# How to Implement JWT in microservices-

Refer Doc-**https://www.learncodewithdurgesh.com/blogs/jwt-authentication-with-spring-boot-31**

1. Create a spring boot project with spring boot initializer with Java 17/Spring Boot 3.0
2. Add following dependencies at starting-
3. spring-boot-starter-web
4. Lombok

Create a temporary project with @GetMapping

Starting with JWT

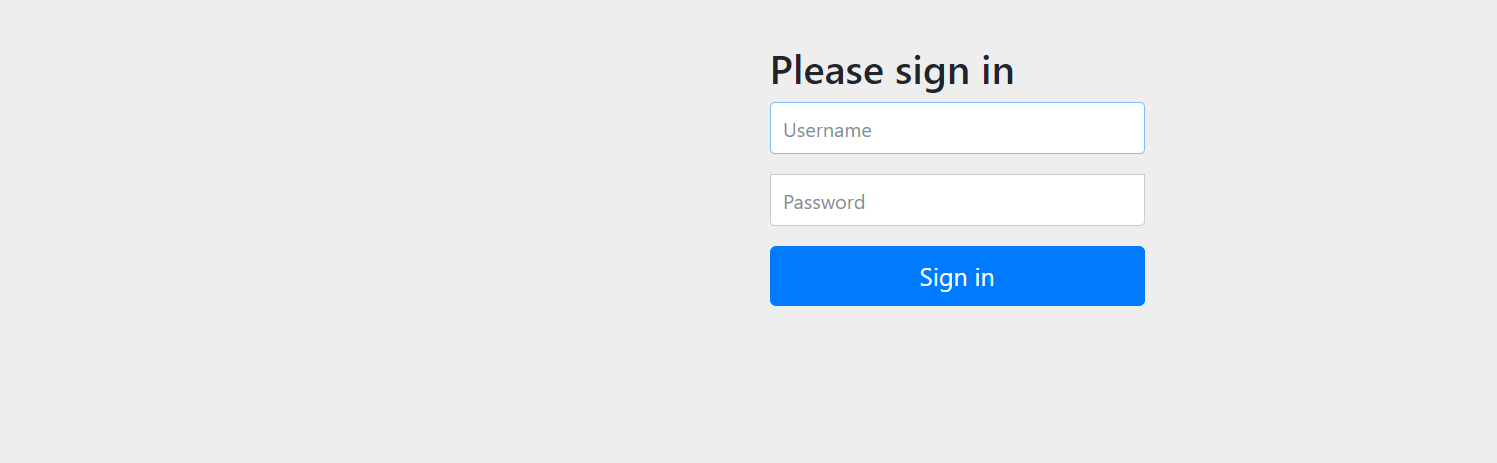
You need to add **spring-security** dependency to secure your apI, default password will be generated at your console with default user user once you enter the correct details ,it will run automatically.

<dependency>

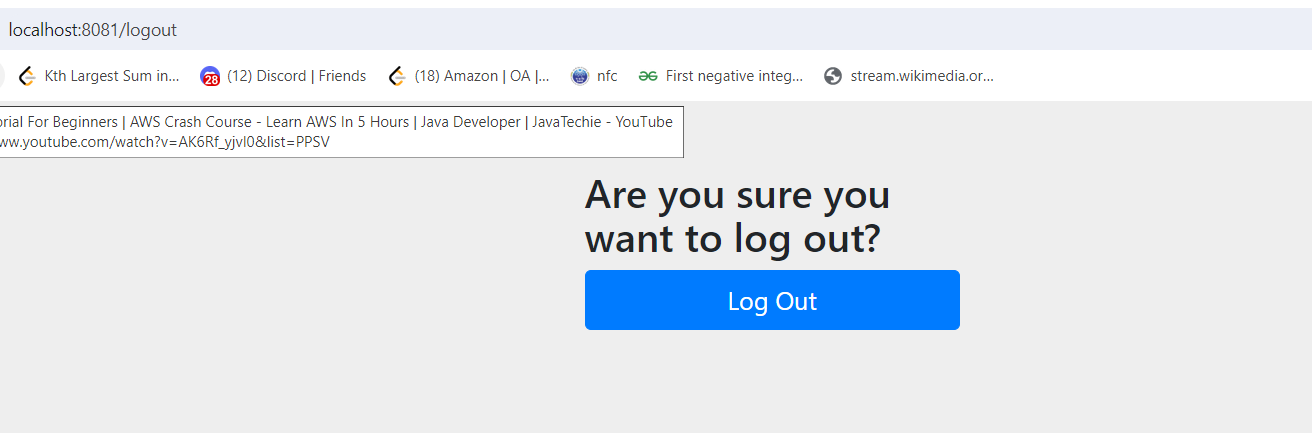
<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>



