Published in Javarevisited

C. R. Raja Vignesh Follow

May 14, 2022 · 8 min read · Listen

Save

# Spring Boot Authorization: Creating an Authorization Server for your Microservices

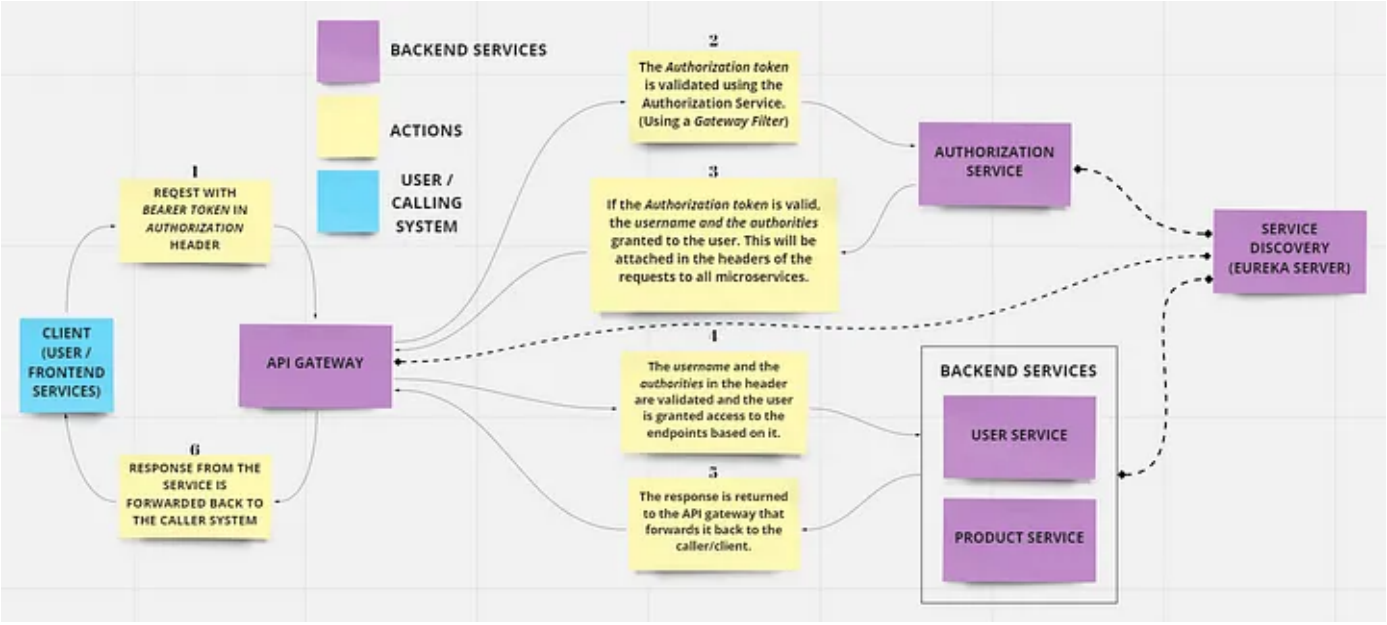
This article explains in detail about implementing an Authentication mechanism using a centralized Authorization Server and an API gateway.

### What is a JWT Token and Why Use It?

> JSON Web Token is a proposed Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a public/private key.

They can be used to achieve stateless authentication. They can be shared across the instances or multiple services and can contain the details needed to authenticate them. Thus, there is no need to setup separate resources to maintain session or store the token/session details in a separate database/cache.

### Architecture Used

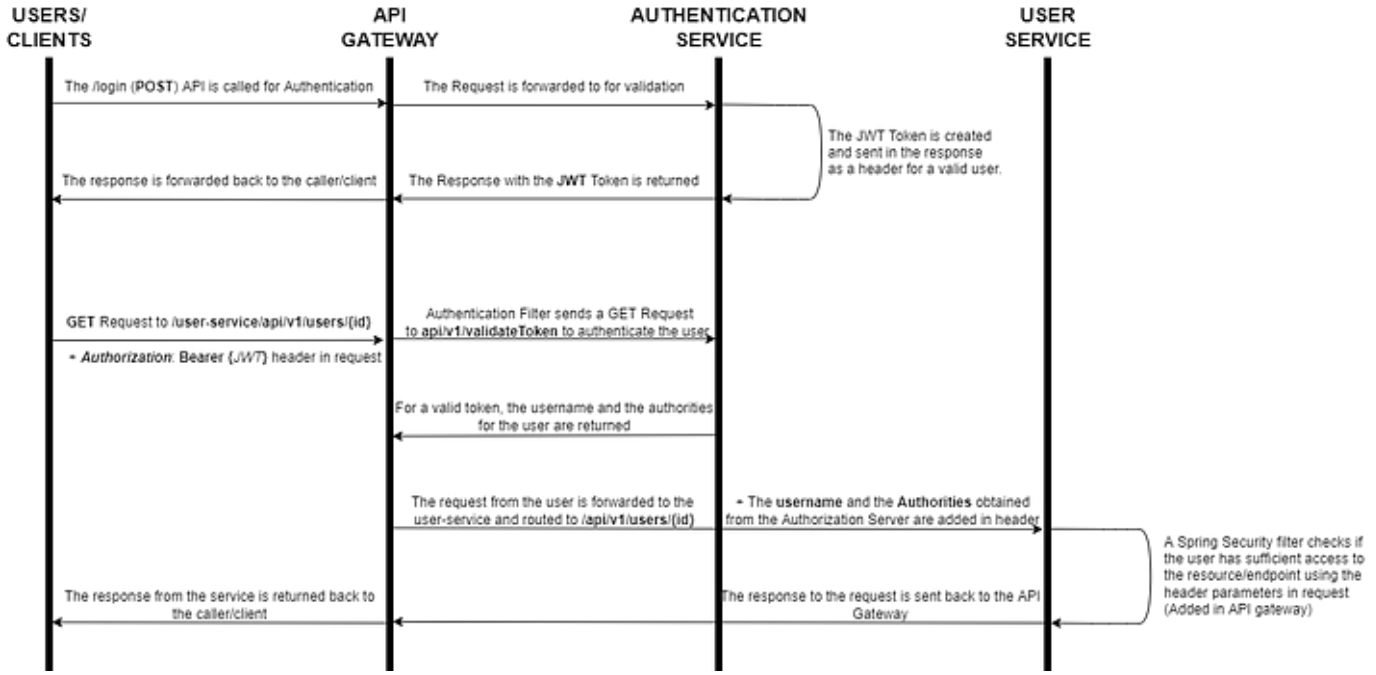



The General Flow of the request to Secured resources

The general design used is:

- A centralized Authorization Server that will be used for creating and validating the JWT tokens.
- The API Gateway will act as the point of entry for the application. This will route the requests to the corresponding microservices.
- A Gateway filter will be added to the routes for validating the JWT tokens in the request for secured resources. This will make an API call to the Authorization Server to validate the token and get the username and authorities to the user. These details will be forwarded to the downstream services in the Header part for validations.
- The Eureka Discovery Client will be used for Service discovery.

