(name) VALUES('ROLE_ADMIN');
```

Figure 12: README.MD

8.3.2 CODEBASE



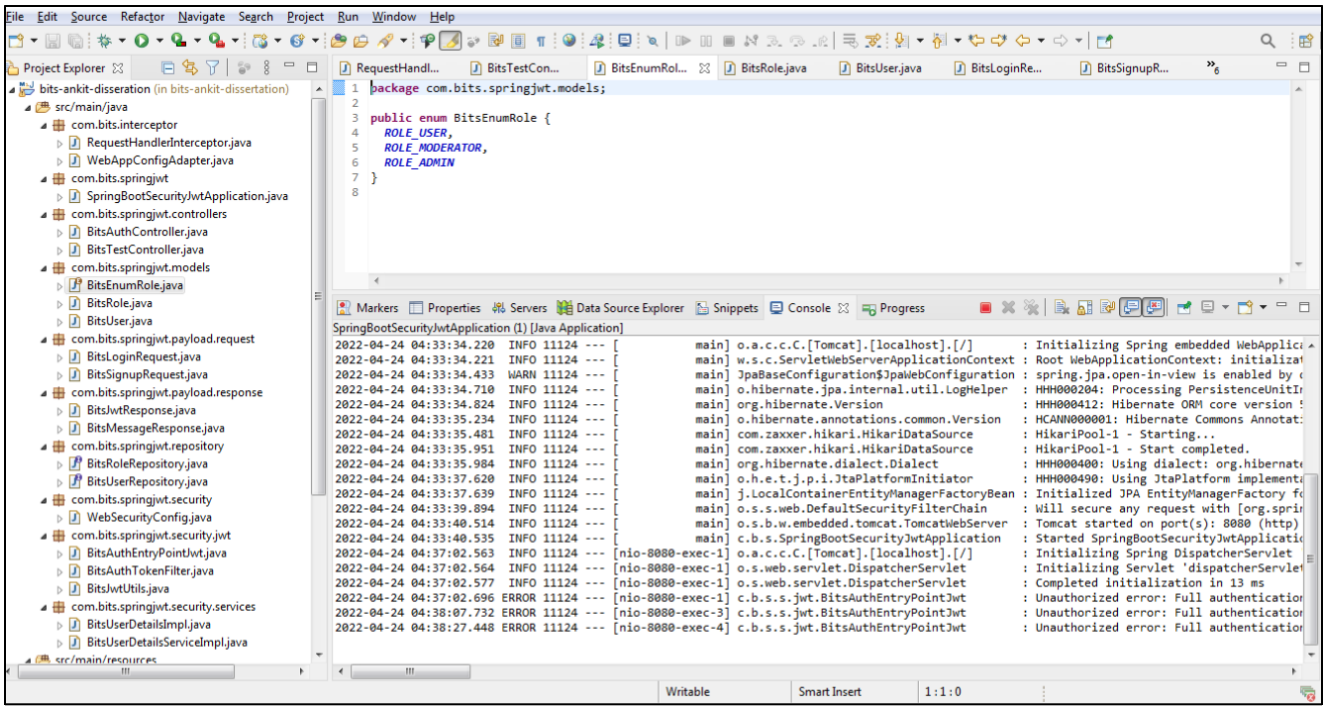
```
bits-ankit-disseration (in bits-ankit-dissertation)
  src/main/java
    com.bits.interceptor
      RequestHandlerInterceptor.java
      WebAppConfigAdapter.java
    com.bits.springjwt
      SpringBootSecurityJwtApplication.java
    com.bits.springjwt.controllers
      BitsAuthController.java
      BitsTestController.java
    com.bits.springjwt.models
      BitsEnumRole.java
      BitsRole.java
      BitsUser.java
    com.bits.springjwt.payload.request
      BitsLoginRequest.java
      BitsSignupRequest.java
    com.bits.springjwt.payload.response
      BitsJwtResponse.java
      BitsMessageResponse.java
    com.bits.springjwt.repository
      BitsRoleRepository.java
      BitsUserRepository.java
    com.bits.springjwt.security
      WebSecurityConfig.java
    com.bits.springjwt.security.jwt
      BitsAuthEntryPointJwt.java
      BitsAuthTokenFilter.java
      BitsJwtUtils.java
    com.bits.springjwt.security.services
      BitsUserDetailsImpl.java
      BitsUserDetailsServiceImpl.java
  src/main/resources
```

```java
package com.bits.springjwt.controllers;

import org.springframework.security.access.prepost.PreAuthorize;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/api/test")
public class BitsTestController {
  @GetMapping("/all")
  public String allAccess() {
    return "Public Content.";
  }
```

SpringBootSecurityJwtApplication (1) [Java Application]

```
2022-04-24 04:33:34.220  INFO 11124 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplic
2022-04-24 04:33:34.221  INFO 11124 --- [  SpringBootSecurityJwtApplication (1) [Java Application] nContext : Root WebApplicationContext: initializat
2022-04-24 04:33:34.433  WARN 11124 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by c
2022-04-24 04:33:34.710  INFO 11124 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitIn
2022-04-24 04:33:34.824  INFO 11124 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5
2022-04-24 04:33:35.234  INFO 11124 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotat:
2022-04-24 04:33:35.481  INFO 11124 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-04-24 04:33:35.951  INFO 11124 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-04-24 04:33:35.984  INFO 11124 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate
2022-04-24 04:33:37.620  INFO 11124 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementa
2022-04-24 04:33:37.639  INFO 11124 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory fo
2022-04-24 04:33:39.894  INFO 11124 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.sprin
2022-04-24 04:33:40.514  INFO 11124 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http)
2022-04-24 04:33:40.535  INFO 11124 --- [           main] c.b.s.SpringBootSecurityJwtApplication   : Started SpringBootSecurityJwtApplicatio
2022-04-24 04:37:02.563  INFO 11124 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet
2022-04-24 04:37:02.564  INFO 11124 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet
2022-04-24 04:37:02.577  INFO 11124 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 13 ms
2022-04-24 04:37:02.696 ERROR 11124 --- [nio-8080-exec-1] c.b.s.s.jwt.BitsAuthEntryPointJwt        : Unauthorized error: Full authenticatior
2022-04-24 04:38:07.732 ERROR 11124 --- [nio-8080-exec-3] c.b.s.s.jwt.BitsAuthEntryPointJwt        : Unauthorized error: Full authenticatior
2022-04-24 04:38:27.448 ERROR 11124 --- [nio-8080-exec-4] c.b.s.s.jwt.BitsAuthEntryPointJwt        : Unauthorized error: Full authenticatior
```

Figure 13: CODEBASE 1

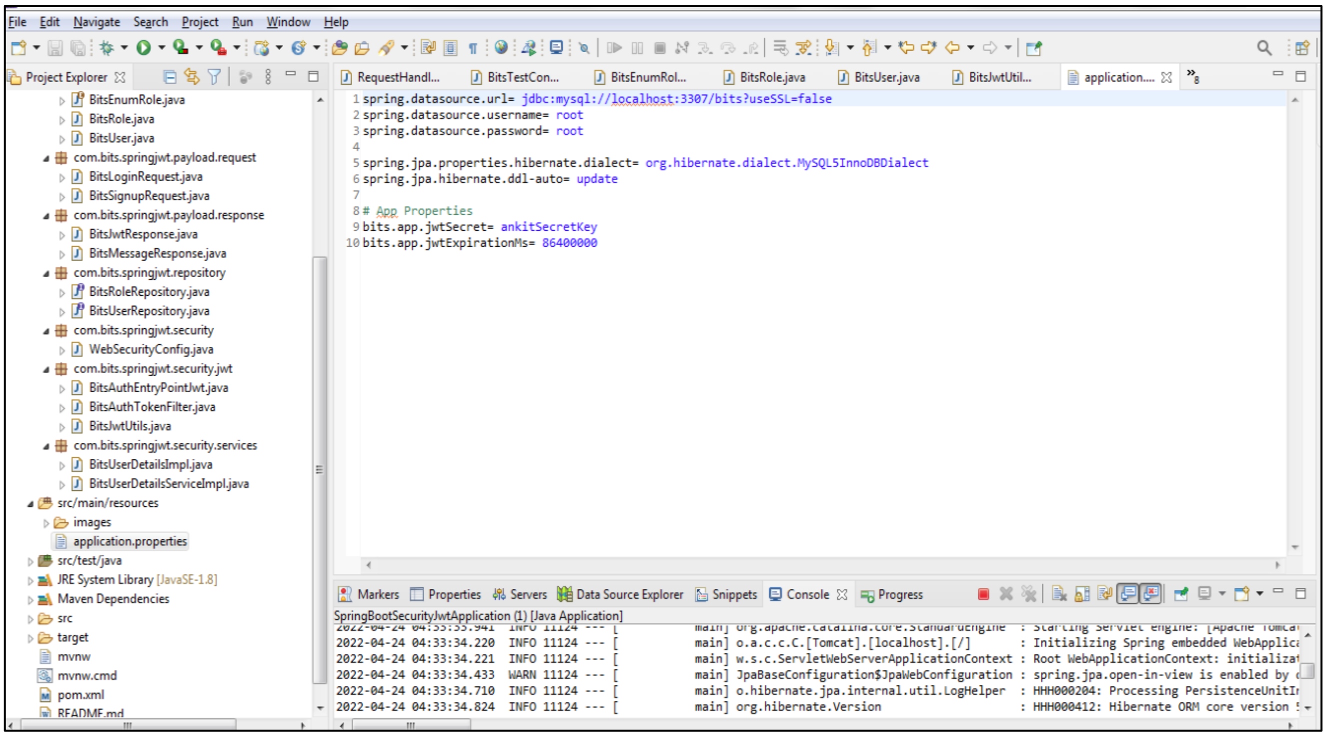


```java
package com.bits.springjwt.models;

public enum BitsEnumRole {
  ROLE_USER,
  ROLE_MODERATOR,
  ROLE_ADMIN
}
```

SpringBootSecurityJwtApplication (1) [Java Application]

