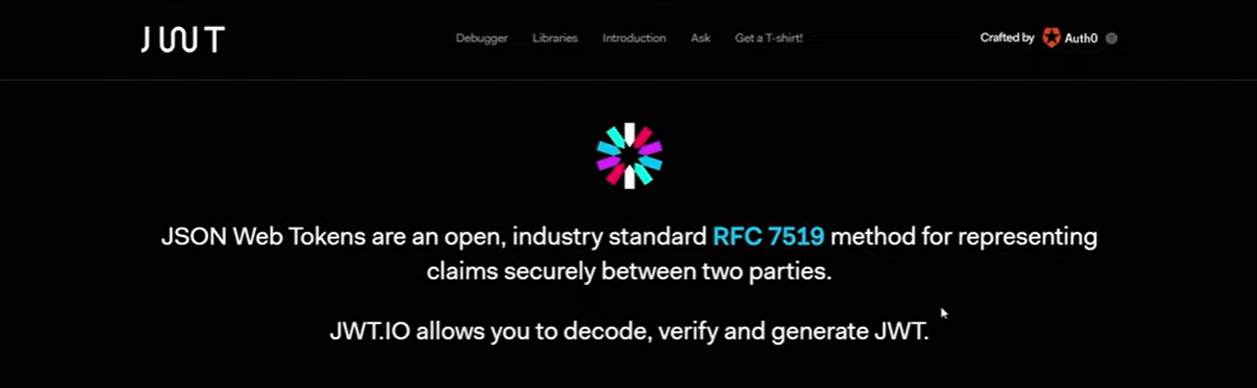
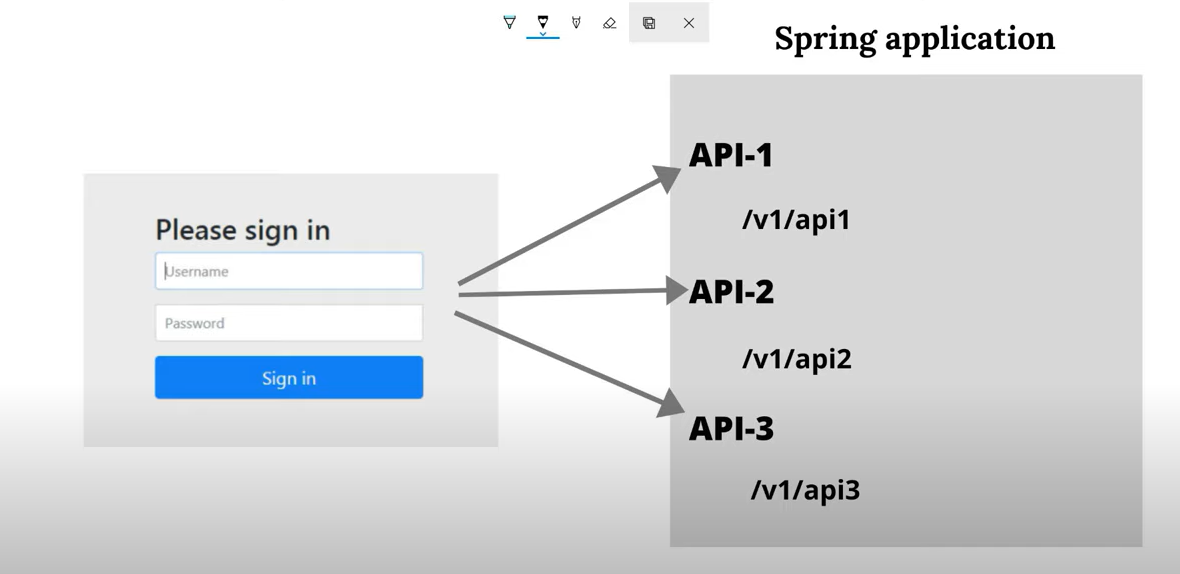
**Spring Boot - Spring Security + JWT Complete Tutorial**

Basically, JWT stands for JSON Web Token. This is one of the best secure way to communicate between client and Server.



The main advantage of JWT is because It completely follows Stateless Authentication Mechanism.

It means all the User Inputs or User states is never saved into Server Memory or Cookies. So, lets understand why JWT what kind of problems we faced with traditional security approach.

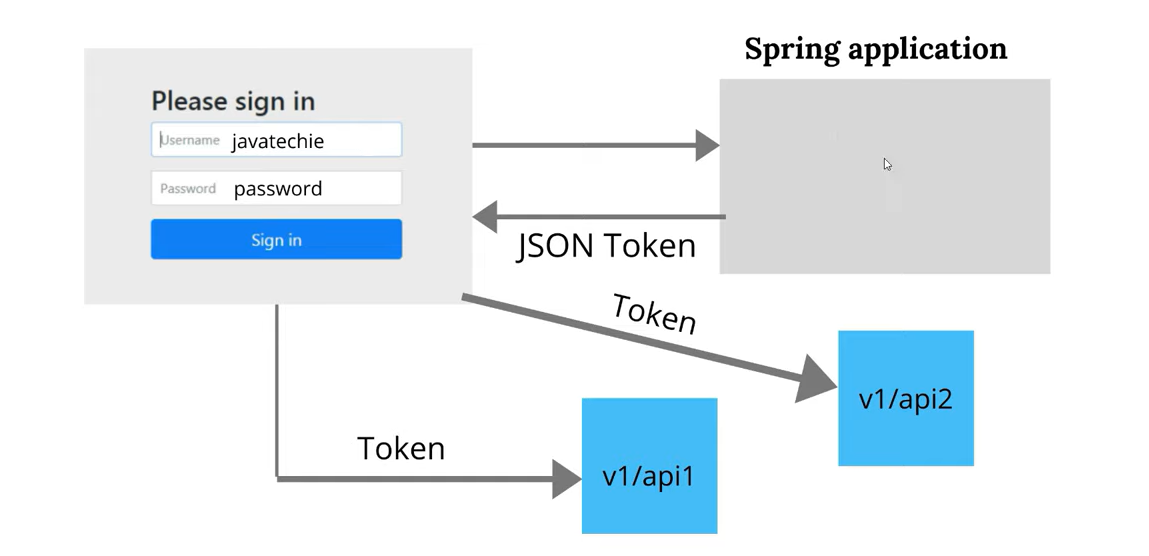


If u see the above diagram, I have client-side application and server-side application. In server-side application we enable Spring-Security. Now when my client wants to access any API from server-side application he needs to provide hard code username and password. Let’s say clients want to access API-1

So, he needs to give url **/v1/api1** along with he needs to pass hardcode username and password as part of request. Same for API-2 and API-3…

It means when client want to access any api from server-side application, always he needs to pass the username and password as part of request. Which doesn’t look good. How we can optimize these things.