It is used to logout.



spring.security.user.name=manthan

spring.security.user.password=manthan

when you will add these properties ,no default password will be generated now.it will login with specific password now.

## How to use In Memory User

1)Comment out - spring.security.user.name=manthan

spring.security.user.password=manthan

2)Create. Config Package-

In config package you can create AppConfig.class,in which it creates beans as well.

@Configuration

**public** **class** AppConfig {

@Bean

**public** UserDetailsService userDetailService() {

UserDetails user=User.*builder*().username("harsh").password(passwordEncoder().encode("abc")).roles("ADMIN").build();

UserDetails user1=User.*builder*().username("manthan").password(passwordEncoder().encode("abc")).roles("ADMIN").build();

**return** **new** InMemoryUserDetailsManager(user,user1);

}

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

}

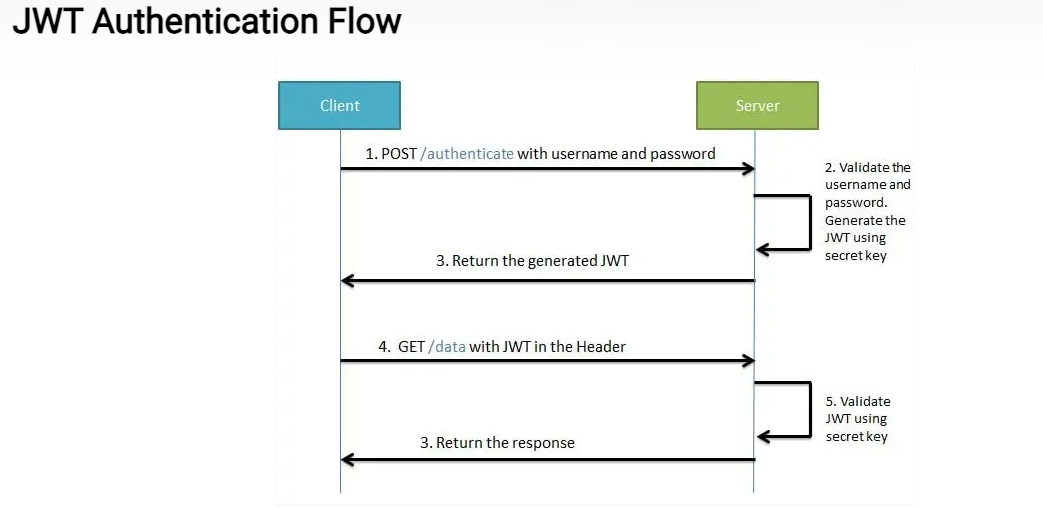
How to Create an ApI for getting current user-

@GetMapping("/current-users")