```
2022-04-24 04:33:34.220  INFO 11124 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplic
2022-04-24 04:33:34.221  INFO 11124 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initializat
2022-04-24 04:33:34.433  WARN 11124 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by c
2022-04-24 04:33:34.710  INFO 11124 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitIn
2022-04-24 04:33:34.824  INFO 11124 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5
2022-04-24 04:33:35.234  INFO 11124 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotat:
2022-04-24 04:33:35.481  INFO 11124 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-04-24 04:33:35.951  INFO 11124 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-04-24 04:33:35.984  INFO 11124 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate
2022-04-24 04:33:37.620  INFO 11124 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementa
2022-04-24 04:33:37.639  INFO 11124 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory fo
2022-04-24 04:33:39.894  INFO 11124 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.sprin
2022-04-24 04:33:40.514  INFO 11124 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http)
2022-04-24 04:33:40.535  INFO 11124 --- [           main] c.b.s.SpringBootSecurityJwtApplication   : Started SpringBootSecurityJwtApplicatio
2022-04-24 04:37:02.563  INFO 11124 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet
2022-04-24 04:37:02.564  INFO 11124 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet
2022-04-24 04:37:02.577  INFO 11124 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 13 ms
2022-04-24 04:37:02.696 ERROR 11124 --- [nio-8080-exec-1] c.b.s.s.jwt.BitsAuthEntryPointJwt        : Unauthorized error: Full authenticatior
2022-04-24 04:38:07.732 ERROR 11124 --- [nio-8080-exec-3] c.b.s.s.jwt.BitsAuthEntryPointJwt        : Unauthorized error: Full authenticatior
2022-04-24 04:38:27.448 ERROR 11124 --- [nio-8080-exec-4] c.b.s.s.jwt.BitsAuthEntryPointJwt        : Unauthorized error: Full authenticatior
```

Figure 14: CODEBASE 2

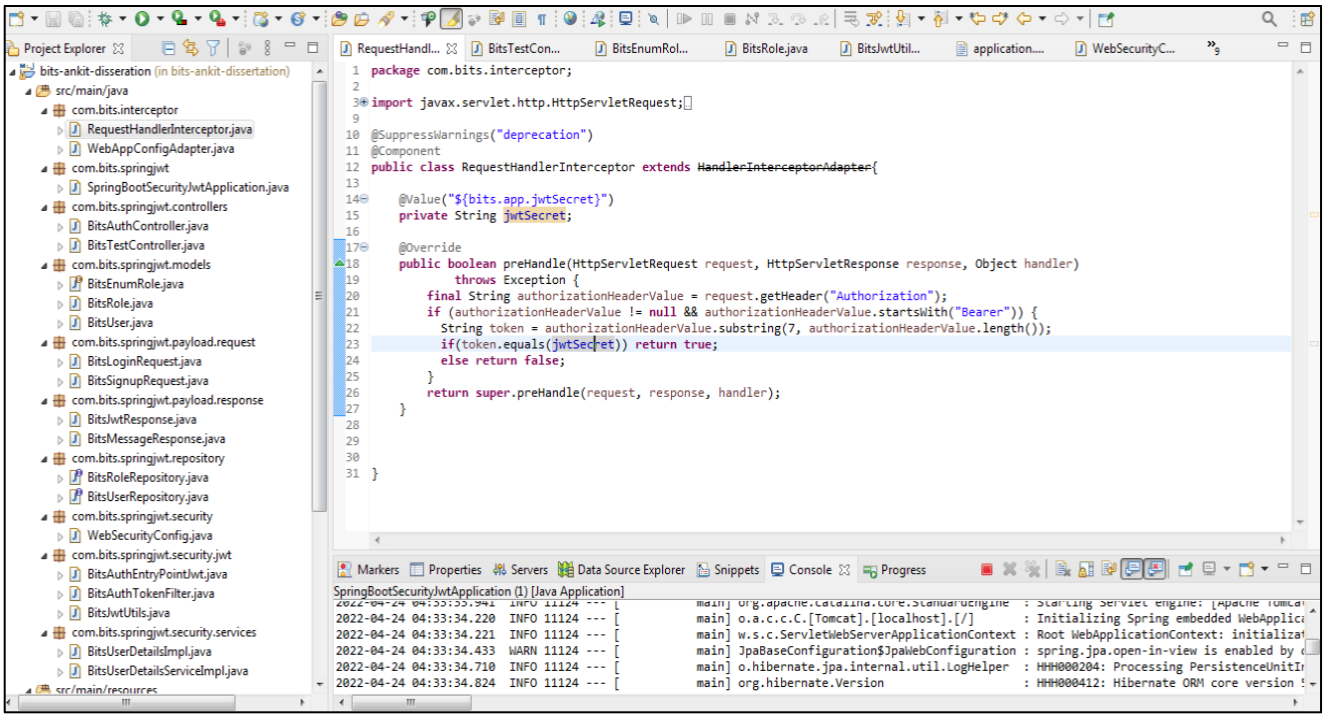

File Edit Navigate Search Project Run Window Help
Project Explorer
BitsEnumRole.java
BitsRole.java
BitsUser.java
com.bits.springjwt.payload.request
BitsLoginRequest.java
BitsSignupRequest.java
com.bits.springjwt.payload.response
BitsJwtResponse.java
BitsMessageResponse.java
com.bits.springjwt.repository
BitsRoleRepository.java
BitsUserRepository.java
com.bits.springjwt.security
WebSecurityConfig.java
com.bits.springjwt.security.jwt
BitsAuthEntryPointJwt.java
BitsAuthTokenFilter.java
BitsJwtUtils.java
com.bits.springjwt.security.services
BitsUserDetailsImpl.java
BitsUserDetailsServiceImpl.java
src/main/resources
images
application.properties
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
mvnw
mvnw.cmd
pom.xml
README.md
RequestHandl... BitsTestCon... BitsEnumRol... BitsRole.java BitsUser.java BitsJwtUtil... application....
Markers Properties Servers Data Source Explorer Snippets Console Progress

```
spring.datasource.url= jdbc:mysql://localhost:3307/bits?useSSL=false
spring.datasource.username= root
spring.datasource.password= root

spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto= update

# App Properties
bits.app.jwtSecret= ankitSecretKey
bits.app.jwtExpirationMs= 86400000
```

SpringBootSecurityJwtApplication (1) [Java Application]

```
2022-04-24 04:33:33.941  INFO 11124 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat
2022-04-24 04:33:34.220  INFO 11124 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplica
2022-04-24 04:33:34.221  INFO 11124 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initializat
2022-04-24 04:33:34.433  WARN 11124 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by d
2022-04-24 04:33:34.710  INFO 11124 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitIn
2022-04-24 04:33:34.824  INFO 11124 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5
```

Figure 15: CODEBASE 3

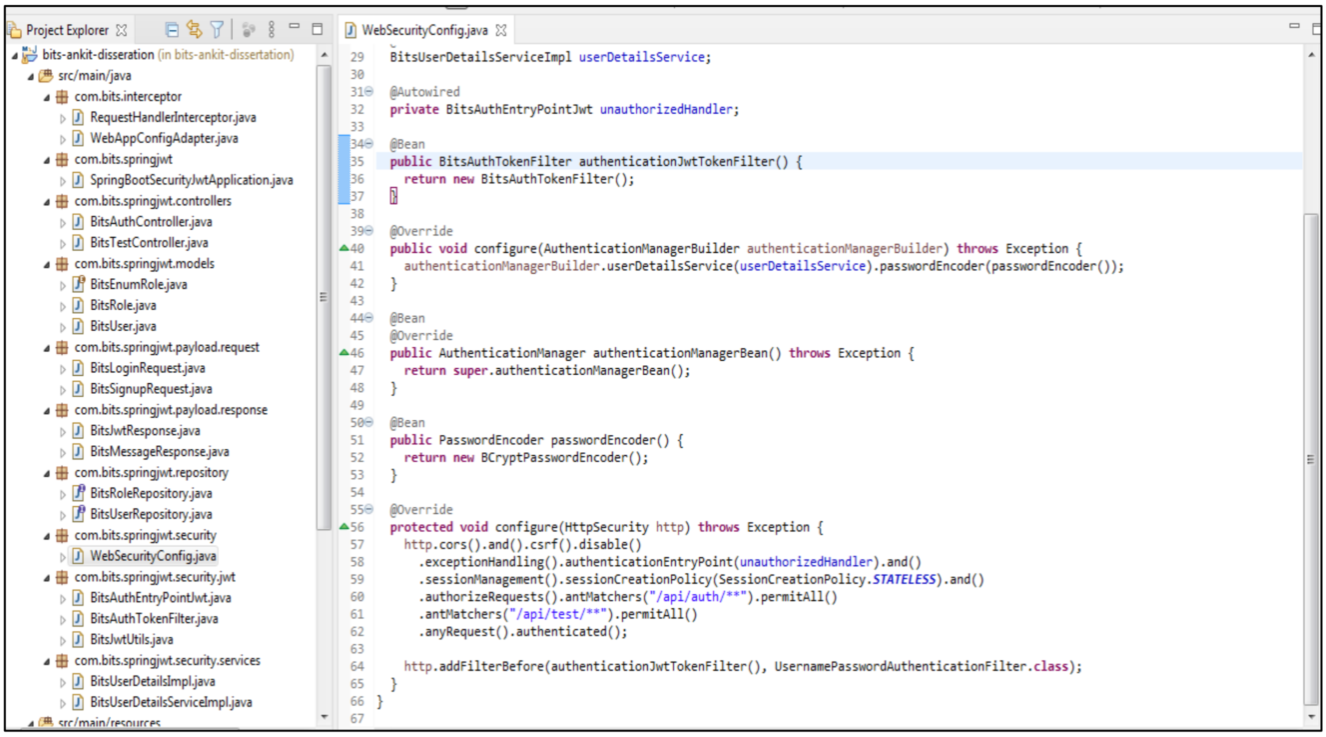

Project Explorer
bits-ankit-disseration (in bits-ankit-dissertation)
src/main/java
com.bits.interceptor
RequestHandlerInterceptor.java
WebAppConfigAdapter.java
com.bits.springjwt
SpringBootSecurityJwtApplication.java
com.bits.springjwt.controllers
BitsAuthController.java
BitsTestController.java
com.bits.springjwt.models
BitsEnumRole.java
BitsRole.java
BitsUser.java
com.bits.springjwt.payload.request
BitsLoginRequest.java
BitsSignupRequest.java
com.bits.springjwt.payload.response
BitsJwtResponse.java
BitsMessageResponse.java
com.bits.springjwt.repository
BitsRoleRepository.java
BitsUserRepository.java
com.bits.springjwt.security
WebSecurityConfig.java
com.bits.springjwt.security.jwt
BitsAuthEntryPointJwt.java
BitsAuthTokenFilter.java
BitsJwtUtils.java
com.bits.springjwt.security.services
BitsUserDetailsImpl.java
BitsUserDetailsServiceImpl.java
src/main/resources
RequestHandl... BitsTestCon... BitsEnumRol... BitsRole.java BitsJwtUtil... application.... WebSecurityC...
Markers Properties Servers Data Source Explorer Snippets Console Progress

```
package com.bits.interceptor;

import javax.servlet.http.HttpServletRequest;

@SuppressWarnings("deprecation")
@Component
public class RequestHandlerInterceptor extends ~~HandlerInterceptorAdapter~~{

    @Value("${bits.app.jwtSecret}")
    private String jwtSecret;

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
            throws Exception {
        final String authorizationHeaderValue = request.getHeader("Authorization");
        if (authorizationHeaderValue != null && authorizationHeaderValue.startsWith("Bearer")) {
          String token = authorizationHeaderValue.substring(7, authorizationHeaderValue.length());
          if(token.equals(jwtSecret)) return true;
          else return false;
        }
        return super.preHandle(request, response, handler);
    }




}
```

SpringBootSecurityJwtApplication (1) [Java Application]