I don’t want to give username and password for each and every requests. So, to overcome this issue lets discuss some different approach.



If u can see this diagram same, we have client-side application and server-side application. As we enabled spring security in server-side application.

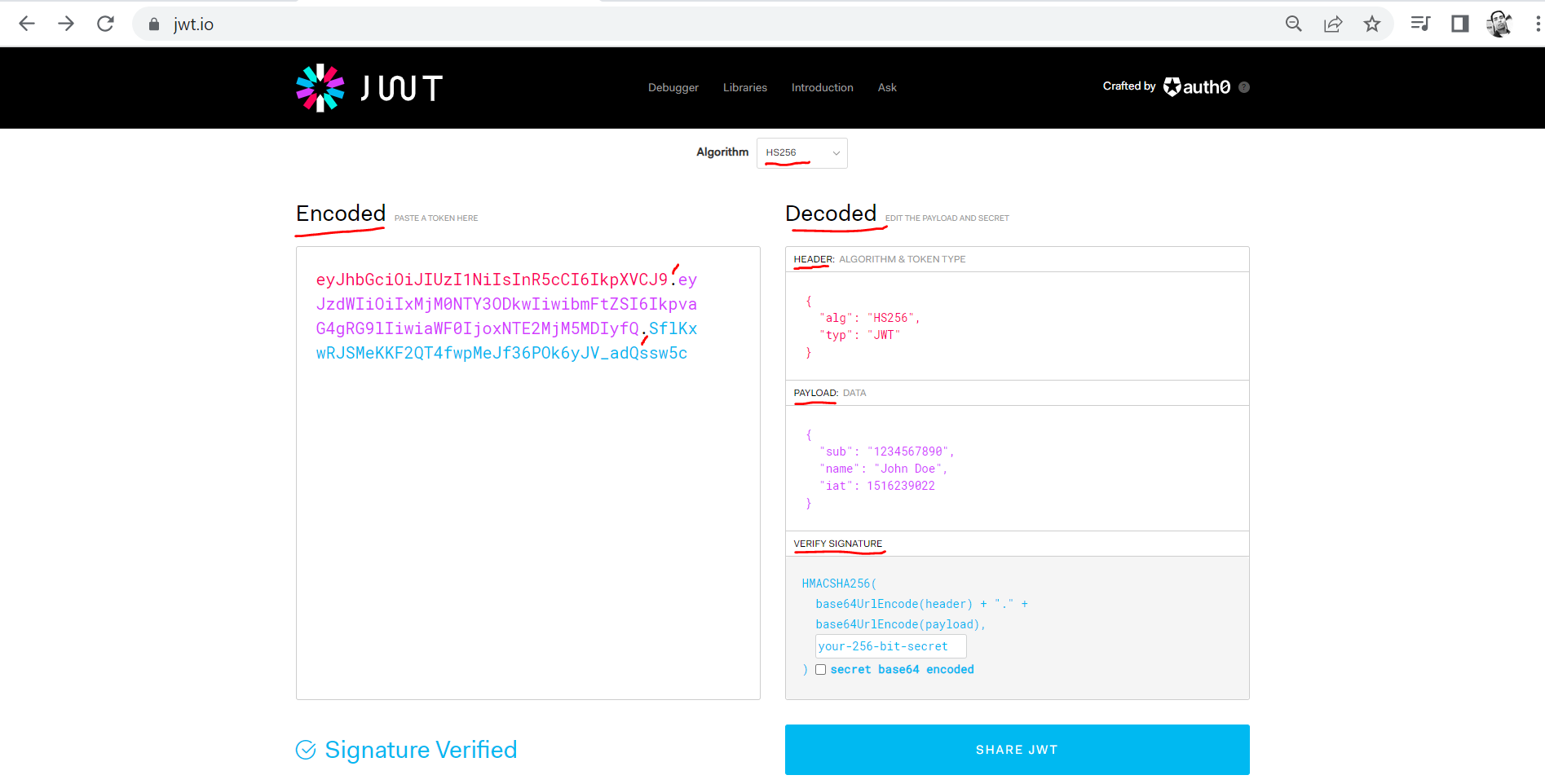
When first time client will send request to this, he need to pass the hardcode username and password as part of request. Now when request will come to ur server-side application, now server will return one encrypted string which is nothing called JSON Token.

And that JSON Token will consist username and password in encrypted format. Now going forward when my client wants to access any other api let’s say u want to access **v1/api2 or v1/api1,** then he no need to pass the username and password in each and every request. He can play with the token which he received from first api call. This is where JSON Web Token came into the picture.

Means Instead of given your username and password in each and every request you can pass your encrypted string which is JSON Web Token.

As we understand this json web token self-contained it means it contained your input in encrypted format. Along with username and password it contained few more fields.

<https://jwt.io/>



If u can see this json web token consists of 3 parts which is separated by dot. the first part is nothing just Header. In header we specified the algorithm type which is by default “**HS256”** and token type is of “**JWT”**

And second part which is nothing all about your payload. Payload in the sense we need to set the user details here. Here we can see name of user, subject and expired date of the token. And third part is nothing about your verified Signature. So, whatever the Header and Payload you will pass that will be converted to base64URLEncoder and it will just give you the verify signature.

After combining all these 3 sections we can able to build one valid JSON Web Token. Which we can use going forward for Authorization.

Let’s create one spring-boot application to understand how we can create a JWT token and how we can implement JWT base security in our application. Here we are going to use H2 Database where we can store username and password.