**public** String getLoggedInUser(Principal principal) {

**return** principal.getName();

}



JWT-Json Web Token All the things are collected

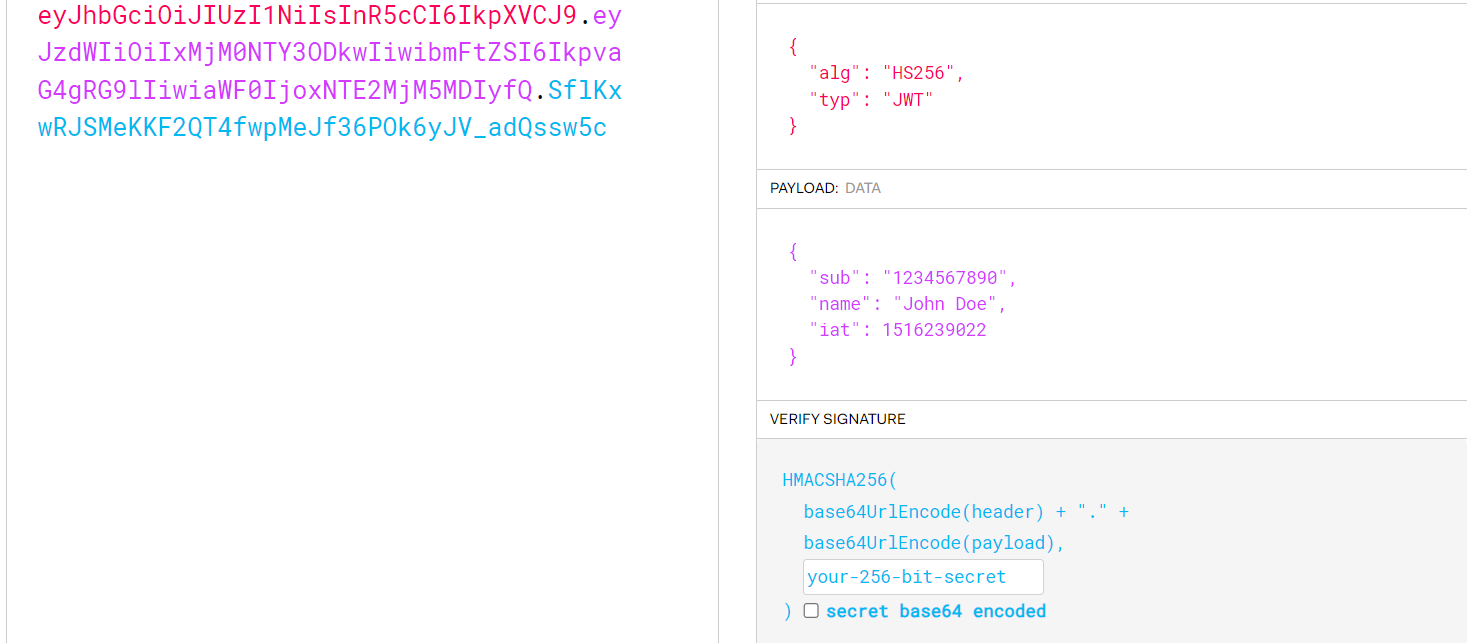
And compiled and shared to client

Json Web Token is composed of three parts-> X.Y.Z

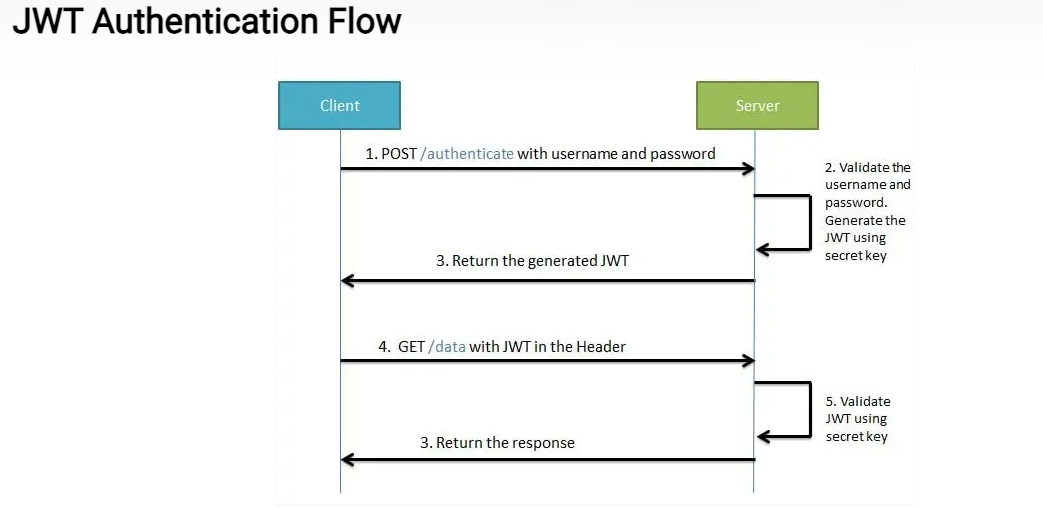
X->Represents algorithm and token type.