### Authentication Flow




The general flow for the services is:

- The user user can login/create an authentication token by calling the **/login** (**POST**) endpoint with the username and password. This will return the Bearer token as a header parameter.
- Get the authorization token from the from the response header. This should passed as the value for the ***Authorization*** header in the format **Bearer** ***access_token*** for requests to secured resources.
- For the secured resources, the custom ***Gateway Filter (*****AuthenticationPrefilter***)* will be called. This will do an API call to the **/api/v1/validateToken** Endpoint in the Authentication Service which validates the token and sends the username and authorities granted to the user in response in case of successful validation.
- If the token is valid, the username and authorities to the user is appended to the header request before being forwarded to the resource requested by the user.
- The other microservices (For eg: user-service) will have an authorization filter extended from **OncePerRequestFilter** in **spring-security** that will create an **Authentication** object using the **UsernamePasswordAuthenticationToken** class (with *username* and *SimpleGrantedAuthority* from header as input with password as null).

- If the user has authority/access to a resource, then the request is allowed. Else, 401 Unathorized / 403 Forbidden response is returned to the client.

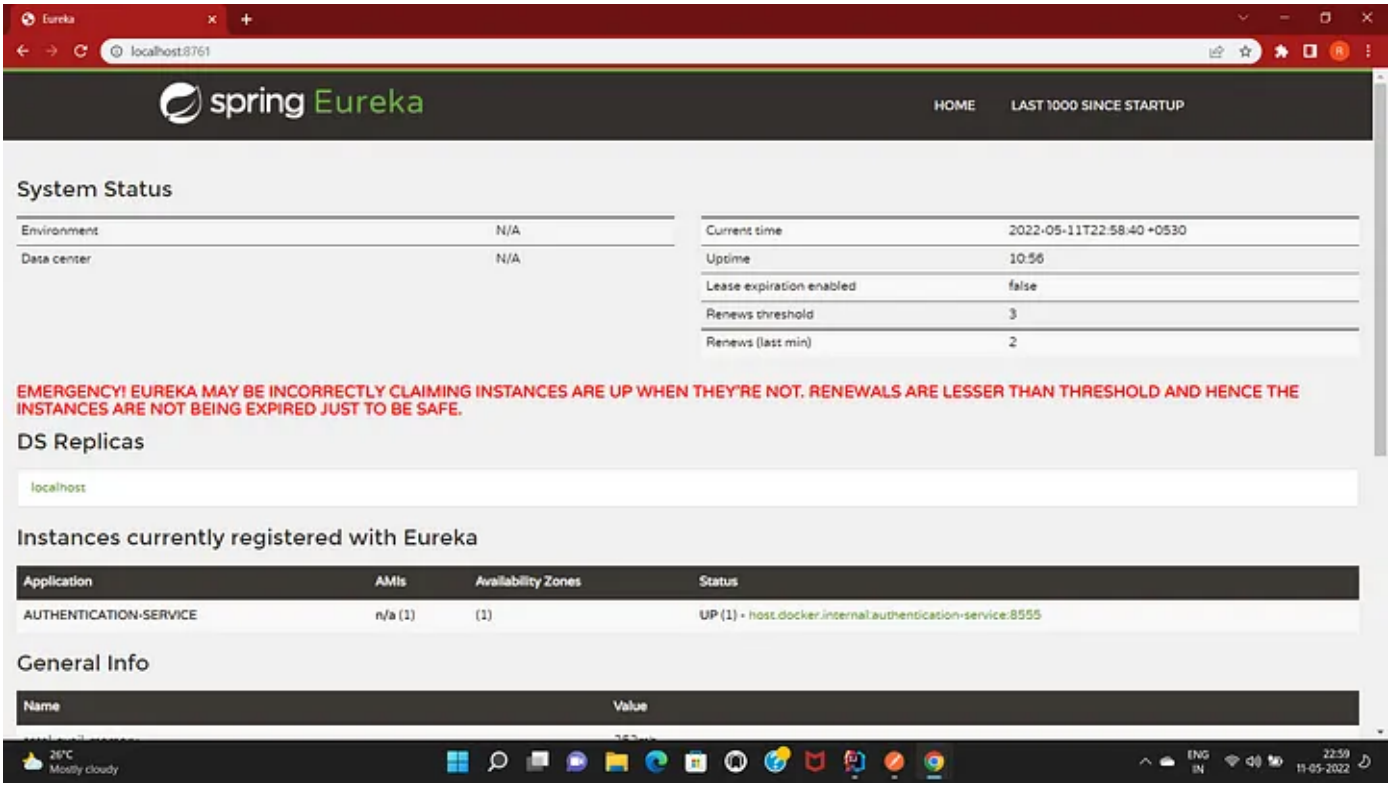
## Setting Up the services:

### Eureka Server

- Create a Spring Boot application using the **Spring initializr** with the spring-cloud-starter-netflix-eureka-server dependency in the pom file. The spring-cloud-dependencies also has to be added under dependency management for support.
- Now, application can be configured to start up as a Eureka Server by adding the **@EnableEurekaServer** annotation to the main class of the application.
- The bellow properties are added to the properties file for supporting the Eureka Server.

```
spring.application.name=naming-server
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.instance.prefer-ip-address=true
```

- Now, the Eureka Server can be accessed in **http://localhost:8761/** after startup. It will display a list of the services registered to it.




### Authorization Service

- Create a Spring Boot Application *spring-boot-starter-security, spring-boot-starter-web, spring-cloud-starter-sleuth, spring-cloud-starter-config, spring-cloud-starter-netflix-eureka-client, spring-boot-starter-data-jpa, spring-boot-starter-data-mongodb, spring-boot-starter-data-redis* and *lombok* dependencies.

- The *spring-boot-starter-security* is needed for the Authorization and Authentication purposes while **spring-boot-starter-data-mongodb** and **spring-boot-starter-data-jpa** are needed for accessing the credentials in mongo DB collections. The dependency *io.jsonwebtoken:jjwt* is need for creating and validating JWT tokens.