```
2022-04-24 04:33:33.941  INFO 11124 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat
2022-04-24 04:33:34.220  INFO 11124 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplica
2022-04-24 04:33:34.221  INFO 11124 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initializat
2022-04-24 04:33:34.433  WARN 11124 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by d
2022-04-24 04:33:34.710  INFO 11124 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitIn
2022-04-24 04:33:34.824  INFO 11124 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5
```

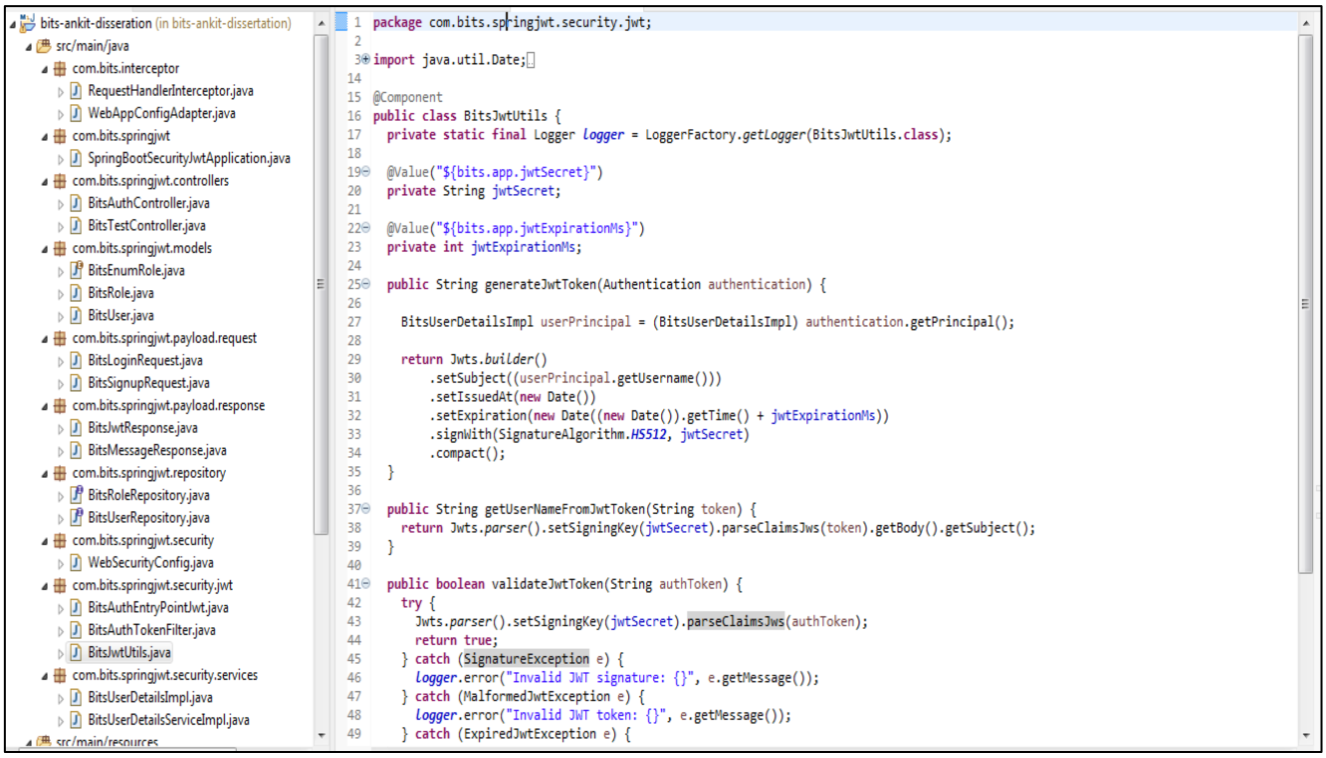
Figure 16: CODEBASE 4



```
Project Explorer
bits-ankit-disseration (in bits-ankit-dissertation)
  src/main/java
    com.bits.interceptor
      RequestHandlerInterceptor.java
      WebAppConfigAdapter.java
    com.bits.springjwt
      SpringBootSecurityJwtApplication.java
    com.bits.springjwt.controllers
      BitsAuthController.java
      BitsTestController.java
    com.bits.springjwt.models
      BitsEnumRole.java
      BitsRole.java
      BitsUser.java
    com.bits.springjwt.payload.request
      BitsLoginRequest.java
      BitsSignupRequest.java
    com.bits.springjwt.payload.response
      BitsJwtResponse.java
      BitsMessageResponse.java
    com.bits.springjwt.repository
      BitsRoleRepository.java
      BitsUserRepository.java
    com.bits.springjwt.security
      WebSecurityConfig.java
    com.bits.springjwt.security.jwt
      BitsAuthEntryPointJwt.java
      BitsAuthTokenFilter.java
      BitsJwtUtils.java
    com.bits.springjwt.security.services
      BitsUserDetailsImpl.java
      BitsUserDetailsServiceImpl.java
  src/main/resources
```

```java
WebSecurityConfig.java

  BitsUserDetailsServiceImpl userDetailsService;

  @Autowired
  private BitsAuthEntryPointJwt unauthorizedHandler;

  @Bean
  public BitsAuthTokenFilter authenticationJwtTokenFilter() {
    return new BitsAuthTokenFilter();
  }

  @Override
  public void configure(AuthenticationManagerBuilder authenticationManagerBuilder) throws Exception {
    authenticationManagerBuilder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
  }

  @Bean
  @Override
  public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
  }

  @Bean
  public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
  }

  @Override
  protected void configure(HttpSecurity http) throws Exception {
    http.cors().and().csrf().disable()
      .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()
      .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
      .authorizeRequests().antMatchers("/api/auth/**").permitAll()
      .antMatchers("/api/test/**").permitAll()
      .anyRequest().authenticated();

    http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);
  }
}
```

Figure 17: CODEBASE 5

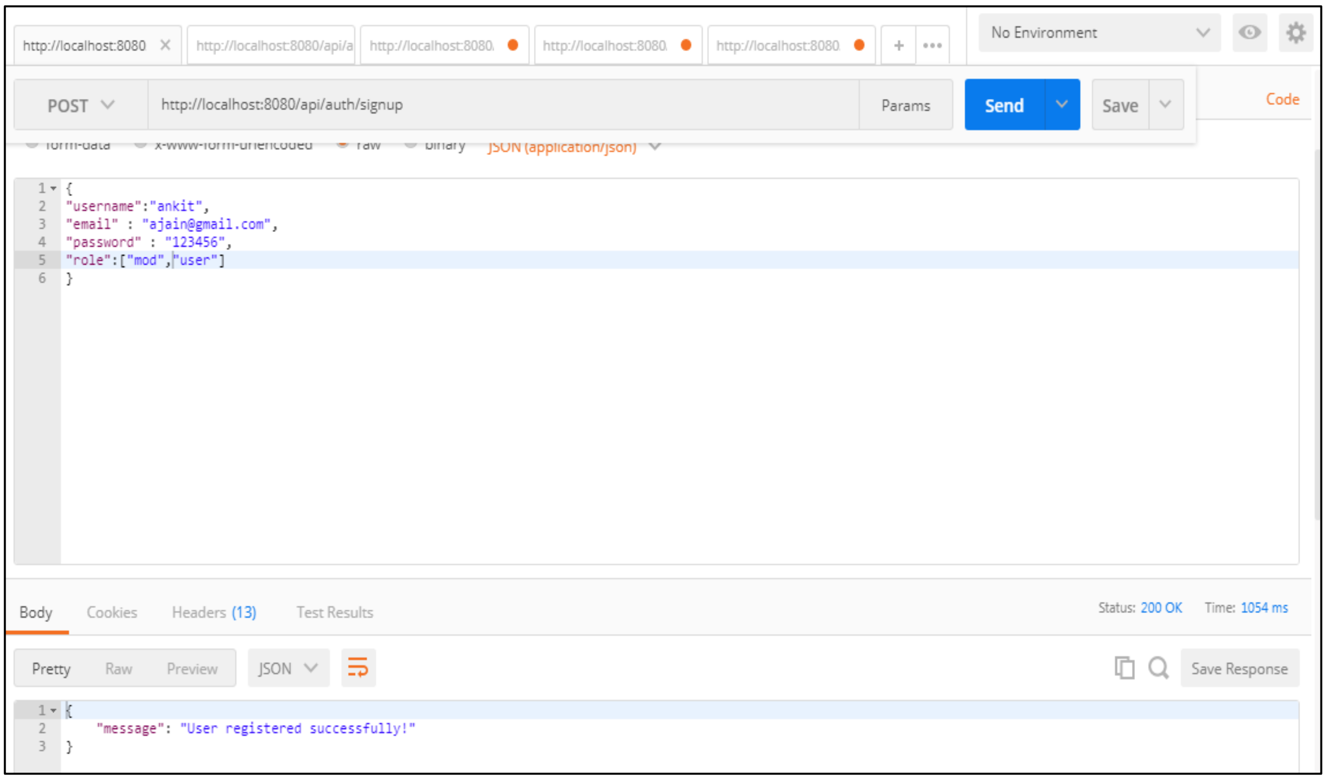


```
bits-ankit-disseration (in bits-ankit-dissertation)
  src/main/java
    com.bits.interceptor
      RequestHandlerInterceptor.java
      WebAppConfigAdapter.java
    com.bits.springjwt
      SpringBootSecurityJwtApplication.java
    com.bits.springjwt.controllers
      BitsAuthController.java
      BitsTestController.java
    com.bits.springjwt.models
      BitsEnumRole.java
      BitsRole.java
      BitsUser.java
    com.bits.springjwt.payload.request
      BitsLoginRequest.java
      BitsSignupRequest.java
    com.bits.springjwt.payload.response
      BitsJwtResponse.java
      BitsMessageResponse.java
    com.bits.springjwt.repository
      BitsRoleRepository.java
      BitsUserRepository.java
    com.bits.springjwt.security
      WebSecurityConfig.java
    com.bits.springjwt.security.jwt
      BitsAuthEntryPointJwt.java
      BitsAuthTokenFilter.java
      BitsJwtUtils.java
    com.bits.springjwt.security.services
      BitsUserDetailsImpl.java
      BitsUserDetailsServiceImpl.java
  src/main/resources
```

```java
package com.bits.springjwt.security.jwt;

import java.util.Date;

@Component
public class BitsJwtUtils {
  private static final Logger logger = LoggerFactory.getLogger(BitsJwtUtils.class);

  @Value("${bits.app.jwtSecret}")
  private String jwtSecret;