Y->Payload and data type(Subject+Name+int)

Z->Verify Signature(E(Header) +E(Payload) +E (key))



Steps how Jwt tokens works-



1)Authenticate username and password set by the client.

2)Return the generated JWT token.

3)pass that token in header it will return response.

## Steps to implement JWT in spring boot Project:-

### Steps to implement jwt token:

**1)** Make sure spring-boot-starter-security is there in pom.xml

**2)**Create Class JWTAthenticationEntryPoint that implement AuthenticationEntryPoint. Method of this class is called whenever as exception is thrown due to unauthenticated user trying to access the resource that required authentication.

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

@Override

public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException, ServletException {

response.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);

PrintWriter writer = response.getWriter();

writer.println("Access Denied !! " + authException.getMessage());

}

}

1. Create **JWTHelper**  class This class contains method related to perform operations with jwt token like generateToken, validateToken etc.

@Component

public class JwtHelper {

//requirement :

public static final long JWT\_TOKEN\_VALIDITY = 5 \* 60 \* 60;

// public static final long JWT\_TOKEN\_VALIDITY = 60;

private String secret = "afafasfafafasfasfasfafacasdasfasxASFACASDFACASDFASFASFDAFASFASDAADSCSDFADCVSGCFVADXCcadwavfsfarvf";

//retrieve username from jwt token

public String getUsernameFromToken(String token) {

return getClaimFromToken(token, Claims::getSubject);

}

//retrieve expiration date from jwt token

public Date getExpirationDateFromToken(String token) {

return getClaimFromToken(token, Claims::getExpiration);

}

public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {

final Claims claims = getAllClaimsFromToken(token);

return claimsResolver.apply(claims);

}

//for retrieveing any information from token we will need the secret key

private Claims getAllClaimsFromToken(String token) {

return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();

}

//check if the token has expired

private Boolean isTokenExpired(String token) {

final Date expiration = getExpirationDateFromToken(token);

return expiration.before(new Date());

}

//generate token for user

public String generateToken(UserDetails userDetails) {

Map<String, Object> claims = new HashMap<>();

return doGenerateToken(claims, userDetails.getUsername());

}

//while creating the token -

//1. Define claims of the token, like Issuer, Expiration, Subject, and the ID

//2. Sign the JWT using the HS512 algorithm and secret key.

//3. According to JWS Compact Serialization(https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41#section-3.1)

// compaction of the JWT to a URL-safe string

private String doGenerateToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + JWT\_TOKEN\_VALIDITY \* 1000))

.signWith(SignatureAlgorithm.HS512, secret).compact();

}

//validate token

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = getUsernameFromToken(token);

return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

}

**4)**Create JWTAuthenticationFilter that extends OncePerRequestFilter and override method and write the logic to check the token that is comming in header. We have to write 5 important logic

**This class will run at first part of the request and will check whether header has token or not and whether or not token is valid or not.**

* 1. Get Token from request
  2. Validate Token
  3. GetUsername from token
  4. Load user associated with this token
  5. set authentication

Now we can start the process of creation of Authentication filter class.

It will execute always first before API.if everything is working fine .this line will execute successfully.