- For Enabling the authentication using the credentials in the database, a custom implementation of the **UserDetailsService** class from Spring Security has to be provided. This will implement the *loadUserByUsername()* method which will be used to fetch the user credentials from the database and return an instance of **UserDetails** (from spring security).

```
@Service
public class ApplicationUserDetailsService implements
UserDetailsService {
```

```
    @Autowired
    private UsersService usersService;


    @Override
    public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {
        return new
ApplicationUsers(usersService.getByUsrName(s).orElseThrow(() -> new
UsernameNotFoundException("Username Not Found")));
    }
}
```

- A custom implementation of the **UserDetails** class is created to map the custom database objects to the format required by spring security.
- A new Configuration class is created extending the **WebSecurityConfigurerAdapter** class in spring security.

```java
package com.infotrends.in.authenticationserver.security.config;

import com.infotrends.in.authenticationserver.security.filters.JWTAuthenticationFilter;
import com.infotrends.in.authenticationserver.security.filters.JWTVerifierFilter;
import com.infotrends.in.authenticationserver.security.services.ApplicationUserDetailsService;
import com.infotrends.in.authenticationserver.services.redis.TokensRedisService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManage
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapt
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {


    @Autowired
    private PasswordEncoder encoder;

    @Autowired
    private ApplicationUserDetailsService applicationUserDetailsService;

    @Autowired
    private TokensRedisService redisService;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .addFilter(new JWTAuthenticationFilter(authenticationManager(), redisService))
                .addFilterAfter(new JWTVerifierFilter(redisService), JWTAuthenticationFilter.class
                .authorizeRequests()
                .antMatchers("/api/v1/validateConnection/whitelisted").permitAll()
                .anyRequest()
                .authenticated()
                .and().httpBasic();
    }

```

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(authenticationProvider());
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
        authenticationProvider.setPasswordEncoder(encoder);
        authenticationProvider.setUserDetailsService(applicationUserDetailsService);

        return authenticationProvider;
    }
}
```

SpringSecurityConfig.java hosted with ❤ by GitHub view raw

***successfulAuthentication()*** to create the JWT token in case of successful Authorization.

```java
package com.infotrends.in.authenticationserver.security.filters;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.infotrends.in.InfoTrendsIn.security.SecurityConstants;
import com.infotrends.in.authenticationserver.model.ConnValidationResponse;
import com.infotrends.in.authenticationserver.model.JwtAuthenticationModel;
import com.infotrends.in.authenticationserver.model.redis.TokensEntity;
import com.infotrends.in.authenticationserver.services.redis.TokensRedisService;
import com.infotrends.in.authenticationserver.utils.Utilities;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.util.Date;

@Slf4j
@RequiredArgsConstructor
public class JWTAuthenticationFilter extends UsernamePasswordAuthenticationFilter {

    private final AuthenticationManager authenticationManager;
    private ObjectMapper mapper=new ObjectMapper();

    private final TokensRedisService tokensRedisService;

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse re
        try {
            JwtAuthenticationModel authModel = mapper.readValue(request.getInputStream(), JwtAuthe
            Authentication authentication = new UsernamePasswordAuthenticationToken(authModel.getU
            return authenticationManager.authenticate(authentication);
```

```
            return authenticationManager.authenticate(authentication);

        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse respon
        String token = Jwts.builder()
                .setSubject(authResult.getName())
                .claim("authorities", authResult.getAuthorities())
                .claim("principal", authResult.getPrincipal())
                .setIssuedAt(new Date())
                .setIssuer(SecurityConstants.ISSUER)
                .setExpiration(Date.from(LocalDateTime.now().plusMinutes(30).toInstant(ZoneOffset.
                .signWith(SignatureAlgorithm.HS256, SecurityConstants.KEY)
                .compact();

        log.info(token);
        TokensEntity tokensEntity = TokensEntity.builder().id(Utilities.generateUuid()).authentica
                        .username(authResult.getName())
                        .createdBy("SYSTEM").createdOn(LocalDateTime.now())
                        .modifiedBy("SYSTEM").modifiedOn(LocalDateTime.now())
                        .build();
        tokensEntity = tokensRedisService.save(tokensEntity);
        response.addHeader(SecurityConstants.HEADER, String.format("Bearer %s", tokensEntity.getId
//        response.addHeader("Expiration", String.valueOf(30*60));

        ConnValidationResponse respModel = ConnValidationResponse.builder().isAuthenticated(true).
        response.addHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE);
        response.getOutputStream().write(mapper.writeValueAsBytes(respModel));
    }
}
```

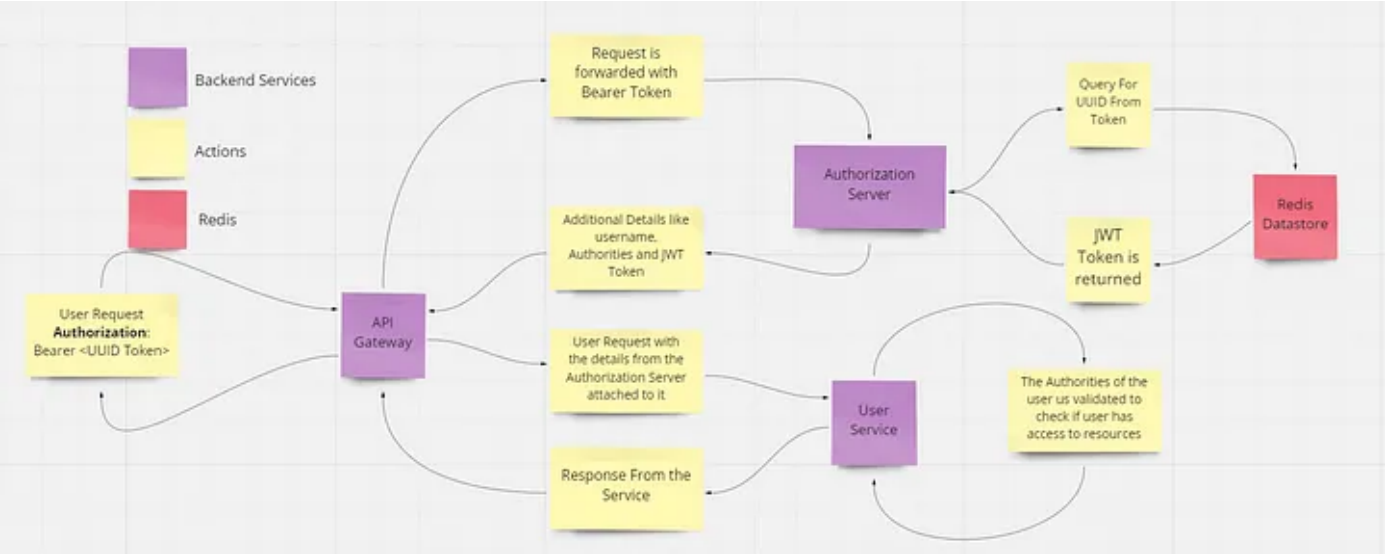
JWTAuthenticationFilter.java hosted with ❤ by GitHub view raw