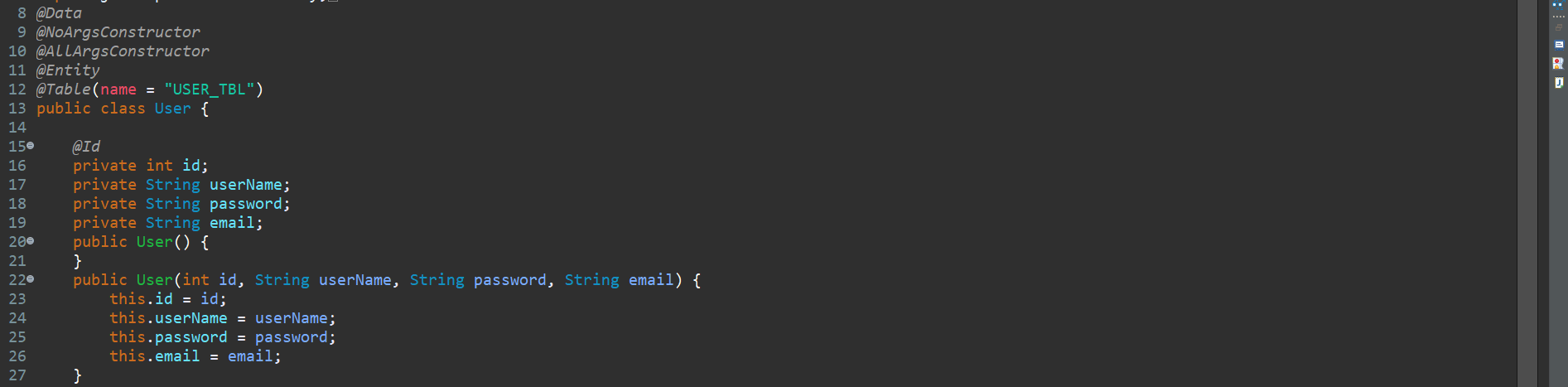
**Application - spring-security-jwt-example**

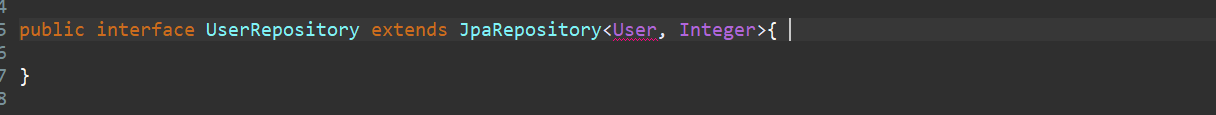
**Dependencies- Lombok, Spring Web, Spring Security, H2 Database, Spring data JPA**

Spring Web and Spring Security these 2 dependencies are required as part of this project, this H2 and JPA I used to store user details later we will add jwt once we will start performing jwt related operation.

Let’s create User Entity and User Repository so we can keep some User details before we create a jwt.

Let’s create all packages 🡪 **Entity/repository/service/controller/util/config**

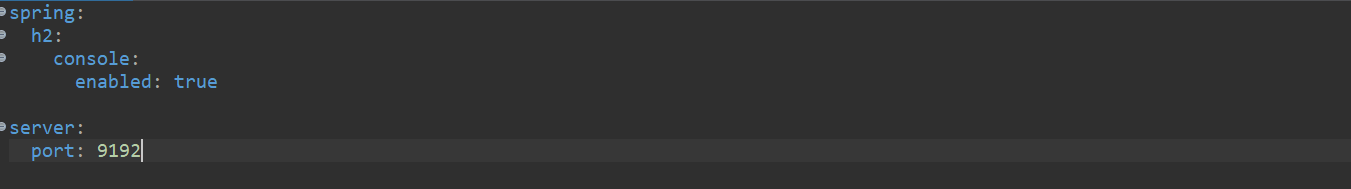




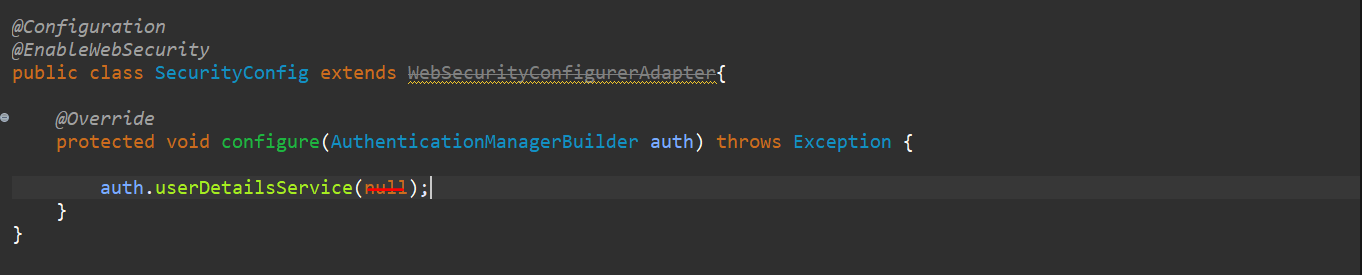
Let’s initialize User’s in our main class…. Now to save the user objects we need user repository. Directly I can call repository.saveAll() and can pass the user object. And as I want this method to be load on my application startup, I can annotate this method @PostConstruct, this will work as init method.



So once my application up this list of users will populate to my database. we will verify that once we will enable the H2 console.