SecurityContextHolder.*getContext*().setAuthentication(authentication);

And request will be forwarded accordingly.

@Component

**public** **class** JwtAuthenticationFilter **extends** OncePerRequestFilter {

**private** Logger logger = LoggerFactory.*getLogger*(OncePerRequestFilter.**class**);

@Autowired

**private** JwtHelper jwtHelper;

@Autowired

**private** UserDetailsService userDetailsService;

@Override

**protected** **void** doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

**throws** ServletException, IOException {

// **TODO** Auto-generated method stub

//Authorization=Bearer ......

String requestHeader = request.getHeader("Authorization");

logger.info(" Header : {}", requestHeader);

String username = **null**;

String token = **null**;

**if** (requestHeader != **null** && requestHeader.startsWith("Bearer")) {

//looking good

token = requestHeader.substring(7);

**try** {

username = **this**.jwtHelper.getUsernameFromToken(token);

} **catch** (IllegalArgumentException e) {

logger.info("Illegal Argument while fetching the username !!");

e.printStackTrace();

} **catch** (ExpiredJwtException e) {

logger.info("Given jwt token is expired !!");

e.printStackTrace();

} **catch** (MalformedJwtException e) {

logger.info("Some changed has done in token !! Invalid Token");

e.printStackTrace();

} **catch** (Exception e) {

e.printStackTrace();

}

} **else** {

logger.info("Invalid Header Value !! ");

}

//

**if** (username != **null** && SecurityContextHolder.*getContext*().getAuthentication() == **null**) {

//fetch user detail from username

UserDetails userDetails = **this**.userDetailsService.loadUserByUsername(username);

Boolean validateToken = **this**.jwtHelper.validateToken(token, userDetails);

**if** (validateToken) {

//set the authentication

//UsernamePasswordAuthenticationToken use-checks whether the token is authenticated or not.

UsernamePasswordAuthenticationToken authentication = **new** UsernamePasswordAuthenticationToken(userDetails, **null**, userDetails.getAuthorities());

authentication.setDetails(**new** WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.*getContext*().setAuthentication(authentication);

} **else** {

logger.info("Validation fails !!");

}

}

filterChain.doFilter(request, response);

}

}

Associates a given [SecurityContext](eclipse-javadoc:%E2%98%82=JwtExample3/C:%5C/Users%5C/Hp%5C/.m2%5C/repository%5C/org%5C/springframework%5C/security%5C/spring-security-core%5C/6.3.1%5C/spring-security-core-6.3.1.jar=/maven.pomderived=/true=/=/maven.pomderived=/true=/=/maven.groupId=/org.springframework.security=/=/maven.artifactId=/spring-security-core=/=/maven.version=/6.3.1=/=/maven.scope=/compile=/%3Corg.springframework.security.core.context(SecurityContextHolder.class%E2%98%83SecurityContextHolder%E2%98%82SecurityContext) with the current execution thread.

This class provides a series of static methods that delegate to an instance of [org.springframework.security.core.context.SecurityContextHolderStrategy](eclipse-javadoc:%E2%98%82=JwtExample3/C:%5C/Users%5C/Hp%5C/.m2%5C/repository%5C/org%5C/springframework%5C/security%5C/spring-security-core%5C/6.3.1%5C/spring-security-core-6.3.1.jar=/maven.pomderived=/true=/=/maven.pomderived=/true=/=/maven.groupId=/org.springframework.security=/=/maven.artifactId=/spring-security-core=/=/maven.version=/6.3.1=/=/maven.scope=/compile=/%3Corg.springframework.security.core.context(SecurityContextHolder.class%E2%98%83SecurityContextHolder%E2%98%82org.springframework.security.core.context.SecurityContextHolderStrategy). The purpose of the class is to provide a convenient way to specify the strategy that should be used for a given JVM. This is a JVM-wide setting, since everything in this class is static to facilitate ease of use in calling code.