```java
package com.infotrends.in.authenticationserver.security.filters;

import com.infotrends.in.InfoTrendsIn.security.SecurityConstants;
import com.infotrends.in.authenticationserver.model.redis.TokensEntity;
import com.infotrends.in.authenticationserver.services.redis.TokensRedisService;
import com.infotrends.in.authenticationserver.utils.Utilities;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jws;
import io.jsonwebtoken.Jwts;
import lombok.RequiredArgsConstructor;
import org.apache.tomcat.util.http.parser.Authorization;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContext;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.Set;
import java.util.stream.Collectors;

@RequiredArgsConstructor
public class JWTVerifierFilter extends OncePerRequestFilter {

    private final TokensRedisService tokensRedisService;

    @Override
    protected void doFilterInternal(HttpServletRequest httpServletRequest, HttpServletResponse htt
        String bearerToken = httpServletRequest.getHeader(SecurityConstants.HEADER);
        if(!(Utilities.validString(bearerToken) && bearerToken.startsWith(SecurityConstants.PREFIX
            filterChain.doFilter(httpServletRequest, httpServletResponse);
            return;
        }

        String authToken = bearerToken.replace(SecurityConstants.PREFIX, "");
```

```java
package com.infotrends.in.authenticationserver.resources;

import com.sun.security.auth.UserPrincipal;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest;
import java.util.List;

@RestController
@RequestMapping("/api/v1/validateToken")
public class ConnectionValidatorResource {

    @GetMapping(value = "", produces = {MediaType.APPLICATION_JSON_VALUE})
    public ResponseEntity<ConnValidationResponse> validateGet(HttpServletRequest request) {
        String username = (String) request.getAttribute("username");
        List<GrantedAuthority> grantedAuthorities = (List<GrantedAuthority>) request.getAttribute(
        return ResponseEntity.ok(ConnValidationResponse.builder().status("OK").methodType(HttpMeth
                        .username(username).authorities(grantedAuthorities)
                .isAuthenticated(true).build());
    }

    @Getter
    @Builder
    @ToString
    public class ConnValidationResponse {
        private String status;
        private boolean isAuthenticated;
        private String methodType;
        private String username;
        private List<GrantedAuthority> authorities;
    }

}
```

ConnectionValidatorResource.java hosted with ❤ by GitHub view raw

## Additional Feature:




The additional process involved for generating UUID and storing the token

In general, the content of the JWT token cannot be modified once created. However, it can be easily decrypted and the content can be read by anyone.

Thus, instead of fully returning the JWT token to the user, we can instead return a random UUID generated for that Authentication request and store reference to them in a redis cache.

As such for the client facing application, the UUID generated would be returned and used while the JWT token can be shared between the services for Authorization/Authentication purposes.

```java
package com.infotrends.in.authenticationserver.model.redis;

import lombok.*;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.data.redis.core.RedisHash;

import java.time.LocalDateTime;

@RedisHash(value = "Tokens", timeToLive = 86400)
@Getter
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class TokensEntity {


    private String id;

    private String username;
    private String authenticationToken;
    private String modifiedBy;
    private LocalDateTime modifiedOn;
    private String createdBy;
    private LocalDateTime createdOn;
}
```

TokensEntity.java hosted with ❤ by GitHub view raw

### API Gateway

- Create a Springboot application with the spring-cloud-starter-gateway, spring-cloud-starter-config and spring-cloud-starter-netflix-eureka-client dependencies needed to setup an API gateway with an Eureka Client.
- The details for the Cloud config Server and Eureka Server are added to the configuration file.

```
debug: true
logging:
  level:
    org.springframework.cloud.gateway: DEBUG
```

```
    reactor.netty.http.client: DEBUG
server:
  port: '8765'
spring:
  cloud:
    config:
      profile: dev
    gateway:
      discovery.locator.enabled: true
  config:
    import:
optional:configserver:http://clouduser:configserver705!@localhost:8888
  application:
    name: api-gateway
  jackson:
    date-format: yyyy-MM-dd HH:mm:ss
management:
  endpoints:
    web:
      exposure:
        include: '*'
eureka:
  client:
    serviceUrl:
      defaultZone: http://eurekauser:eureka124!@localhost:8761/eureka
  instance:
    prefer-ip-address: 'true'
```

- The **@EnableFeignClients** annotation is added to the main class of the Api Gateway Application so that it will connect to the Configured Eureka Server.
- A Gateway Filter is configured that will validate the Bearer tokens in the requests by calling the **/validateToken** endpoint in the Authorization Server. This class is created by extending the **AbstractGatewayFilterFactory** class provided by the Spring-API Gateway and overriding its ***apply(Config config)*** method which returns an object for **GatewayFilter**.

```
1  package com.infotrends.in.InfoTrendsIn.ApiGateway.filters;
2
3  import com.fasterxml.jackson.core.JsonProcessingException;
4  import com.fasterxml.jackson.core.type.TypeReference;
5  import com.fasterxml.jackson.databind.ObjectMapper;
6  import com.infotrends.in.InfoTrendsIn.ApiGateway.model.Authorities;
7  import com.infotrends.in.InfoTrendsIn.ApiGateway.model.ConnValidationResponse;
8  import com.infotrends.in.InfoTrendsIn.ApiGateway.utils.Utilities;
9  import com.infotrends.in.InfoTrendsIn.exceptions.model.ExceptionResponseModel;
10 import com.infotrends.in.InfoTrendsIn.security.SecurityConstants;
11 import lombok.AllArgsConstructor;
12 import lombok.NoArgsConstructor;
13 import lombok.RequiredArgsConstructor;
14 import lombok.extern.slf4j.Slf4j;
15 import org.springframework.beans.factory.annotation.Autowired;
16 import org.springframework.beans.factory.annotation.Qualifier;
17 import org.springframework.cloud.context.config.annotation.RefreshScope;
18 import org.springframework.cloud.gateway.filter.GatewayFilter;
19 import org.springframework.cloud.gateway.filter.GatewayFilterChain;
20 import org.springframework.cloud.gateway.filter.GlobalFilter;
21 import org.springframework.cloud.gateway.filter.factory.AbstractGatewayFilterFactory;
22 import org.springframework.core.io.buffer.DataBufferFactory;
23 import org.springframework.http.HttpHeaders;
24 import org.springframework.http.HttpStatus;
25 import org.springframework.http.ResponseEntity;
26 import org.springframework.http.server.reactive.ServerHttpRequest;
27 import org.springframework.http.server.reactive.ServerHttpResponse;
28 import org.springframework.stereotype.Component;
29 import org.springframework.web.reactive.function.client.WebClient;
30 import org.springframework.web.reactive.function.client.WebClientResponseException;
31 import org.springframework.web.server.ServerWebExchange;
32 import reactor.core.publisher.Mono;
33
34 import java.util.Date;
35 import java.util.List;
36 import java.util.function.Predicate;
37
38 @Component
39 @Slf4j
40 public class AuthenticationPrefilter extends AbstractGatewayFilterFactory<AuthenticationPrefilter
41
42     @Autowired
43     @Qualifier("excludedUrls")
44     List<String> excludedUrls;
45     private final WebClient.Builder webClientBuilder;
```