  @Value("${bits.app.jwtExpirationMs}")
  private int jwtExpirationMs;

  public String generateJwtToken(Authentication authentication) {

    BitsUserDetailsImpl userPrincipal = (BitsUserDetailsImpl) authentication.getPrincipal();

    return Jwts.builder()
        .setSubject((userPrincipal.getUsername()))
        .setIssuedAt(new Date())
        .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))
        .signWith(SignatureAlgorithm.HS512, jwtSecret)
        .compact();
  }

  public String getUserNameFromJwtToken(String token) {
    return Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();
  }

  public boolean validateJwtToken(String authToken) {
    try {
      Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
      return true;
    } catch (SignatureException e) {
      logger.error("Invalid JWT signature: {}", e.getMessage());
    } catch (MalformedJwtException e) {
      logger.error("Invalid JWT token: {}", e.getMessage());
    } catch (ExpiredJwtException e) {
```

Figure 18: CODEBASE 6

8.4 WALKTHROUGH OF APPLICATION



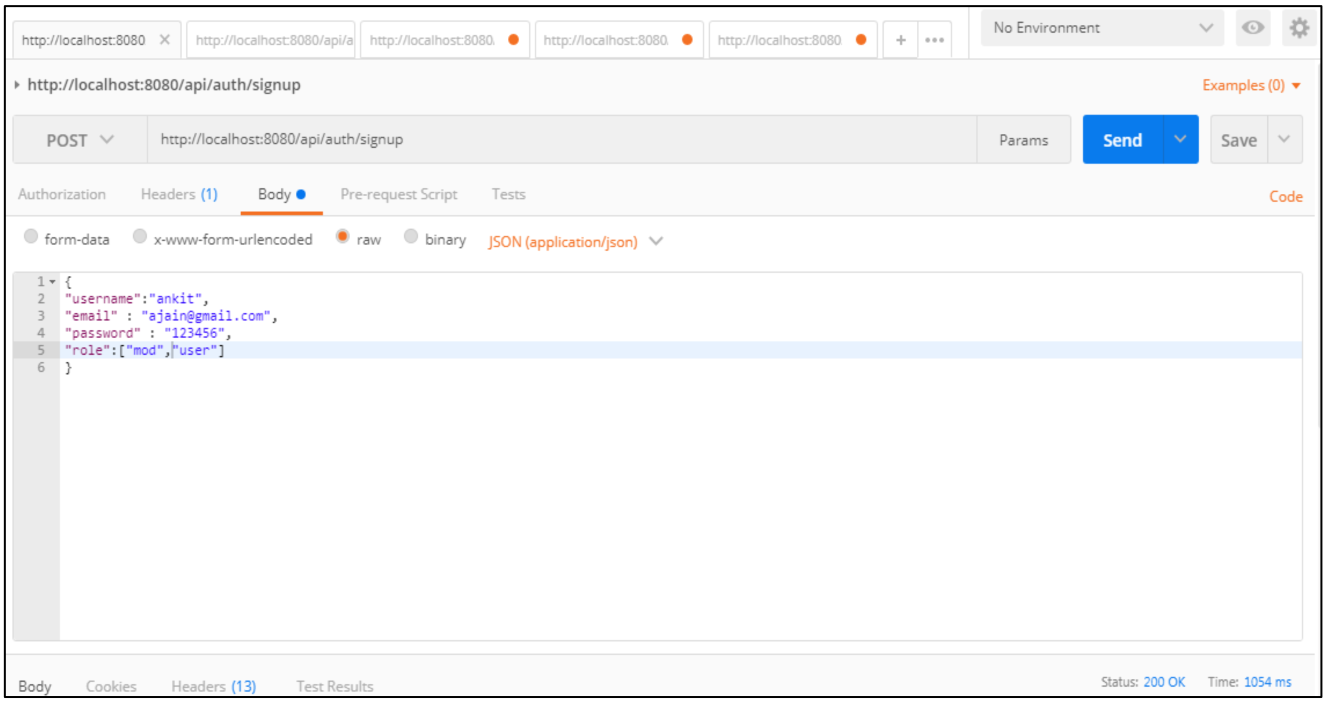
POST http://localhost:8080/api/auth/signup

```
{
"username":"ankit",
"email" : "ajain@gmail.com",
"password" : "123456",
"role":["mod","user"]
}
```

Status: 200 OK Time: 1054 ms

```
{
    "message": "User registered successfully!"
}
```

Figure 19: SIGNUP API



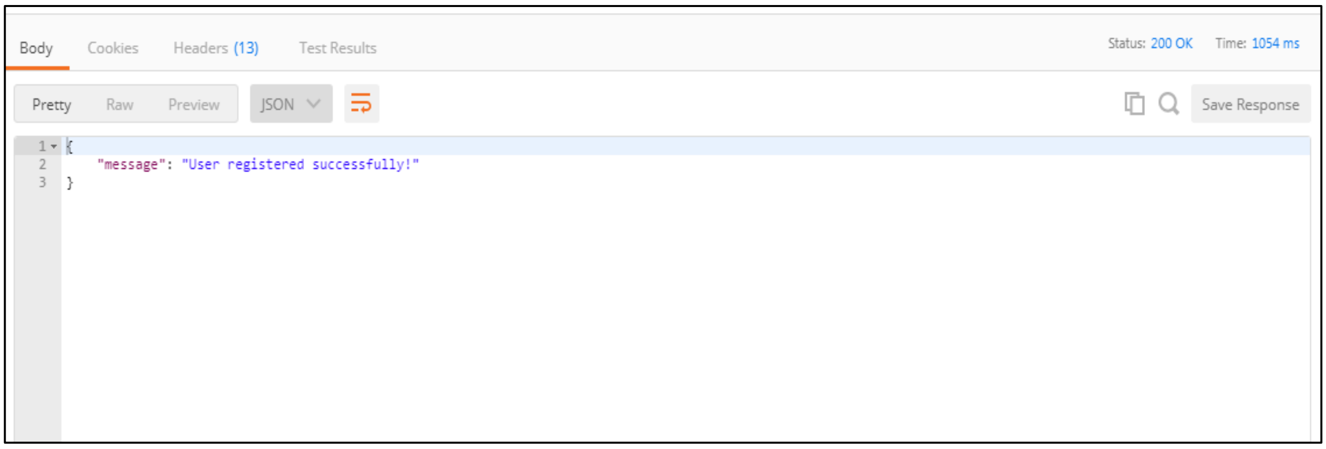
http://localhost:8080/api/auth/signup

POST http://localhost:8080/api/auth/signup

```
{
"username":"ankit",
"email" : "ajain@gmail.com",
"password" : "123456",
"role":["mod","user"]
}
```

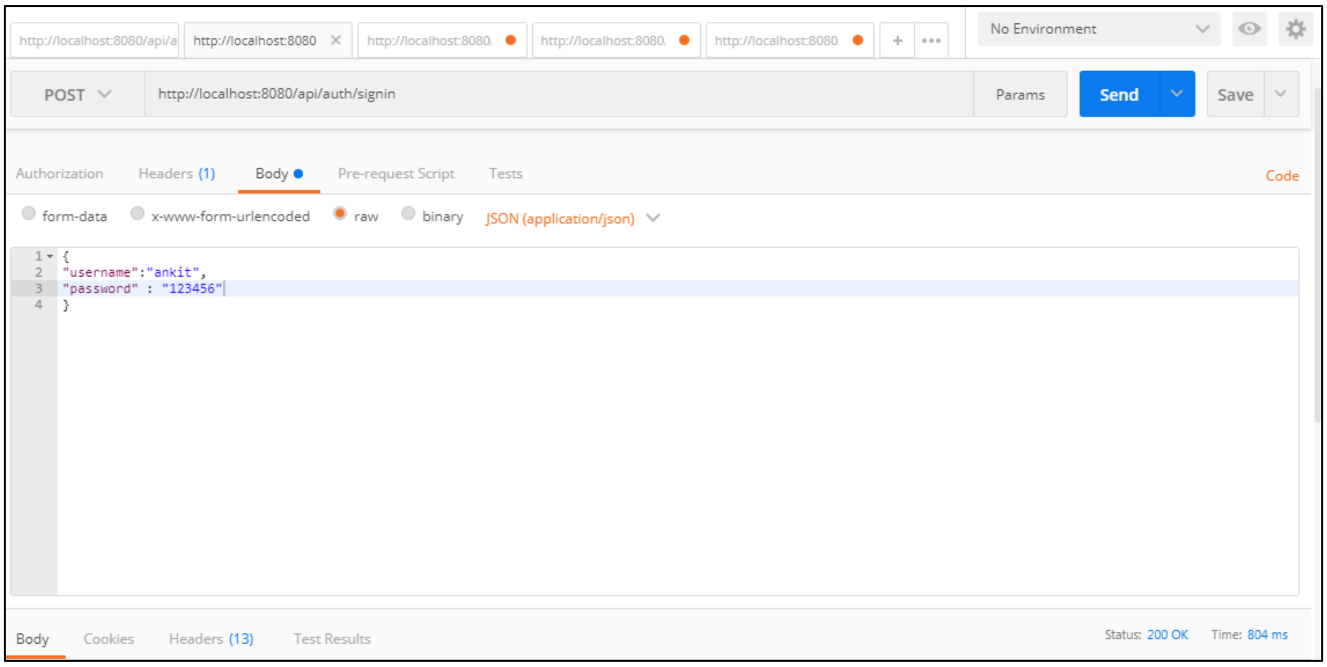
Status: 200 OK Time: 1054 ms

Figure 20: SIGNUP POST REQUEST



```
{
    "message": "User registered successfully!"
}
```

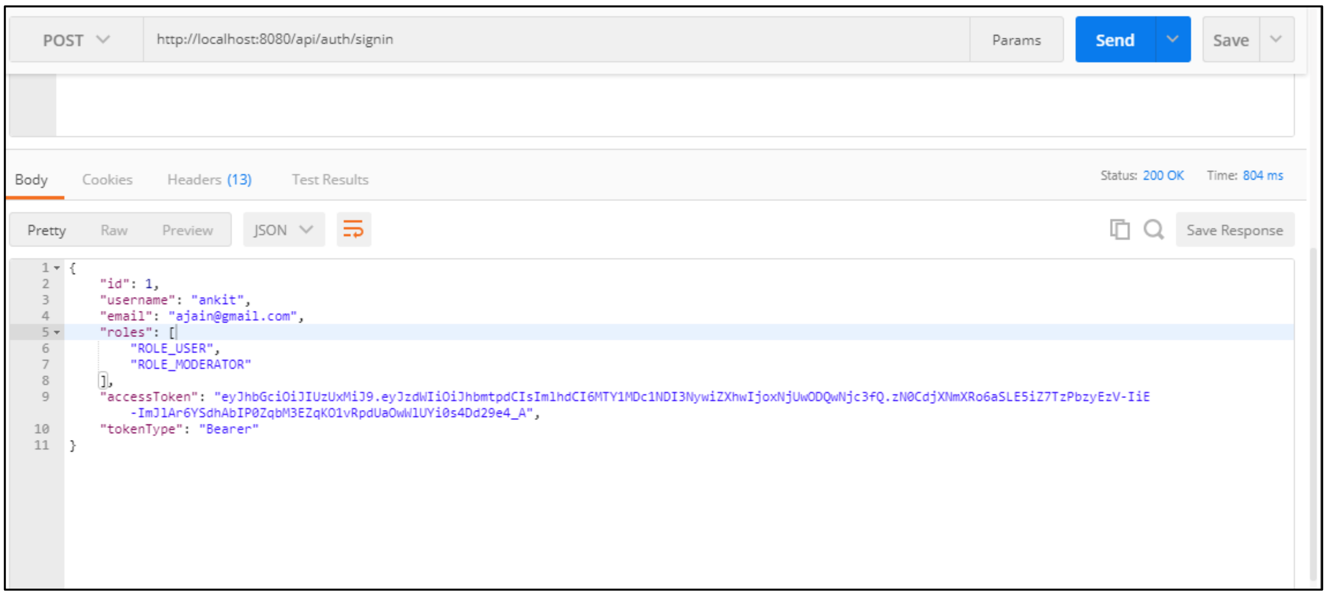
Figure 21: SIGNUP POST RESPONSE



POST http://localhost:8080/api/auth/signin