To specify which strategy should be used, you must provide a mode setting. A mode setting is one of the three valid MODE\_ settings defined as static final fields, or a fully qualified classname to a concrete implementation of [org.springframework.security.core.context.SecurityContextHolderStrategy](eclipse-javadoc:%E2%98%82=JwtExample3/C:%5C/Users%5C/Hp%5C/.m2%5C/repository%5C/org%5C/springframework%5C/security%5C/spring-security-core%5C/6.3.1%5C/spring-security-core-6.3.1.jar=/maven.pomderived=/true=/=/maven.pomderived=/true=/=/maven.groupId=/org.springframework.security=/=/maven.artifactId=/spring-security-core=/=/maven.version=/6.3.1=/=/maven.scope=/compile=/%3Corg.springframework.security.core.context(SecurityContextHolder.class%E2%98%83SecurityContextHolder%E2%98%82org.springframework.security.core.context.SecurityContextHolderStrategy) that provides a public no-argument constructor.

There are two ways to specify the desired strategy mode String. The first is to specify it via the system property keyed on [SYSTEM\_PROPERTY](eclipse-javadoc:%E2%98%82=JwtExample3/C:%5C/Users%5C/Hp%5C/.m2%5C/repository%5C/org%5C/springframework%5C/security%5C/spring-security-core%5C/6.3.1%5C/spring-security-core-6.3.1.jar=/maven.pomderived=/true=/=/maven.pomderived=/true=/=/maven.groupId=/org.springframework.security=/=/maven.artifactId=/spring-security-core=/=/maven.version=/6.3.1=/=/maven.scope=/compile=/%3Corg.springframework.security.core.context(SecurityContextHolder.class%E2%98%83SecurityContextHolder%E2%98%82%E2%98%82SYSTEM_PROPERTY). The second is to call [setStrategyName(String)](eclipse-javadoc:%E2%98%82=JwtExample3/C:%5C/Users%5C/Hp%5C/.m2%5C/repository%5C/org%5C/springframework%5C/security%5C/spring-security-core%5C/6.3.1%5C/spring-security-core-6.3.1.jar=/maven.pomderived=/true=/=/maven.pomderived=/true=/=/maven.groupId=/org.springframework.security=/=/maven.artifactId=/spring-security-core=/=/maven.version=/6.3.1=/=/maven.scope=/compile=/%3Corg.springframework.security.core.context(SecurityContextHolder.class%E2%98%83SecurityContextHolder%E2%98%82%E2%98%82setStrategyName%E2%98%82String) before using the class. If neither approach is used, the class will default to using [MODE\_THREADLOCAL](eclipse-javadoc:%E2%98%82=JwtExample3/C:%5C/Users%5C/Hp%5C/.m2%5C/repository%5C/org%5C/springframework%5C/security%5C/spring-security-core%5C/6.3.1%5C/spring-security-core-6.3.1.jar=/maven.pomderived=/true=/=/maven.pomderived=/true=/=/maven.groupId=/org.springframework.security=/=/maven.artifactId=/spring-security-core=/=/maven.version=/6.3.1=/=/maven.scope=/compile=/%3Corg.springframework.security.core.context(SecurityContextHolder.class%E2%98%83SecurityContextHolder%E2%98%82%E2%98%82MODE_THREADLOCAL), which is backwards compatible, has fewer JVM incompatibilities and is appropriate on servers (whereas [MODE\_GLOBAL](eclipse-javadoc:%E2%98%82=JwtExample3/C:%5C/Users%5C/Hp%5C/.m2%5C/repository%5C/org%5C/springframework%5C/security%5C/spring-security-core%5C/6.3.1%5C/spring-security-core-6.3.1.jar=/maven.pomderived=/true=/=/maven.pomderived=/true=/=/maven.groupId=/org.springframework.security=/=/maven.artifactId=/spring-security-core=/=/maven.version=/6.3.1=/=/maven.scope=/compile=/%3Corg.springframework.security.core.context(SecurityContextHolder.class%E2%98%83SecurityContextHolder%E2%98%82%E2%98%82MODE_GLOBAL) is definitely inappropriate for server use).