```
45      private final WebClient.Builder webClientBuilder;
46
47      public AuthenticationPrefilter(WebClient.Builder webClientBuilder) {
48          super(Config.class);
49          this.webClientBuilder=webClientBuilder;
50      }
51
52      @Autowired
53      private ObjectMapper objectMapper;
54
55      @Override
56      public GatewayFilter apply(Config config) {
57          return (exchange, chain) -> {
58              ServerHttpRequest request = exchange.getRequest();
59              log.info("************************************************************************"
60              log.info("URL is - " + request.getURI().getPath());
61              String bearerToken = request.getHeaders().getFirst(SecurityConstants.HEADER);
62              log.info("Bearer Token: "+ bearerToken);
63
64              if(isSecured.test(request)) {
65                  return webClientBuilder.build().get()
66                          .uri("lb://authentication-service/api/v1/validateToken")
67                          .header(SecurityConstants.HEADER, bearerToken)
68                          .retrieve().bodyToMono(ConnValidationResponse.class)
69                          .map(response -> {
70                              exchange.getRequest().mutate().header("username", response.getUsernam
71                              exchange.getRequest().mutate().header("authorities", response.getAuth
72
73                              return exchange;
74                          }).flatMap(chain::filter).onErrorResume(error -> {
75                              log.info("Error Happened");
76                              HttpStatus errorCode = null;
77                              String errorMsg = "";
78                              if (error instanceof WebClientResponseException) {
79                                  WebClientResponseException webCLientException = (WebClientRespons
80                                  errorCode = webCLientException.getStatusCode();
81                                  errorMsg = webCLientException.getStatusText();
82
83                              } else {
84                                  errorCode = HttpStatus.BAD_GATEWAY;
85                                  errorMsg = HttpStatus.BAD_GATEWAY.getReasonPhrase();
86                              }
87  //                            AuthorizationFilter.AUTH_FAILED_CODE
88                              return onError(exchange, String.valueOf(errorCode.value()) ,errorMsg,
89                          });
```

```
package com.infotrends.in.InfoTrendsIn.ApiGateway.config;

import com.fasterxml.jackson.core.JsonFactory;
import com.fasterxml.jackson.core.JsonGenerator;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateSerializer;
import com.fasterxml.jackson.datatype.jsr310.ser.LocalDateTimeSerializer;
import com.infotrends.in.InfoTrendsIn.ApiGateway.filters.AuthenticationPrefilter;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.jackson.Jackson2ObjectMapperBuilderCustomizer;
import org.springframework.cloud.gateway.route.RouteLocator;
import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.text.SimpleDateFormat;
import java.time.format.DateTimeFormatter;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

@Configuration
public class RouteConfiguration {

    @Bean
    public RouteLocator routes(
            RouteLocatorBuilder builder,
            AuthenticationPrefilter authFilter) {
        return builder.routes()
                .route("auth-service-route", r -> r.path("/authentication-service/**")
                        .filters(f ->
                                f.rewritePath("/authentication-service(?<segment>/?.*)", "$\\{segm
                                        .filter(authFilter.apply(
                                                new AuthenticationPrefilter.Config())))
                        .uri("lb://authentication-service"))
                .route("user-service-route", r -> r.path("/user-service/**")
                        .filters(f ->
                                f.rewritePath("/user-service(?<segment>/?.*)", "$\\{segment}")
                                        .filter(authFilter.apply(
                                                new AuthenticationPrefilter.Config())))
                        .uri("lb://user-service"))
                .build();
    }
```

```
45          }
46
47      }
```

RouteConfiguration.java hosted with ❤ by GitHub view raw

- A Configuration class extending the WebSecurityConfigurerAdapter class overriding it's ***void configure(HttpSecurity http) throws Exception*** is created. Here, the Custom filter is defined to be executing the **UsernamePasswordAuthenticationFilter** filter in Spring Boot.

```java
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired
    private PasswordEncoder encoder;

    @Value("${security.users.username}")
    private String username;

    @Value("${security.users.password}")
    private String password;

    @Autowired
    private AppUserDetailsService appUserDetailsService;


    @Override
    protected void configure(HttpSecurity http)
      throws Exception {
        http.csrf().disable()
                .headers().frameOptions().disable()
                .and()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .addFilterBefore(new JWTVerifierFilter(), UsernamePasswordAuthenticationFilter.cla
                .authorizeRequests()
                .antMatchers(HttpMethod.GET, "/api/v1/users").permitAll()
                .anyRequest()
                .authenticated()
                .and().httpBasic();

    }
}
```

UsersSecurityConfig.java hosted with ❤ by GitHub view raw

- The **JWTVerifierFilter** filter will check if any username and authority data were added in the header parameter for the request (by the API Gateway) and will create

SecurityContextHolder class.