```
{
"username":"ankit",
"password" : "123456"
}
```

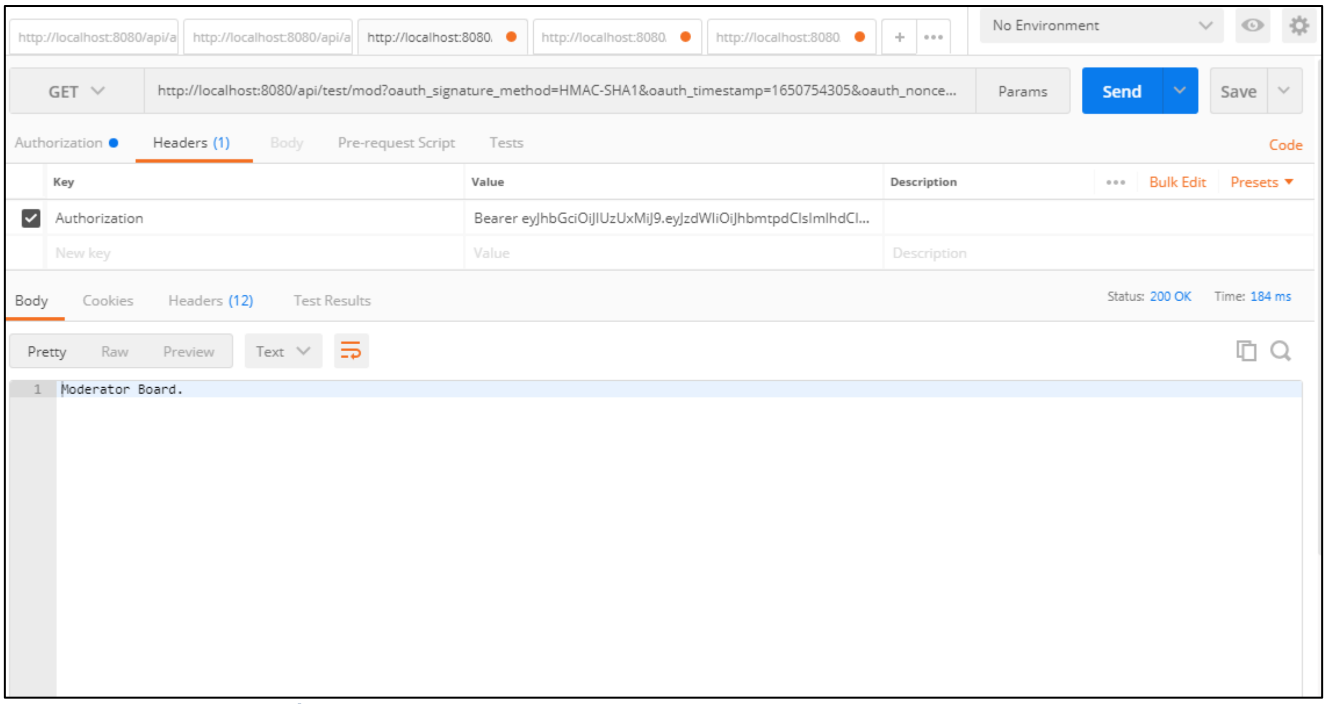
Figure 22: SIGNIN POST REQUEST



POST http://localhost:8080/api/auth/signin

```
{
    "id": 1,
    "username": "ankit",
    "email": "ajain@gmail.com",
    "roles": [
        "ROLE_USER",
        "ROLE_MODERATOR"
    ],
    "accessToken": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhbmtpdCIsImlhdCI6MTY1MDc1NDI3NywiZXhwIjoxNjUwODQwNjc3fQ.zN0CdjXNmXRo6aSLE5iZ7TzPbzyEzV-IiE
        -ImJ1Ar6YSdhAbIP0ZqbM3EZqKO1vRpdUaOwWlUYi0s4Dd29e4_A",
    "tokenType": "Bearer"
}
```

Figure 23: SIGNIN POST RESPONSE



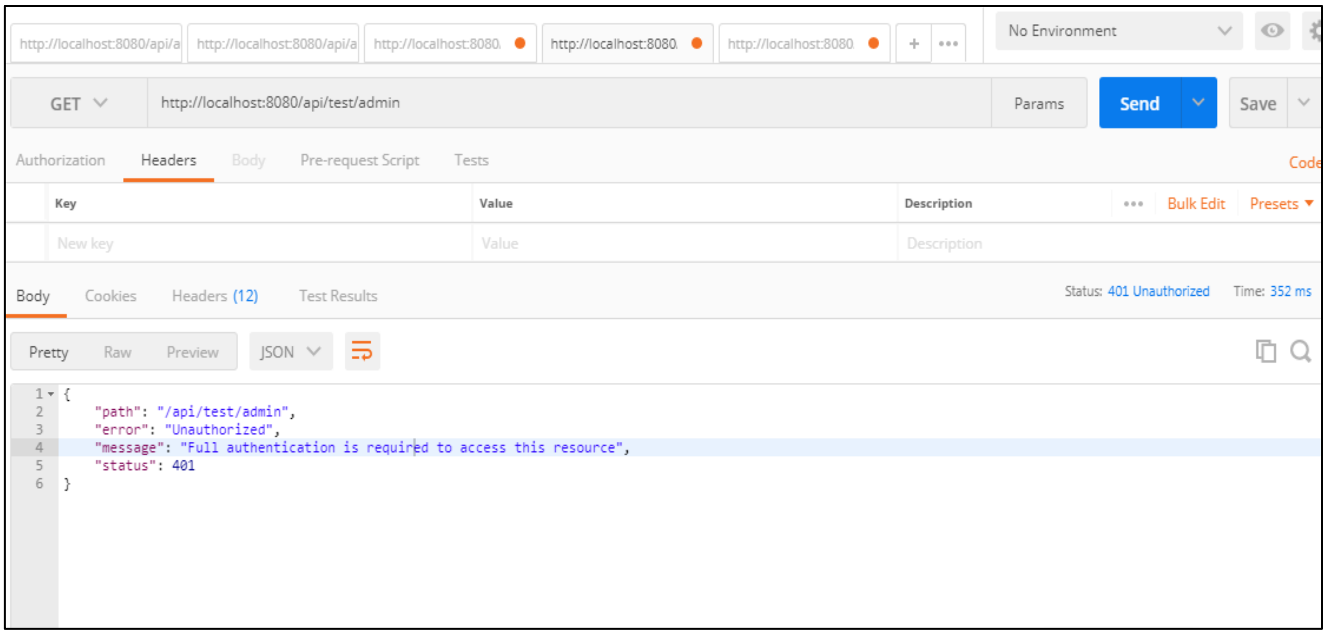
GET http://localhost:8080/api/test/mod?oauth_signature_method=HMAC-SHA1&oauth_timestamp=1650754305&oauth_nonce...

Key	Value	Description
Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhbmtpdCIsImlhdCI...	

Status: 200 OK Time: 184 ms

```
Moderator Board.
```

Figure 24: MODERATOR ACCESSES API USING CORRECT TOKEN

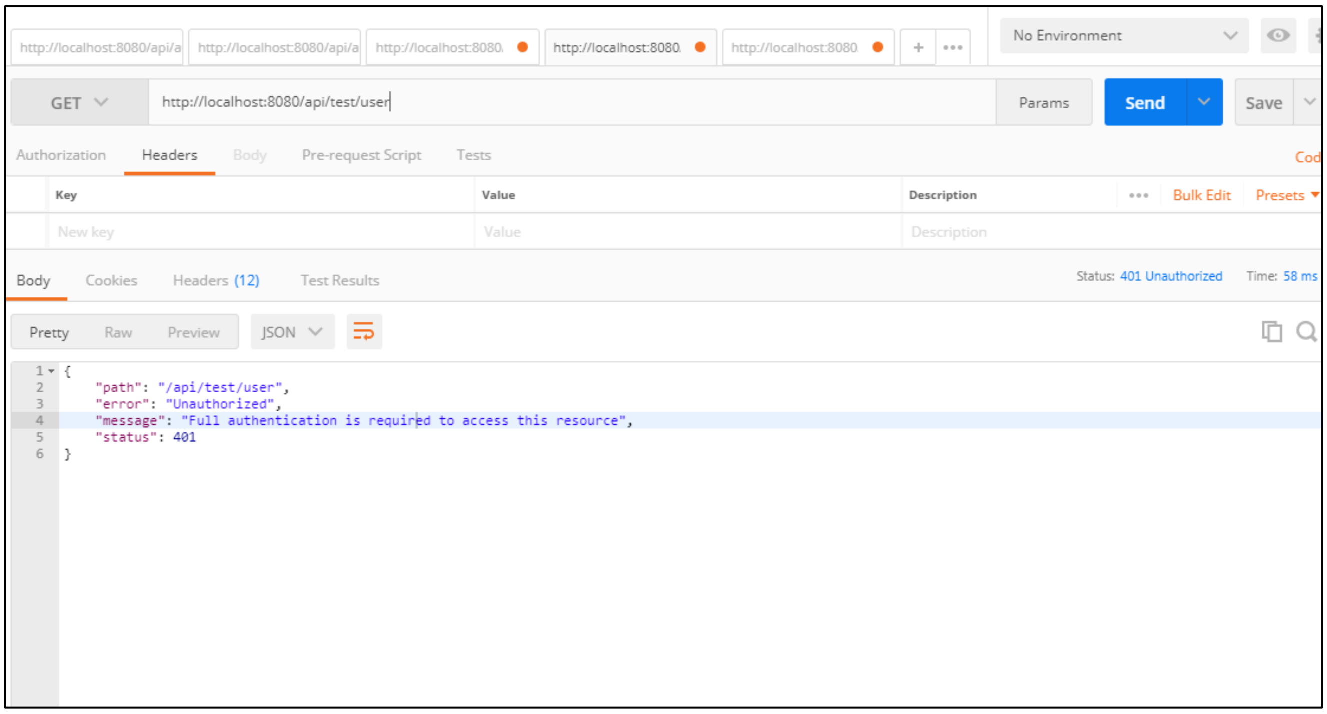


GET http://localhost:8080/api/test/admin

Status: 401 Unauthorized Time: 352 ms