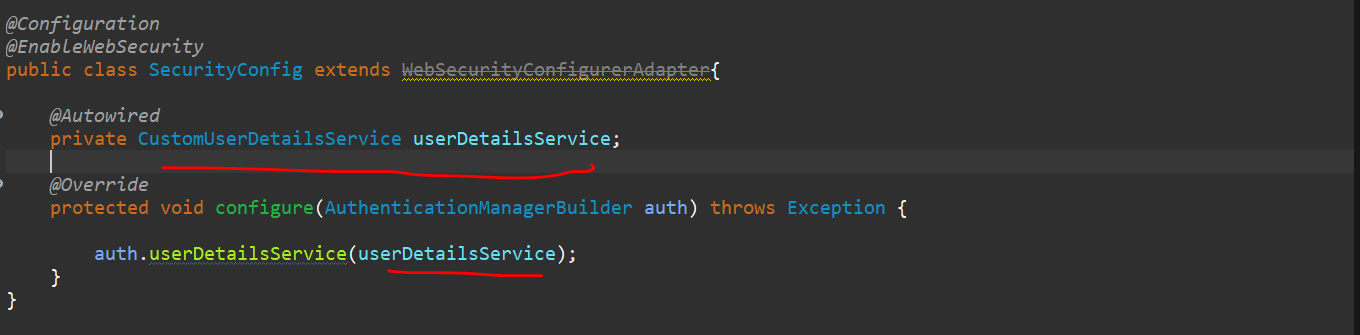


Now let’s write one config class where we can validate the username and password. We have to extends the class **WebSecurityConfigurerAdapter** and we need to annotate with @**Configuration** and @**EnableWebSecurity**. We need to override the configure method which is taking **AuthenticationManagerBuilder**. And use this **auth. userDetailService ()**



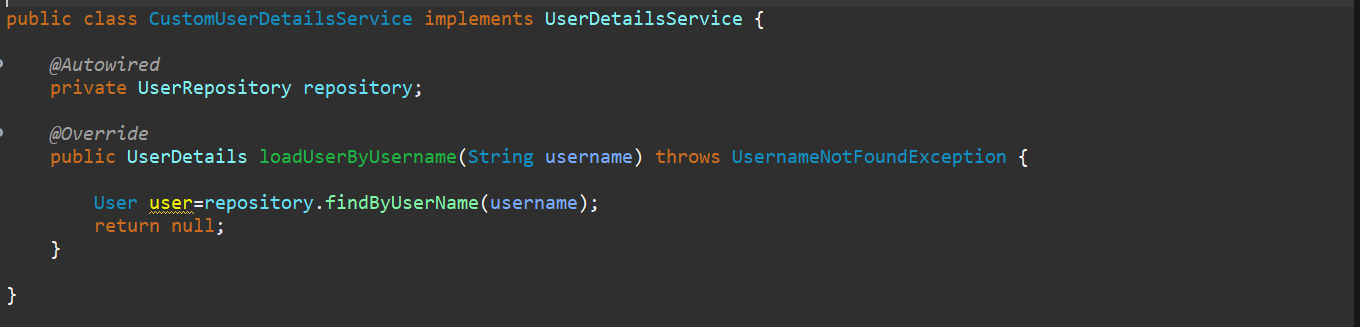
Here you just need to create your own custom UserDetailService and you need to fetch the User object based on the incoming request which will be your username and password. So those logic we will write in separate class. So, let me create a class first with name **CustomUserDetailsService**.

Now go to SecurityConfig class and let me Inject the CustomUserDetailsService and pass it to this guy.

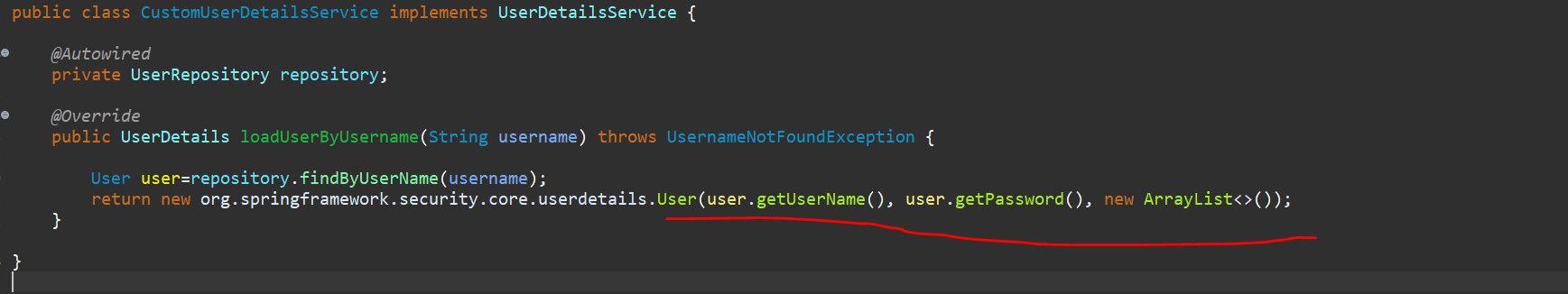


Now our newly created class **CustomUserDetailsService** must implements **UserDetailsService** interface and override one method which would be **loadUserByUserName**.

So whatever username u will give it will go to the DB and based on username it will fetch the User Object. And we need to inject the Repository here so that we can communicate with DB. And now let’s use this repository to fetch User Object from Database and so we need to create findByUserName () method in our Repository.

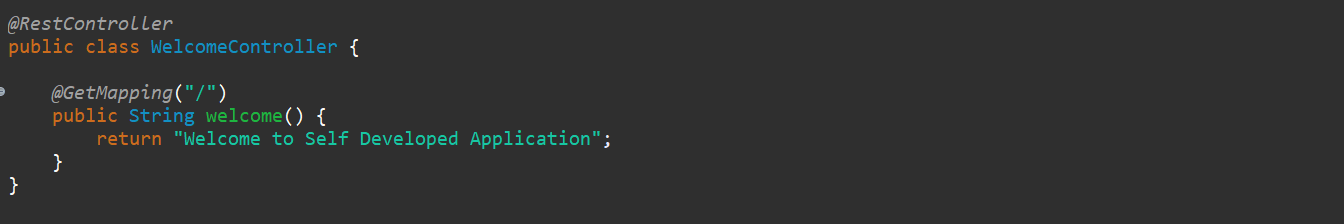


Now come to your CustomUserDetailsService. We got the User Entity, which is presents in our DB, but our method return type is UserDetails. So, what we will do we will just fetch this UserDetails and we will be mapped it to User Object provided by Spring Security.