```
package com.infotrends.in.InfoTrendsIn.config.security.filters;

import com.infotrends.in.InfoTrendsIn.security.SecurityConstants;
import com.infotrends.in.InfoTrendsIn.utils.Utilities;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jws;
import io.jsonwebtoken.Jwts;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.*;
import java.util.stream.Collectors;

public class JWTVerifierFilter extends OncePerRequestFilter {


    @Override
    protected void doFilterInternal(HttpServletRequest httpServletRequest, HttpServletResponse htt
        String authHeader = httpServletRequest.getHeader("Authorization");
        if(!Utilities.validString(authHeader) || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(httpServletRequest, httpServletResponse);
            return;
        }

        logHeaders(httpServletRequest);
        String username=httpServletRequest.getHeader("username");
        List<Map<String, String>> authorities = new ArrayList<>();
        String authoritiesStr = httpServletRequest.getHeader("authorities");
        Set<SimpleGrantedAuthority> simpleGrantedAuthorities = new HashSet<>();
        if(Utilities.validString(authoritiesStr)) {
            simpleGrantedAuthorities=Arrays.stream(authoritiesStr.split(",")).distinct()
                    .filter(Utilities::validString).map(SimpleGrantedAuthority::new).collect(Colle
        }
        Authentication authentication = new UsernamePasswordAuthenticationToken(username, null, si
        SecurityContextHolder.getContext().setAuthentication(authentication);
```

```
46          filterChain.doFilter(httpServletRequest, httpServletResponse);
47
48      }
49
50      private void logHeaders(HttpServletRequest httpServletRequest) {
51          Enumeration<String> headerNames = httpServletRequest.getHeaderNames();
52          while(headerNames.hasMoreElements()) {
53              String header=headerNames.nextElement();
54              logger.info(String.format("Header: %s --- Value: %s", header, httpServletRequest.getHe
55
56          }
57      }
58  }
```

JWTVerifierFilter.java hosted with ❤ by GitHub view raw

## Common Dependencies Used:

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

```
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
</dependency>
```

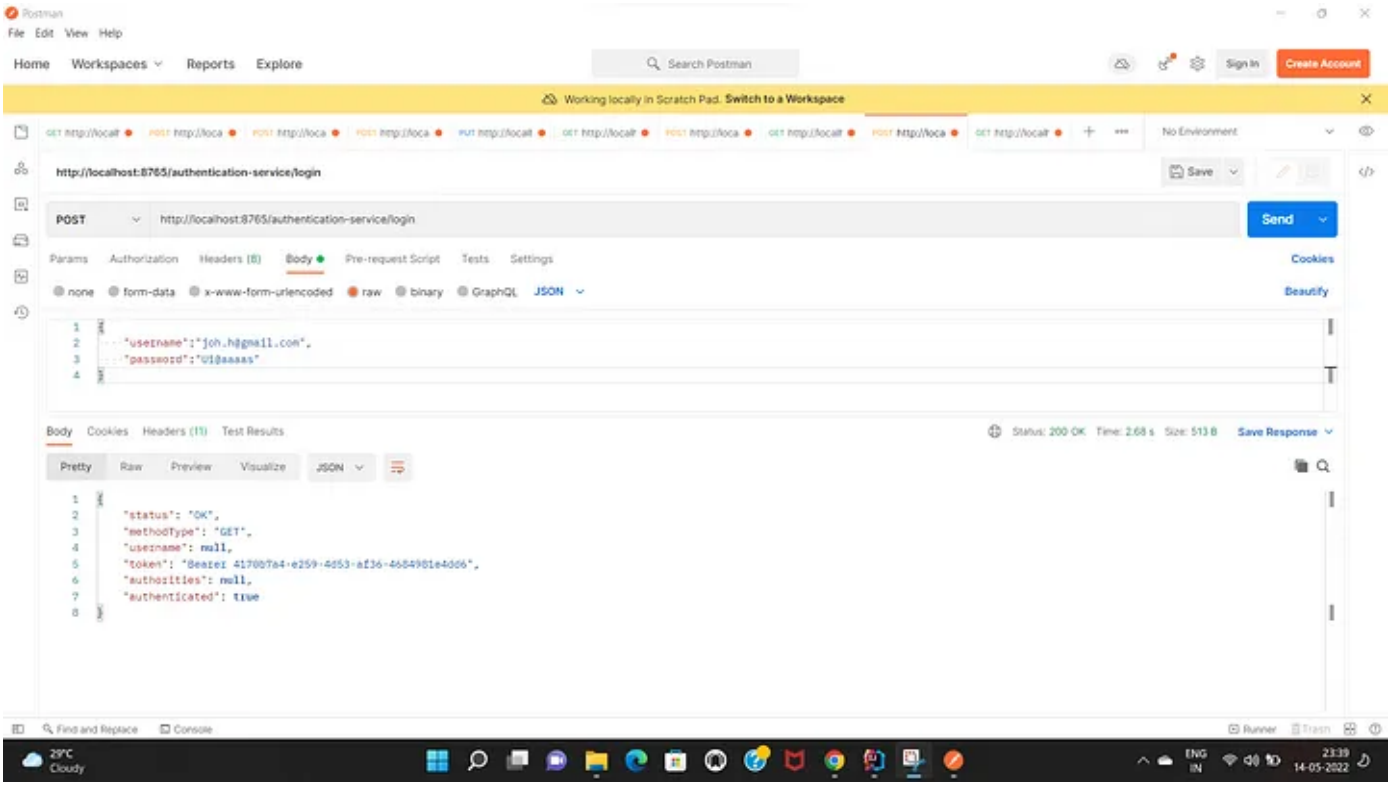
For Using the Spring Cloud libraries like the Config Server and Eureka Clients, a separate section must be added under dependency management in the corresponding POM files.