```
{
    "path": "/api/test/admin",
    "error": "Unauthorized",
    "message": "Full authentication is required to access this resource",
    "status": 401
}
```

Figure 25: ADMIN API WITH NO ACCESS TOKEN RESPONSE

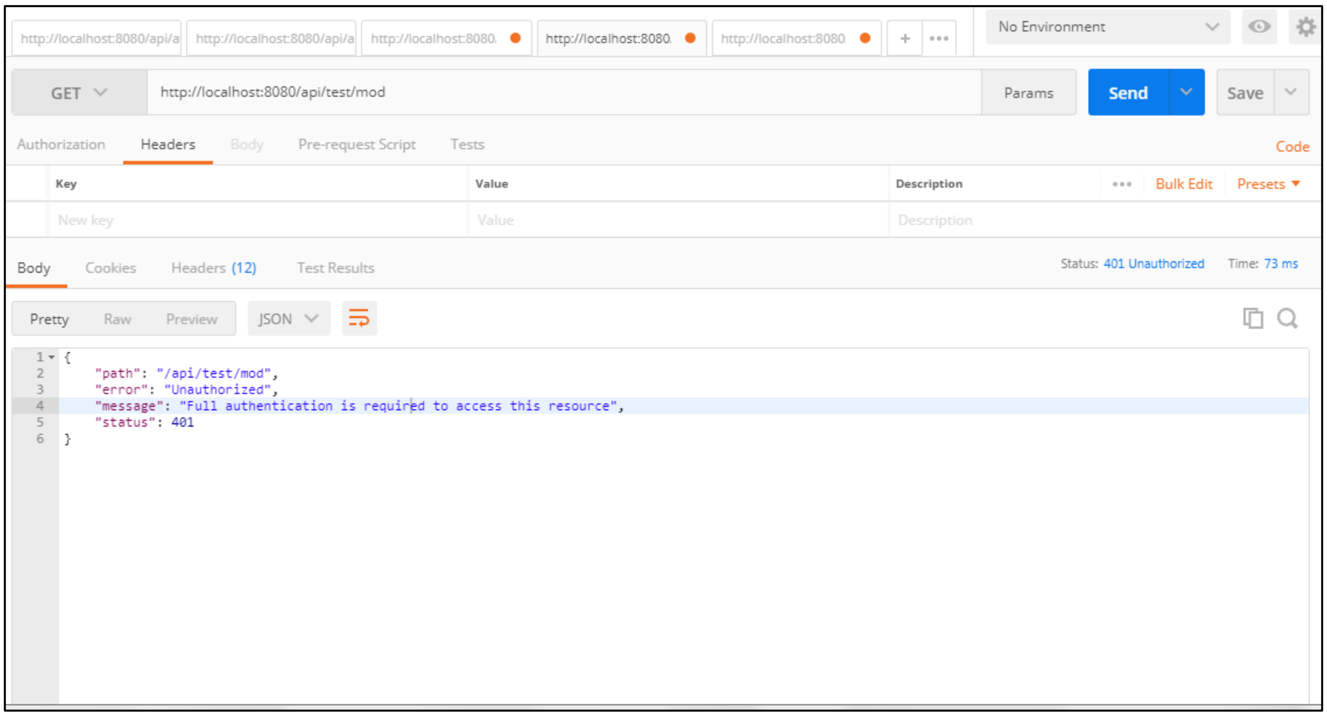


http://localhost:8080/api/test/user

Status: 401 Unauthorized Time: 58 ms

```
{
    "path": "/api/test/user",
    "error": "Unauthorized",
    "message": "Full authentication is required to access this resource",
    "status": 401
}
```

Figure 26: USER API WITH NO ACCESS TOKEN RESPONSE

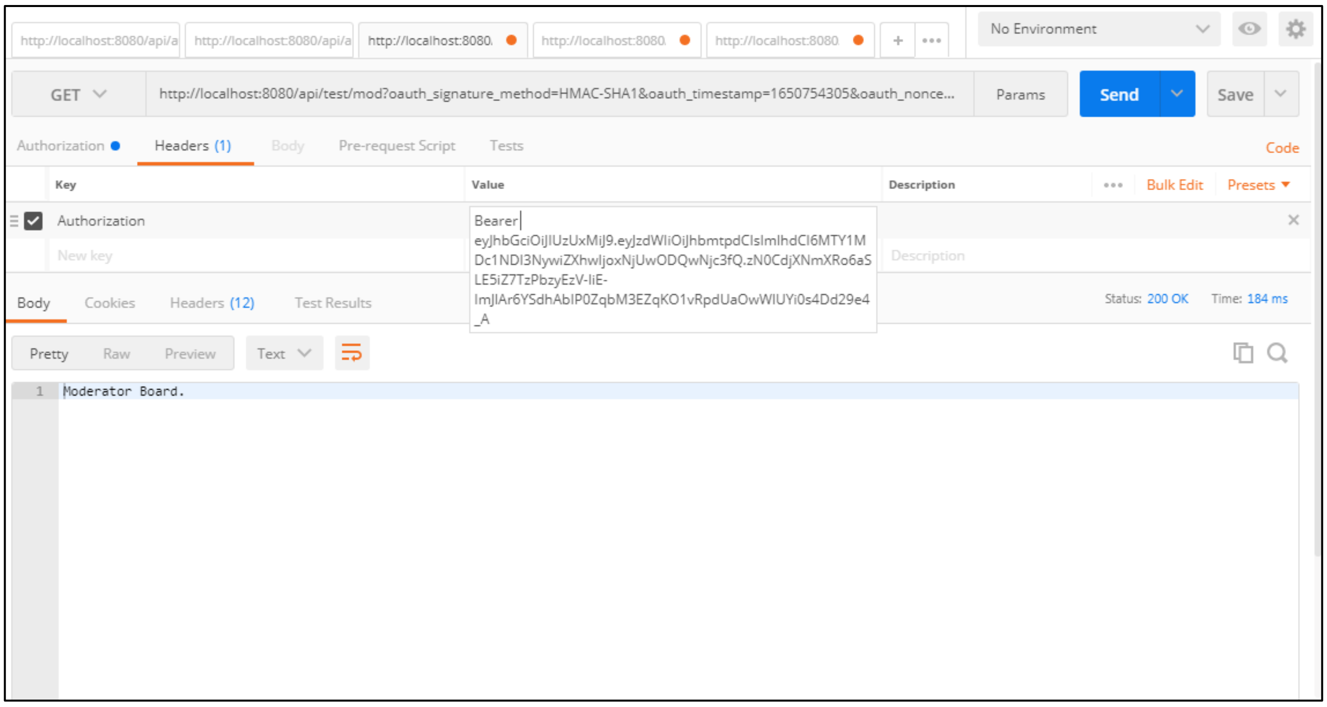


http://localhost:8080/api/test/mod

Status: 401 Unauthorized Time: 73 ms

```
{
    "path": "/api/test/mod",
    "error": "Unauthorized",
    "message": "Full authentication is required to access this resource",
    "status": 401
}
```

Figure 27: MODERATOR API WITH NO ACCESS TOKEN RESPONSE



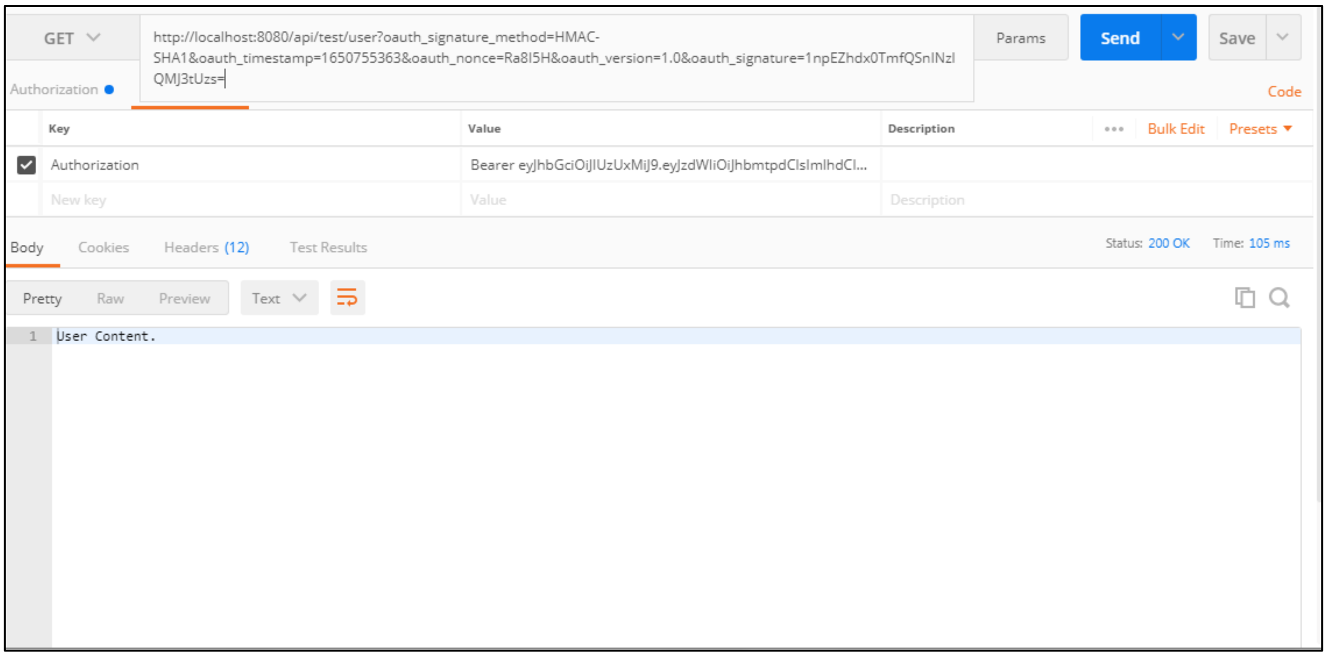
```
http://localhost:8080/api/test/mod?oauth_signature_method=HMAC-SHA1&oauth_timestamp=1650754305&oauth_nonce...
```

Key	Value	Description
Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhbmtpdCIsImlhdCI6MTY1MDc1NDI3NywiZXhwIjoxNjUwODQwNjc3fQ.zN0CdjXNmXRo6aSLE5iZ7TzPbzyEzV-liE-ImJlAr6YSdhAbIP0ZqbM3EZqKO1vRpdUaOwWIUYi0s4Dd29e4_A	

Status: 200 OK Time: 184 ms