## In the next step we need to create SecurityConfig in config package-

**package** com.jwt.example.JwtExample3.config;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.http.SessionCreationPolicy;

**import** org.springframework.security.web.SecurityFilterChain;

**import** org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

**import** com.jwt.example.JwtExample3.security.JwtAuthenticationEntryPoint;

**import** com.jwt.example.JwtExample3.security.JwtAuthenticationFilter;

@Configuration

**public** **class** SecurityConfig {

@Autowired

**private** JwtAuthenticationEntryPoint point;

@Autowired

**private** JwtAuthenticationFilter filter;

@Bean

**public** SecurityFilterChain securityFilterChain(HttpSecurity http) **throws** Exception {

//configuration

http.csrf(csrf->csrf.disable())

.cors(cors->cors.disable())

.authorizeHttpRequests(

auth->

auth.requestMatchers("/home/\*\*").authenticated()

.requestMatchers("/auth/login").permitAll()

.anyRequest().authenticated())

.exceptionHandling(ex->ex.authenticationEntryPoint(point))

.sessionManagement(session ->session.sessionCreationPolicy(SessionCreationPolicy.***STATELESS***));

// http.addFilterBefore(filter, UsernamePasswordAuthenticationFilter.class);

http.addFilterBefore(filter,UsernamePasswordAuthenticationFilter.**class**);

**return** http.build();

}

@Bean

**public** AuthenticationManager authenticationManagerBean(AuthenticationConfiguration configuration) **throws** Exception{

**return** configuration.getAuthenticationManager();

}

}

## Create Authentication controller class->

**package** com.jwt.example.JwtExample3.controller;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.authentication.BadCredentialsException;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.core.userdetails.UserDetailsService;

**import** org.springframework.web.bind.annotation.ExceptionHandler;

**import** org.springframework.web.bind.annotation.PostMapping;

**import** org.springframework.web.bind.annotation.RequestBody;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

**import** com.jwt.example.JwtExample3.models.JwtRequest;

**import** com.jwt.example.JwtExample3.models.JwtResponse;

**import** com.jwt.example.JwtExample3.security.JwtHelper;

@RestController

@RequestMapping("/auth")

**public** **class** AuthController {

@Autowired

**private** UserDetailsService userDetailsService;

@Autowired

**private** AuthenticationManager Authmanager;

@Autowired

**private** JwtHelper helper;

**private** Logger logger = LoggerFactory.*getLogger*(AuthController.**class**);

@PostMapping("/login")

**public** ResponseEntity<JwtResponse> login(@RequestBody JwtRequest request) {

**this**.doAuthenticate(request.getEmail(), request.getPassword());

UserDetails userDetails = userDetailsService.loadUserByUsername(request.getEmail());

String token = **this**.helper.generateToken(userDetails);

JwtResponse response = JwtResponse.*builder*()

.jwtToken(token)

.username(userDetails.getUsername()).build();

**return** **new** ResponseEntity<>(response, HttpStatus.***OK***);

}

**private** **void** doAuthenticate(String email, String password) {

UsernamePasswordAuthenticationToken authentication = **new** UsernamePasswordAuthenticationToken(email, password);

**try** {

Authmanager.authenticate(authentication);

} **catch** (BadCredentialsException e) {

**throw** **new** BadCredentialsException(" Invalid Username or Password !!");

}

}

@ExceptionHandler(BadCredentialsException.**class**)