```
<properties>
   <spring-cloud.version>2020.0.3</spring-cloud.version>
</properties>
<dependencyManagement>
   <dependencies>
      <dependency>
         <groupId>org.springframework.cloud</groupId>
         <artifactId>spring-cloud-dependencies</artifactId>
         <version>${spring-cloud.version}</version>
         <type>pom</type>
         <scope>import</scope>
      </dependency>
   </dependencies>
</dependencyManagement>
```

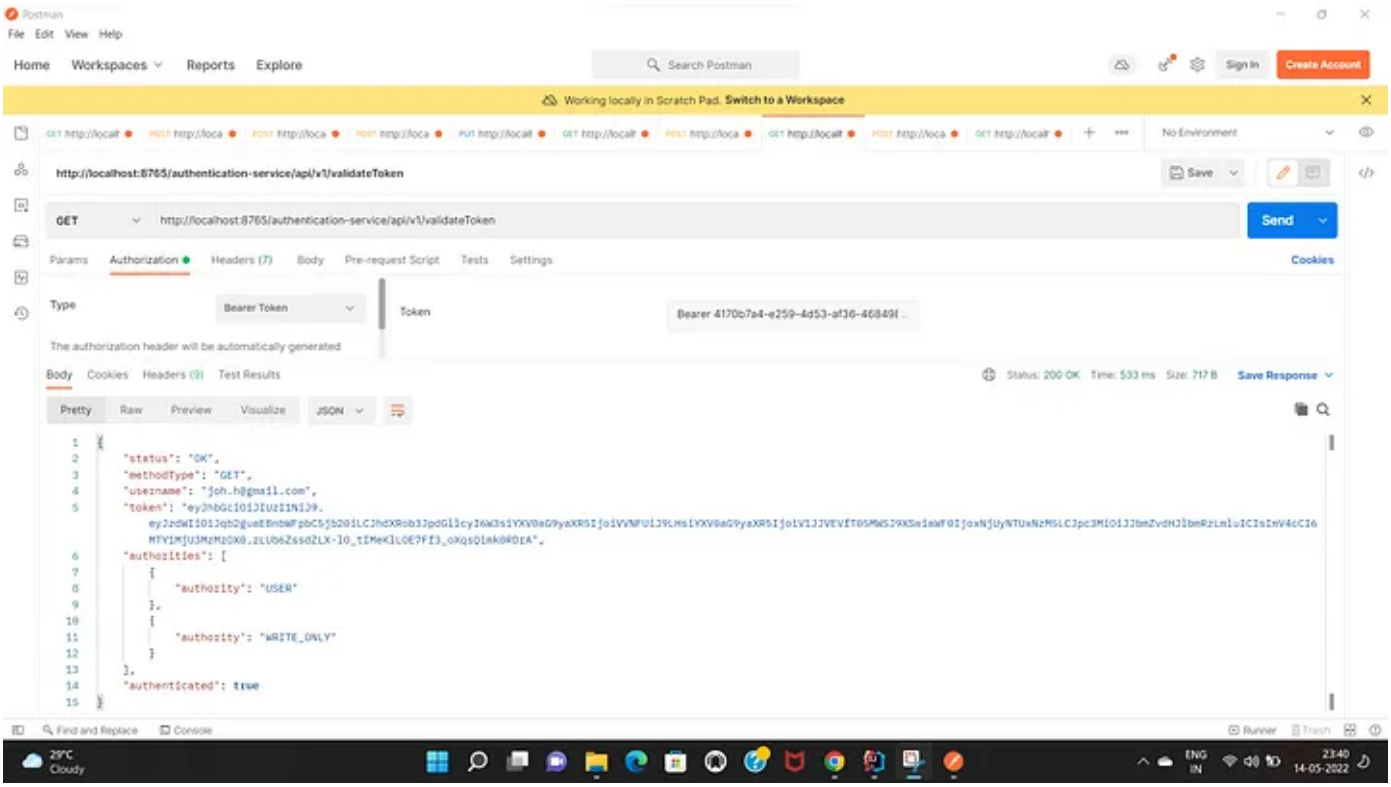
## JWT Token Usage

**Validation of the requests:**

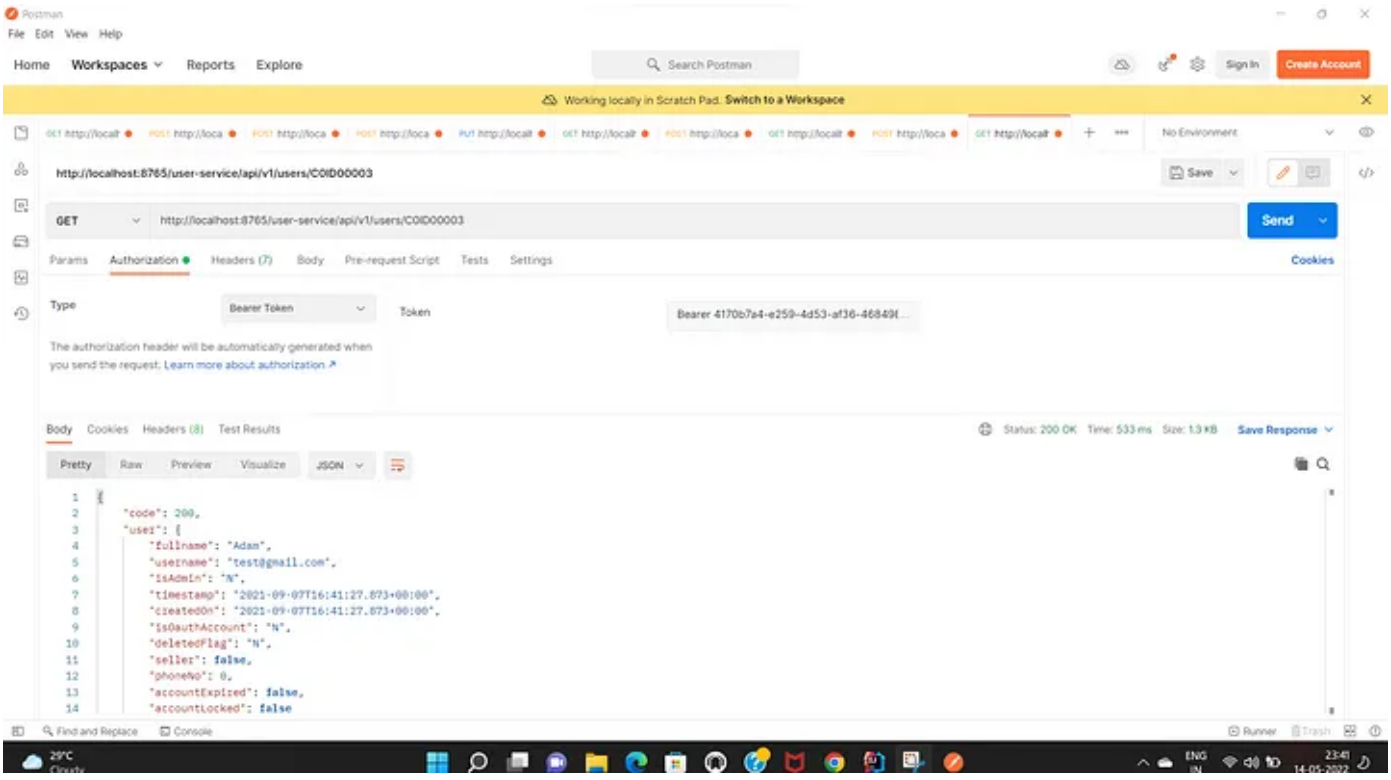
- This is the login API that is exposed at the end that can be used to Authenticate the users and generate the JWT Token.



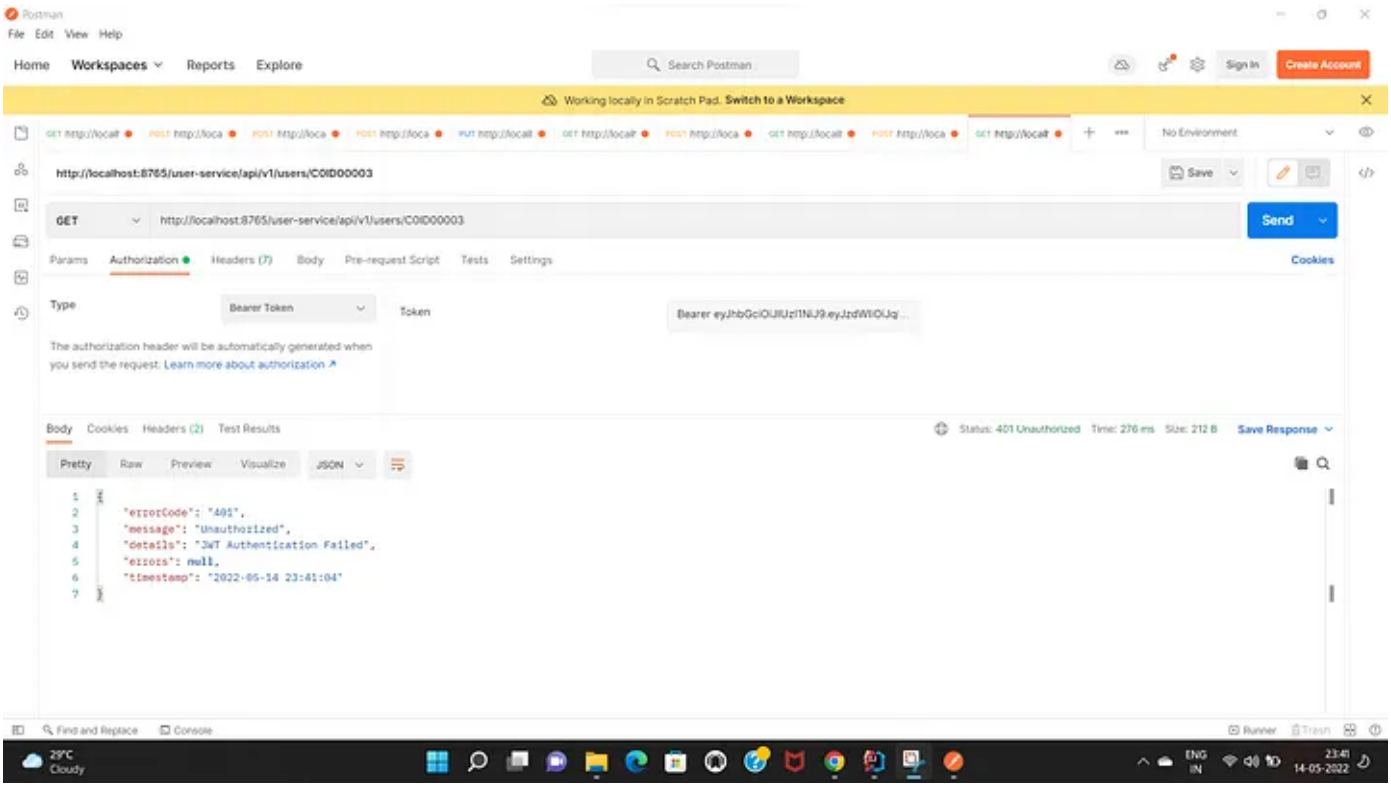