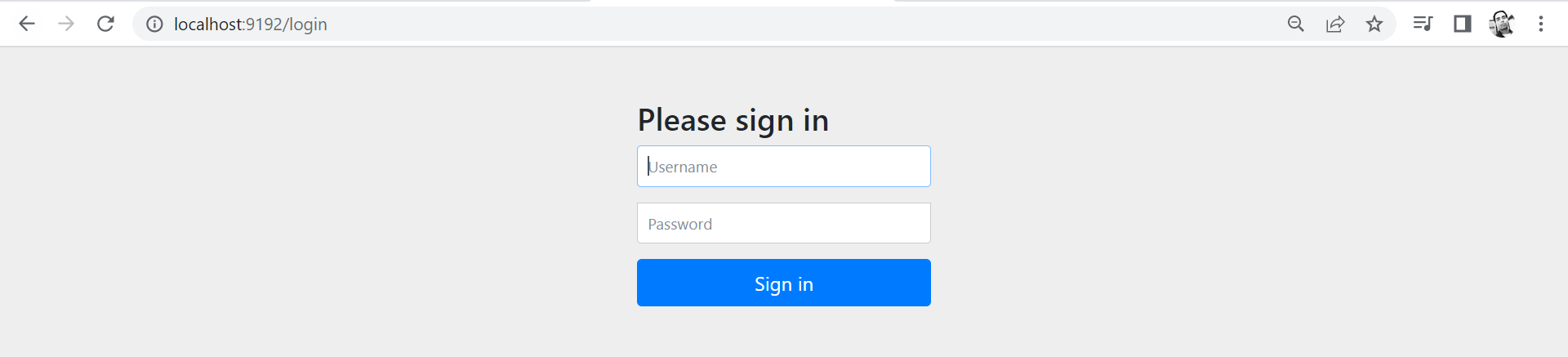
What we are doing in this method, we are getting username as part of Request Header, and based on the username we are fetching User Object from DB, and then same User Object we are giving to the User Object providing by the Spring Security, to validate if the username or password is valid or not. So, we need to annotate this class as Service. so, we have done with our configuration.

let’s create one endpoint in controller class.

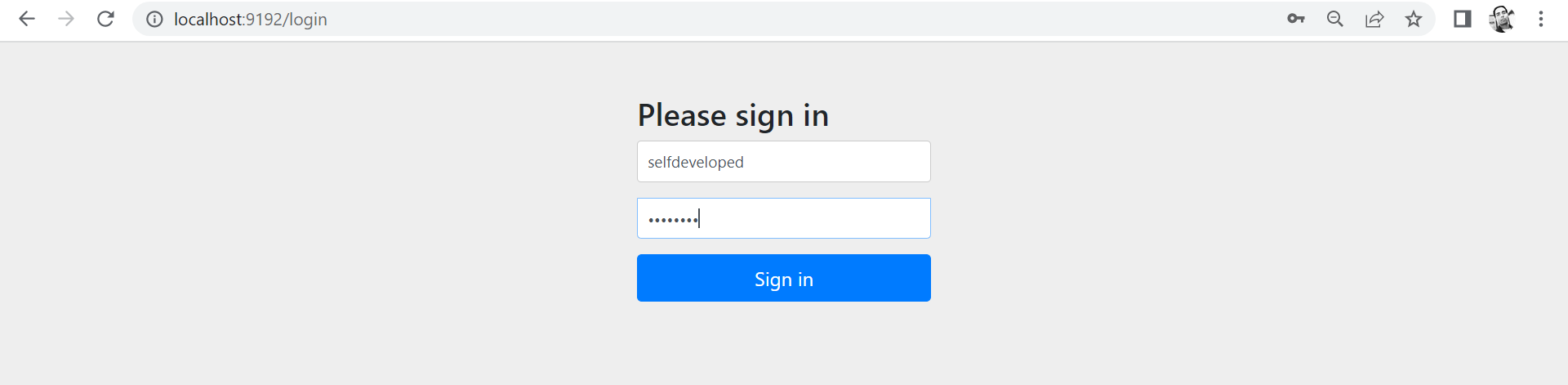


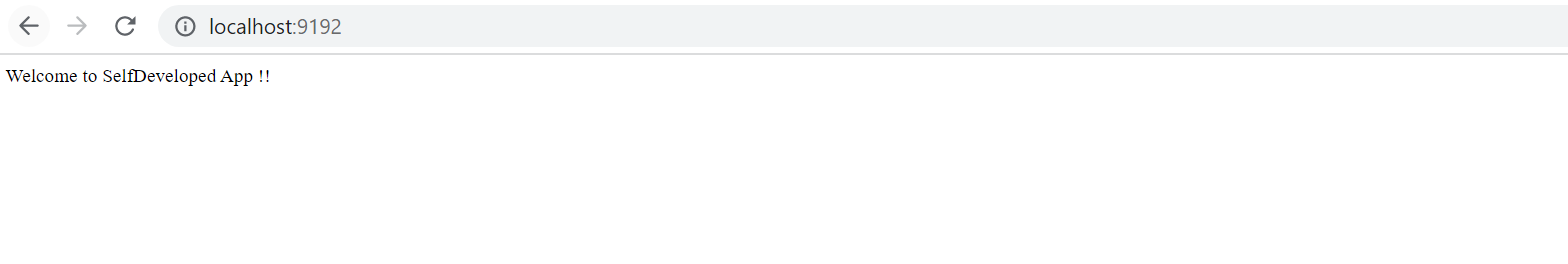
Let’s start our application. So, our application is up on port 9192.

<http://localhost:9192/>



Here we need to provide username and password. So, before that lets verify what are the User objects, we have added into our DB. We have added in our main class.





If u are using spring 2.x or version greater than 2. Then u need to create one Bean Object **PasswordEncoder**.



**Now we are good we have successfully implemented the Spring Security in our application.**

**But our Intention is not to use this spring security we need to use jwt. It means once user will send request with username and password by considering these 2 fields, we want to generate one web token. So that going forward user can directly pass the web token as part of Request Header for Authorization. Which means user no need to pass the username and password in each and every request he can directly play with the web token.**

**Now how we can generate JSON Web Token?**

So, for that we need to add one jwt libraries. Once we add that library we need to create one Utility class where we can write logic to create token or to validate token.