```
Moderator Board.
```

Figure 28: MODERATOR API WITH CORRECT BEARER TOKEN



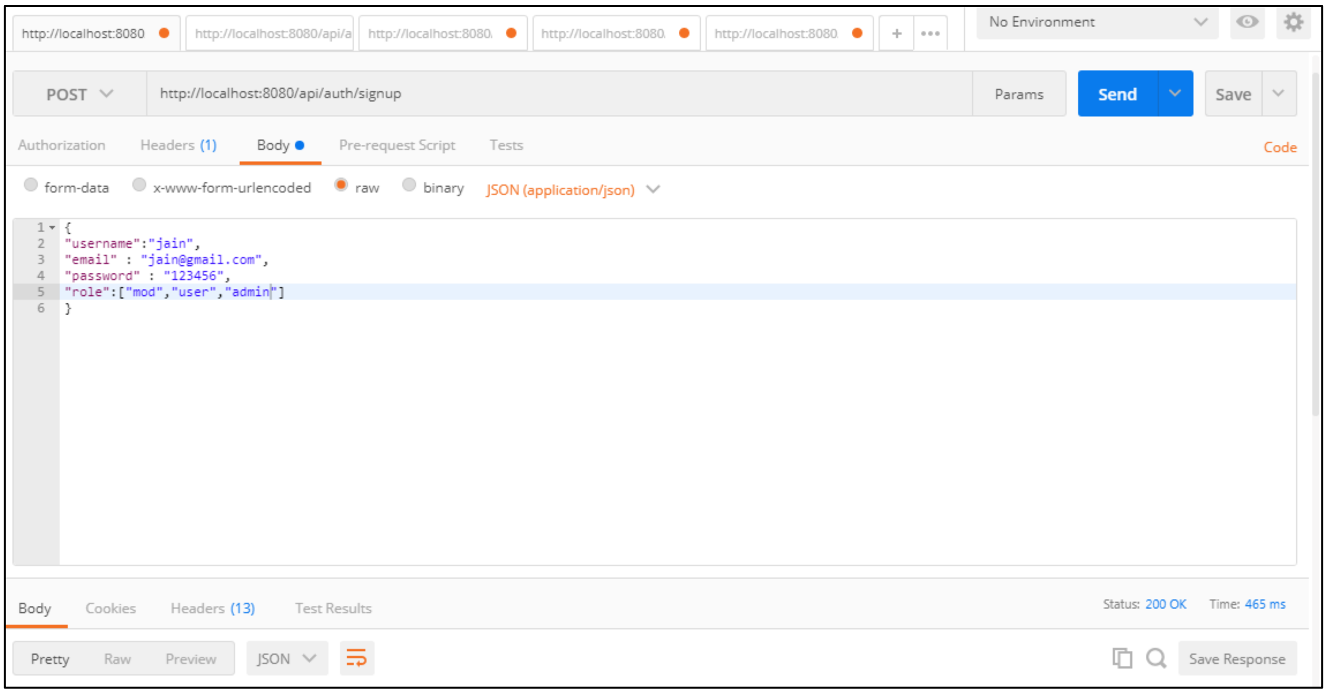
```
http://localhost:8080/api/test/user?oauth_signature_method=HMAC-SHA1&oauth_timestamp=1650755363&oauth_nonce=Ra8I5H&oauth_version=1.0&oauth_signature=1npEZhdx0TmfQSnINzlQMJ3tUzs=
```

Key	Value	Description
Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhbmtpdCIsImlhdCI...	

Status: 200 OK Time: 105 ms

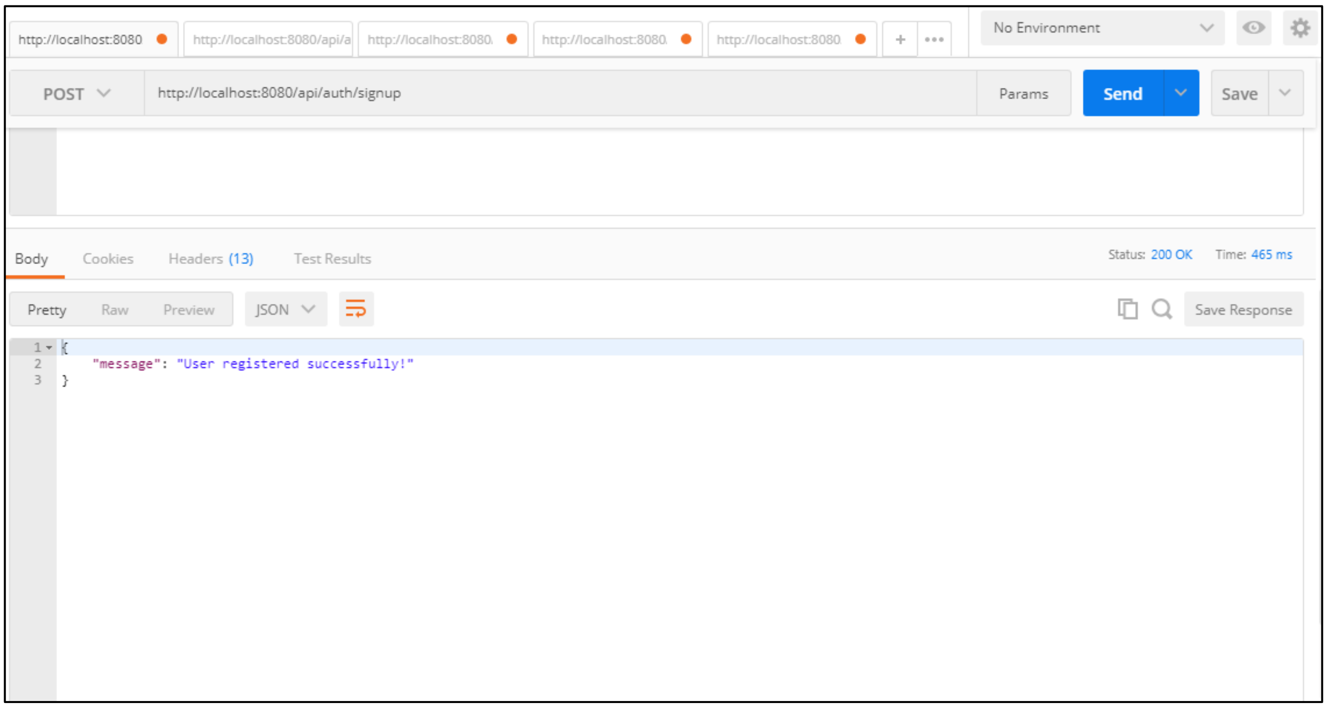
```
User Content.
```

Figure 29: USER API WITH CORRECT BEARER TOKEN



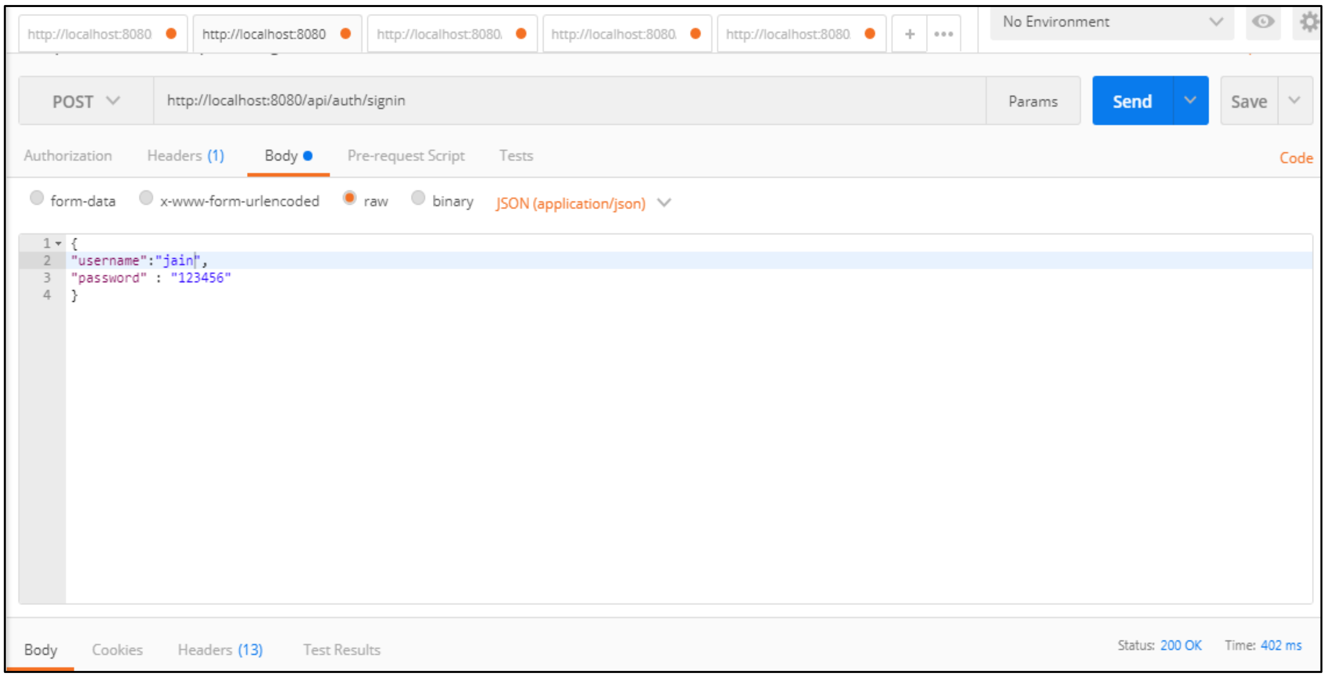
```
{
"username":"jain",
"email" : "jain@gmail.com",
"password" : "123456",
"role":["mod","user","admin"]
}
```

Figure 30: SIGNUP OF A USER WITH ALL THREE ROLES REQUEST



```
{
    "message": "User registered successfully!"
}
```

Figure 31: SIGNUP RESPONSE

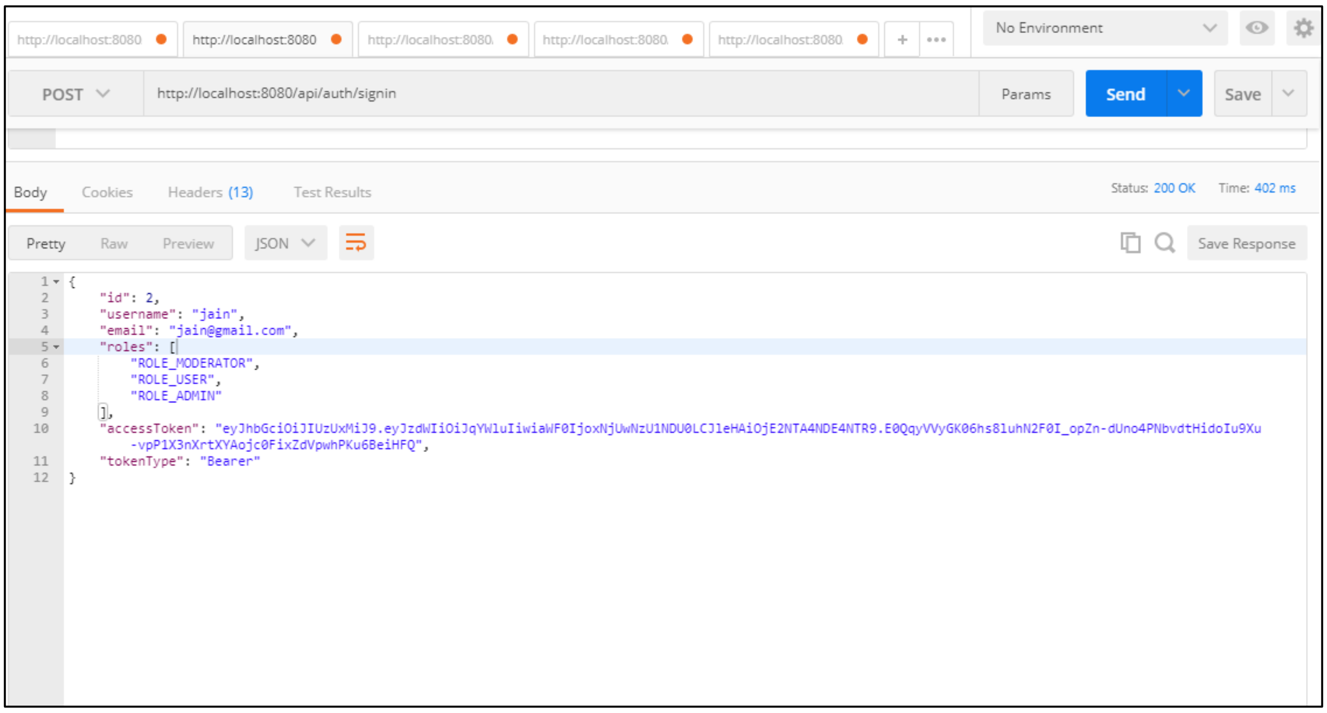


POST http://localhost:8080/api/auth/signin