- The *validateToken* API that is used internally to validate the token being sent in the request and get the Authorization details for the token. (This API will later be blocked from external access).



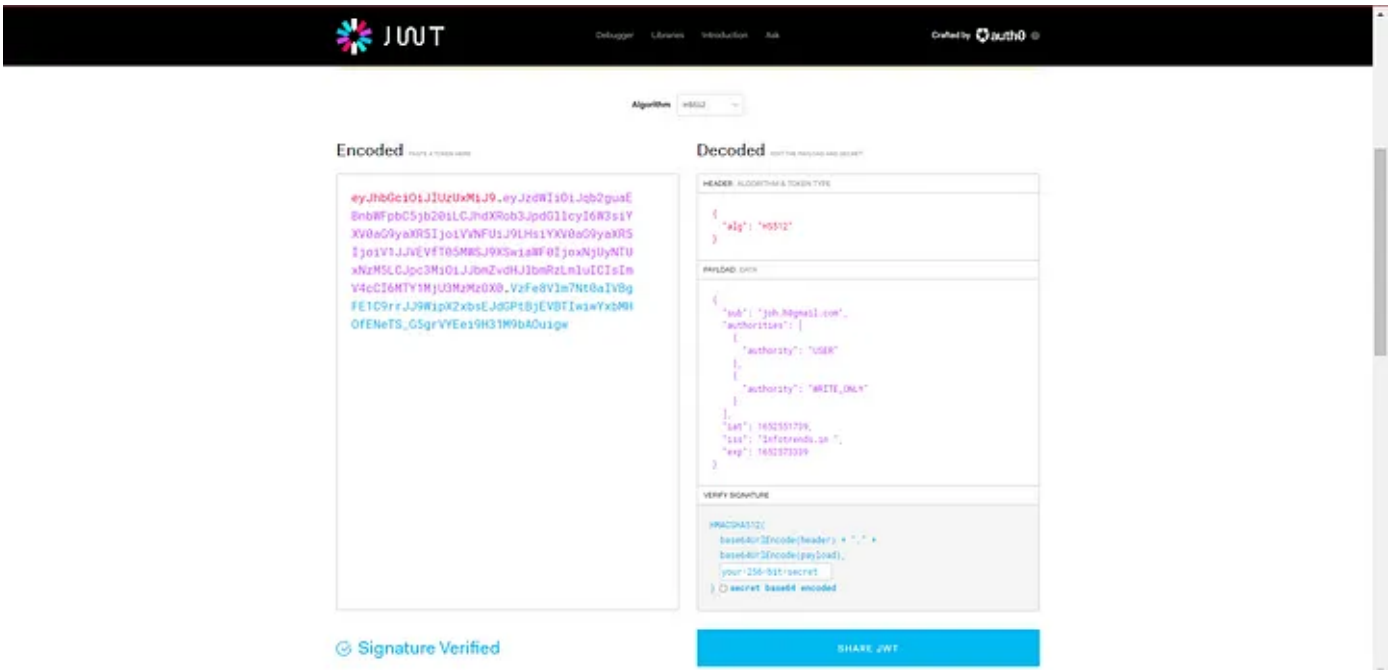

- A Secured Resource in the User-Service when called via the API gateway with a valid Authorization Token.




- The Secured Resource in the User-Service when called via the API gateway without a valid Authorization Token.




- A Sample JWT Token being generated by the Authorization service.




**Update:** You can find the Full Project at: https://github.com/Vicky-cmd/Authentication-Service.git

So that's all in this topic for now. Reviews are always appreciated. ✌

Java Spring Spring Boot Microservices API

---