**<dependency>**

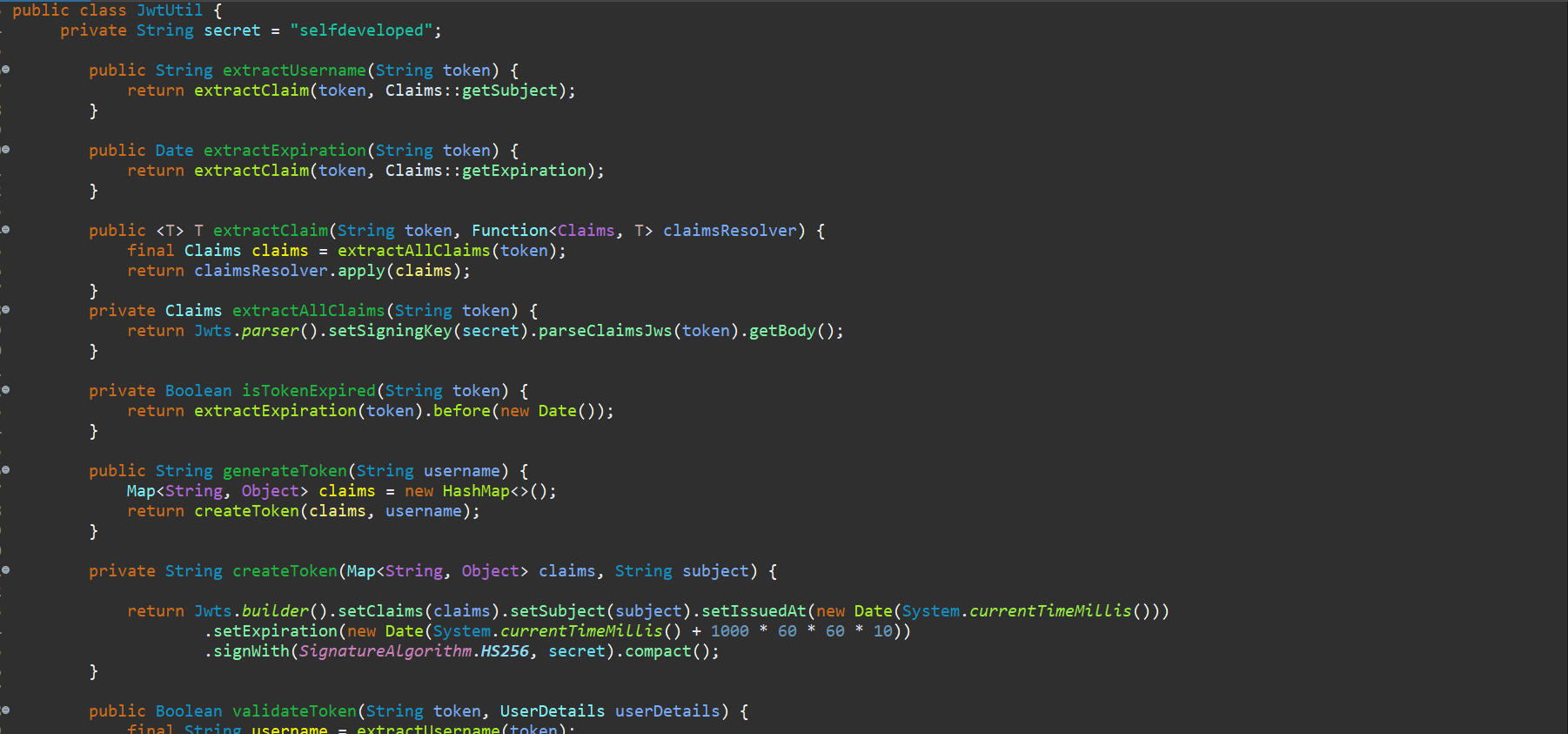
**<groupId>io.jsonwebtoken</groupId>**

**<artifactId>jjwt</artifactId>**

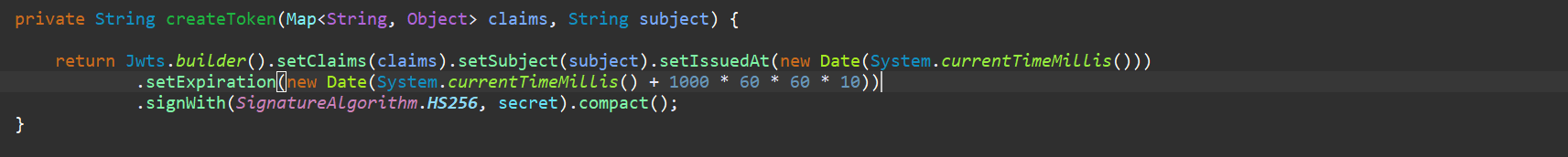
**<version>0.9.1</version>**

**</dependency>**

Let’s write one Utility class. Already I have the template so that I can explained you.



So if u can see here there is a method called **generateToken (String username)**… the method argument is username. So, 1st time when user will send username and password as part of Request Header, we will fetch the username and based on username we will create the token. So here if you come into generateToken() method, we are giving one Empty Map. So, if u go inside the **createToken**() method



We are using the jwt libraries. We are setting claims, so claims are nothing our empty Map. And we are setting the subject, subject is our username, and then we are setting the issued date, when we issued that token, which is the current date, then we are setting the expired limit date for specific token. So, we set the limit for 10 hours. So, u can see here the date when we issued the token, it will be available for next 10 hours. .setExpiration(new Date(System.*currentTimeMillis*() + 1000 \* 60 \* 60 \* 10)).

And next is sign with algorithm, which is **HS256** we are using, signWith (*SignatureAlgorithm*.***HS256***, secret) and we are passing secret variable as “selfdeveloped”.

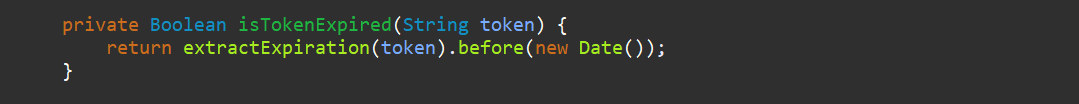
So, if we are going inside SignatureAlgorithm, then there are many algorithm ….