```
{
"username":"jain",
"password" : "123456"
}
```

Status: 200 OK Time: 402 ms

Figure 32: SIGN IN POST REQUEST

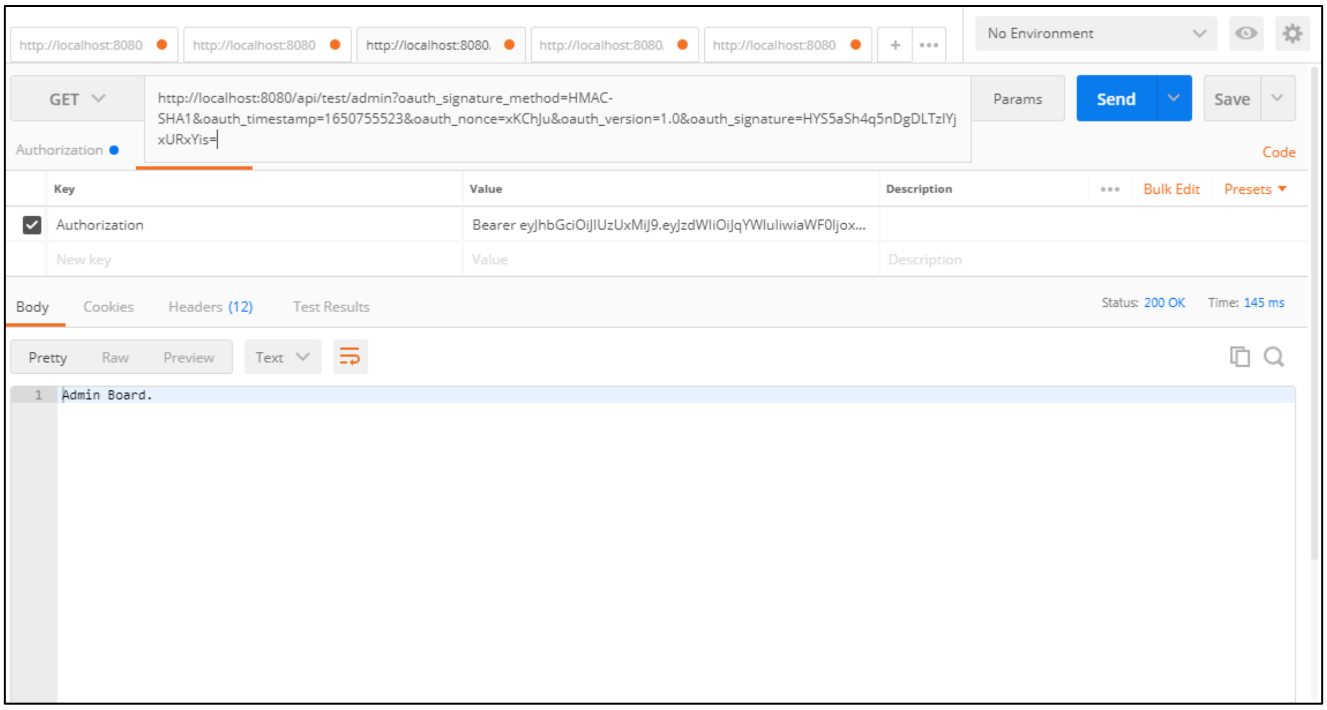


POST http://localhost:8080/api/auth/signin

Status: 200 OK Time: 402 ms

```
{
    "id": 2,
    "username": "jain",
    "email": "jain@gmail.com",
    "roles": [
        "ROLE_MODERATOR",
        "ROLE_USER",
        "ROLE_ADMIN"
    ],
    "accessToken": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJqYWluIiwiaWF0IjoxNjUwNzU1NDU0LCJleHAiOjE2NTA4NDE4NTR9.E0QqyVVyGK06hs8luhN2F0I_opZn-dUno4PNbvdtHidoIu9Xu
        -vpP1X3nXrtXYAojc0FixZdVpwhPKu6BeiHFQ",
    "tokenType": "Bearer"
}
```

Figure 33: SIGN IN POST RESPONSE



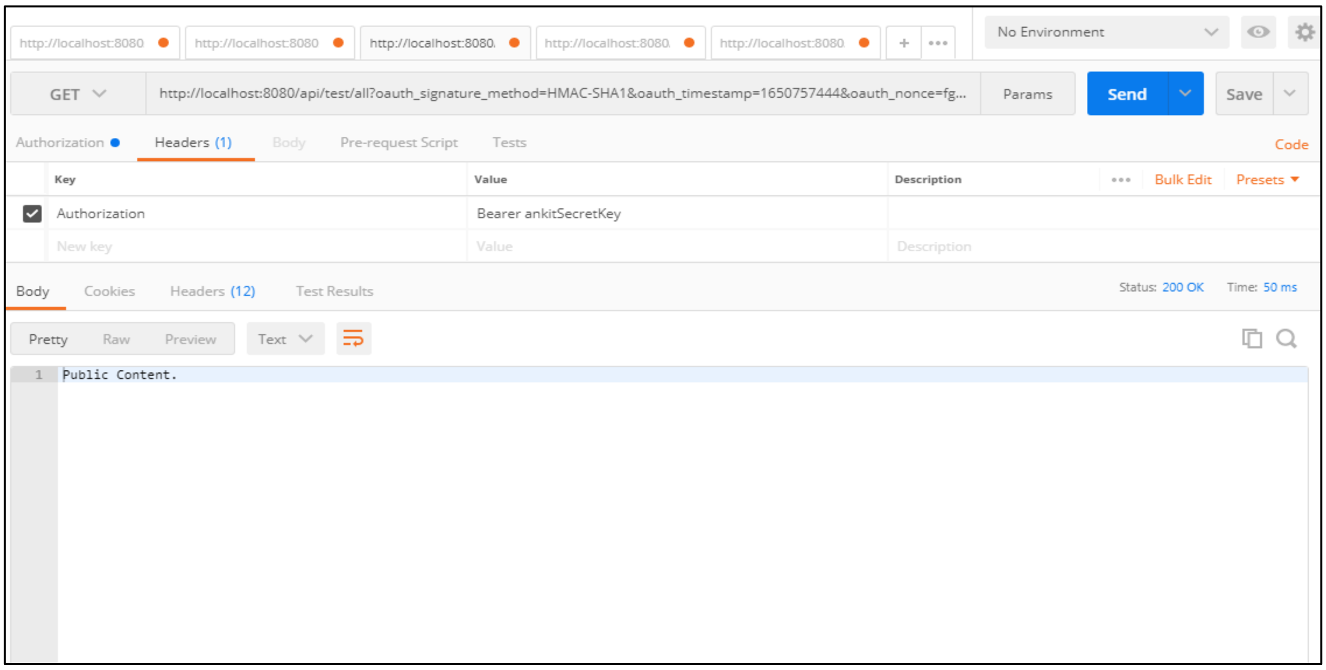
GET http://localhost:8080/api/test/admin?oauth_signature_method=HMAC-SHA1&oauth_timestamp=1650755523&oauth_nonce=xKChJu&oauth_version=1.0&oauth_signature=HYS5aSh4q5nDgDLTzlYjxURxYis=

	Key	Value	Description
✓	Authorization	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJqYWluIiwiaWF0Ijox...	

Status: 200 OK Time: 145 ms

```
Admin Board.
```

Figure 34: ADMIN BOARD WITH ACCESS



GET http://localhost:8080/api/test/all?oauth_signature_method=HMAC-SHA1&oauth_timestamp=1650757444&oauth_nonce=fg...

	Key	Value	Description
✓	Authorization	Bearer ankitSecretKey	

Status: 200 OK Time: 50 ms

```
Public Content.
```

Figure 35: INTERCEPTOR BASED SECURITY USING BEARER TOKEN

9. CONCLUSION

The research covered authentication and authorization of microservice based application which range from using interceptor based security to using Identity Access Management system such as google firebase. The research covered role based authentication access then shifted towards enhancing security through OAuth2 and JWT tokens which are digitally signed and encrypted . The purpose of research was to analyse different solution for securing microservice based application and it was thoroughly achieved.

10. REFERENCES

- https://medium.com/@akhileshanand/spring-boot-api-security-with-jwt-and-role-based-authorization-fea1fd7c9e32
- https://elang2.github.io/myblog/posts/2018-09-29-Role-Based-Access-Control-MicroServices.html#:~:text=RBAC%20stands%20for%20Role%20Based,users%2C%20vendors%20and%20customers%20efficient.
- https://www.devglan.com/spring-security/spring-boot-security-oauth2-example
- https://walkingtree.tech/securing-microservices-oauth2/
- https://medium.com/capgemini-norway/oauth2-authorization-patterns-and-microservices-45ffc67a8541
- https://konghq.com/blog/jwt-kong-gateway
- https://nirajsonawane.github.io/2019/07/14/Securing-Spring-Boot-Microservices-Using-JWT-Token/#:~:text=JSON%20Web%20Tokens%20are%20an,JWT%20signature%20should%20be%20computed
- https://firebase.google.com/docs/auth
- https://developer.flowroute.com/quickstarts/two-factor-authentication/#:~:text=SMS%20Two%2Dfactor%20Authentication%20is,allow%20access%20to%20that%20application.
- https://openliberty.io/docs/latest/single-sign-on.html
- https://frontegg.com/blog/authentication-in-microservices#:~:text=Single%20Sign%2DOn%20(SSO)&text=In%20the%20context%20of%20a,to%20log%20into%20your%20app.
- https://medium.com/@sureshdsk/how-json-web-token-jwt-authentication-works-585c4f076033
- https://www.tutorialspoint.com/spring_boot/spring_boot_oauth2_with_jwt.htm#:~:text=OAuth2%20is%20an%20authorization%20framework,Client%20ID%20and%20Client%20secrethttps://spring.io/guides/tutorials/spring-boot-oauth2/
- https://medium.com/capgemini-norway/oauth2-authorization-patterns-and-microservices-45ffc67a8541
- https://medium.com/geekculture/role-based-access-control-rbac-with-spring-boot-and-jwt-bc20a8c51c15

Checklist of Items for the Final Dissertation / Project / Project Work Report

1.	**Is the final report neatly formatted with all the elements required for a technical Report?**	Yes / ~~No~~
2.	Is the Cover page in proper format as given in Annexure A?	Yes / ~~No~~
3.	Is the Title page (Inner cover page) in proper format?	Yes / ~~No~~
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	Yes / ~~No~~ Yes / ~~No~~
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	Yes / ~~No~~ Yes / ~~No~~
6.	Is the title of your report appropriate? **The title should be adequately descriptive, precise and must reflect scope of the actual work done.** Uncommon abbreviations / Acronyms should not be used in the title	Yes / ~~No~~
7.	Have you included the List of abbreviations / Acronyms?	~~Yes~~ / No
8.	Does the Report contain a summary of the literature survey?	~~Yes~~ / No
9.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (Ch. 1 should start on Page # 1) (ii). Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) (iii). Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly? Are their titles appropriate	Yes / ~~No~~ Yes / ~~No~~ Yes / ~~No~~ Yes / ~~No~~ Yes / ~~No~~ Yes / ~~No~~
10.	Is the conclusion of the Report based on discussion of the work?	Yes / ~~No~~
11.	Are References or Bibliography given at the end of the Report? Have the References been cited properly inside the text of the Report? Are all the references cited in the body of the report	Yes / ~~No~~ Yes / ~~No~~ Yes / ~~No~~
12.	Is the report format and content according to the guidelines? The report should not be a mere printout of a Power Point Presentation, or a user manual. Source code of software need not be included in the report.	Yes / ~~No~~

Declaration by Student:

I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

[illegible]

Signature of the Student

Place: Delhi

Date: 24th April, 2022

Name: ANKIT JAIN

ID No.: 2020MT93155