**public** String exceptionHandler() {

**return** "Credentials Invalid !!";

}

}

🡪Run your Code

🡪Test your Api’s

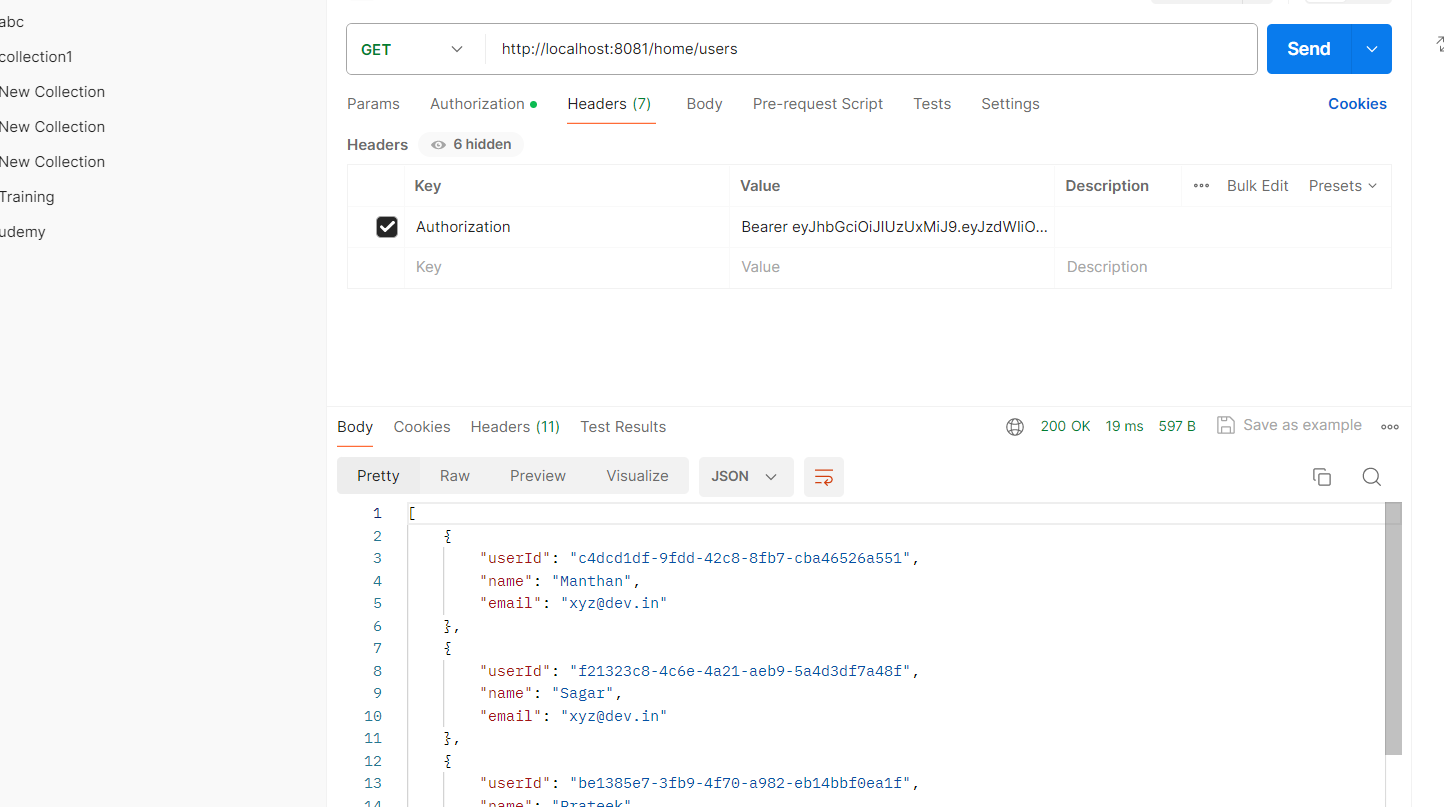
For Token-

<http://localhost:8081/auth/login>

For Results-

<http://localhost:8081/home/user>

http://localhost:8081/home/current-users



It will give results like this

Remember to Run on postman

1)Paste the API on postman

2)In the header write auth only and select Authorization from the popup bar

3)Write Bearer In the Value section write it as Bearer …..

4)get the results