So, this is how we are creating a token. Then we are validating token here…

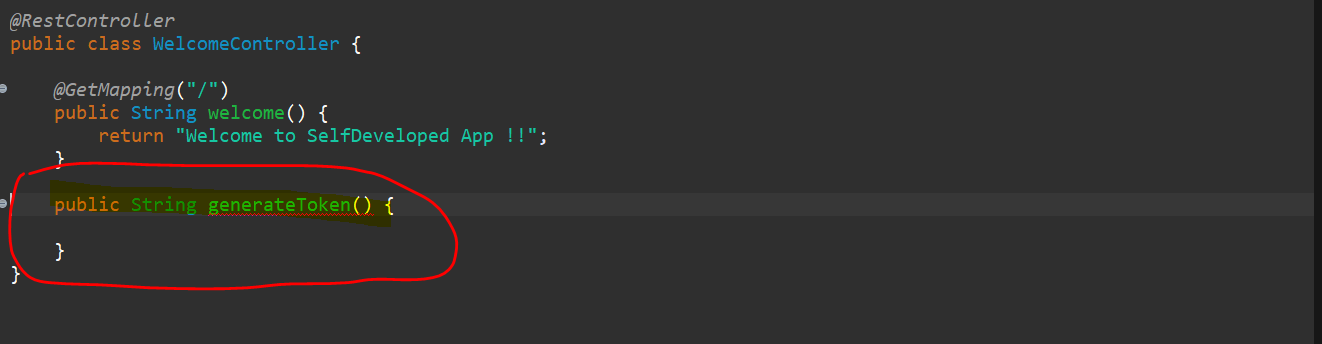


We are giving the token and we are giving the UserDetails. UserDetails is nothing your username and password. Then **extractUserName** () which will extract username from token. Basically, token will be generated in form of encrypted String. From that string we extract the username and password. Then we are validating the username and verifying whether the token is expired or not. So if we ll go inside the method ….



Its invoking extractExpiration(token), if it is created before new date. It means the limit which we set for next 10 hours. If the token is created before 10 hours, then return true else return false. Similarly, there are few methods, but we are going to use **createToken ()** and **validateToken ()** method.

So, lets create one rest endpoint which will take username and password and generate json web token for us. So just go to the controller,

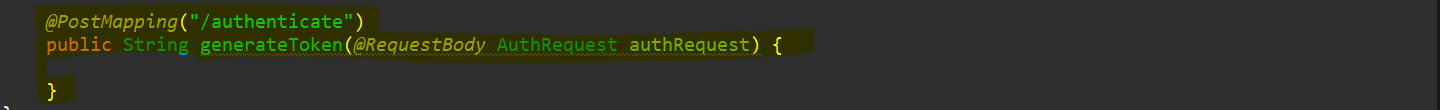


And we need to give username and password, for that what I will do. I will just create one Domain class, or we can say DTO.

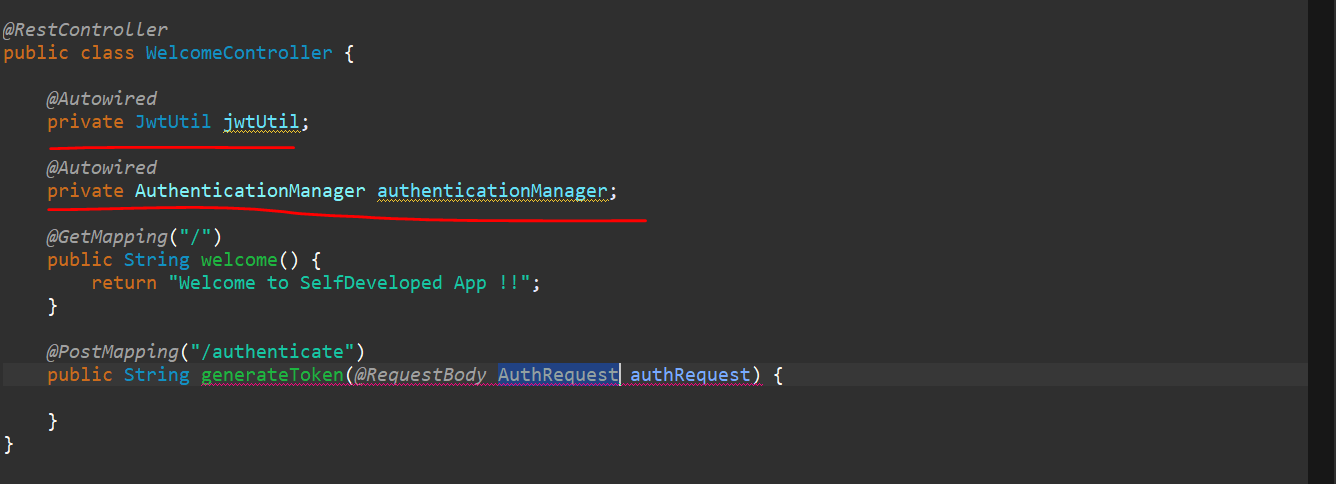
I can give the name **AuthRequest**



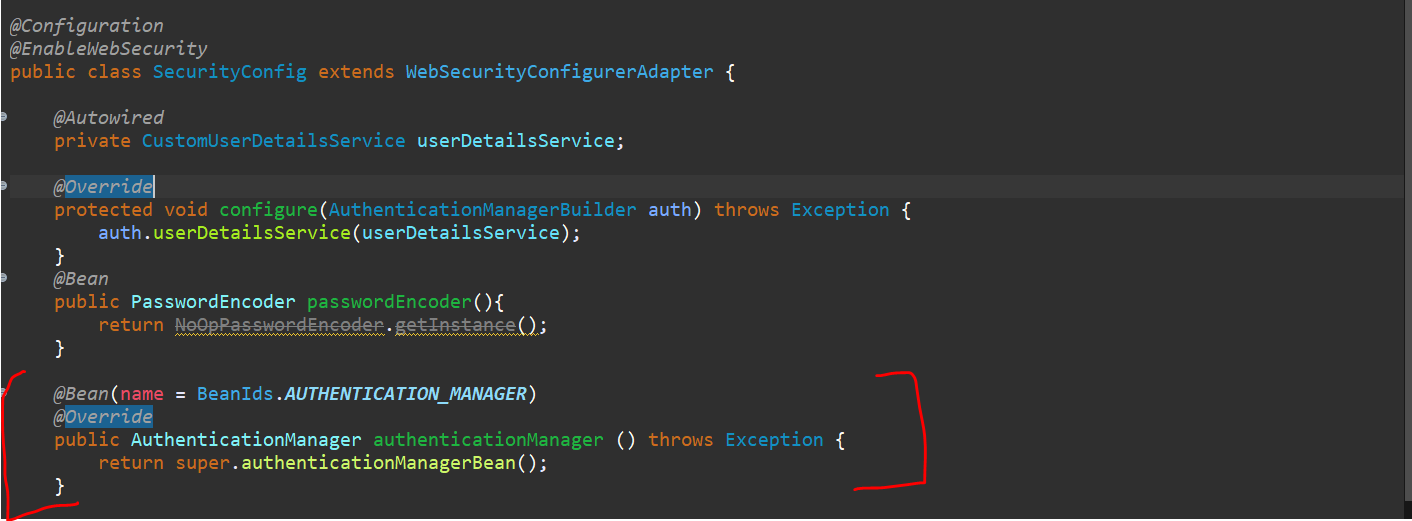
Now go to the controller and pass this as a method argument.



So once the User will give Request with username and password then we just want to invoke **JWTUtil** to generate token. So, I just want to inject the JWTUtil and we need one Bean which is **AuthenticationManager** to authenticate username and password.



And we want to create a Bean with same name in **AuthenticationManager** in our config file.



Then go to your Controller, we can directly use the AuthenticationManager and. **authenticate ()** and here we need to give the username and password which we get from the Authentication Request. and there is an in-build class new **UsernamePasswordAuthenticationToken,** just give your username and password here. So, this particular line will validate the username and password if the authentication is success then only generate the token so I ll put this in try catch bcz if any exception is there then I don’t want to proceed at all and if no exception then please go ahead and generate the token.



Let me tell what am I doing here, we just take the help of **AuthenticationManager** to authenticate username and password, so we gave username and password what we are getting from the user request so if this is succeed then only create a jwt token for us. Now this will give us the encrypted string which contains username and password in encrypted format.

So now we need to inform to spring framework If request is coming with /authenticate then don’t apply the security for this specific method. Bcz if user will give /authenticate

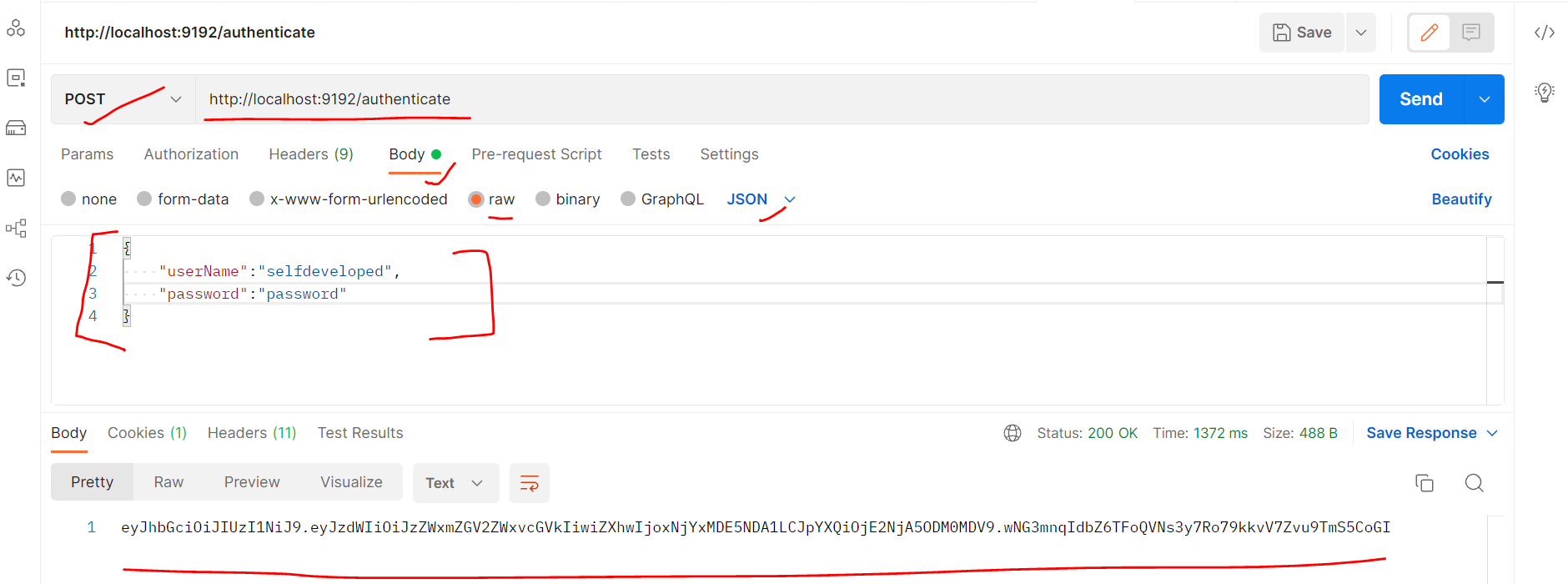
This url and user will give username and pwd we want to generate one token. As we enable the spring security so spring security is enables for all the rest endpoint, so we want to disable for specific endpoint.   
 so go to config class and override one **configure** () method.

So for this specific endpoint /authenticate we just permit all the Request and any other request u are getting it we are just authenticating it.



Now let’s start our application to verify either we are able to generate web token or not.

Now go to postman hit **9192** with endpoint **/authenticate** to get encrypted string which is nothing just a web token. And change the body. So, first time user needs to pass the username and password.

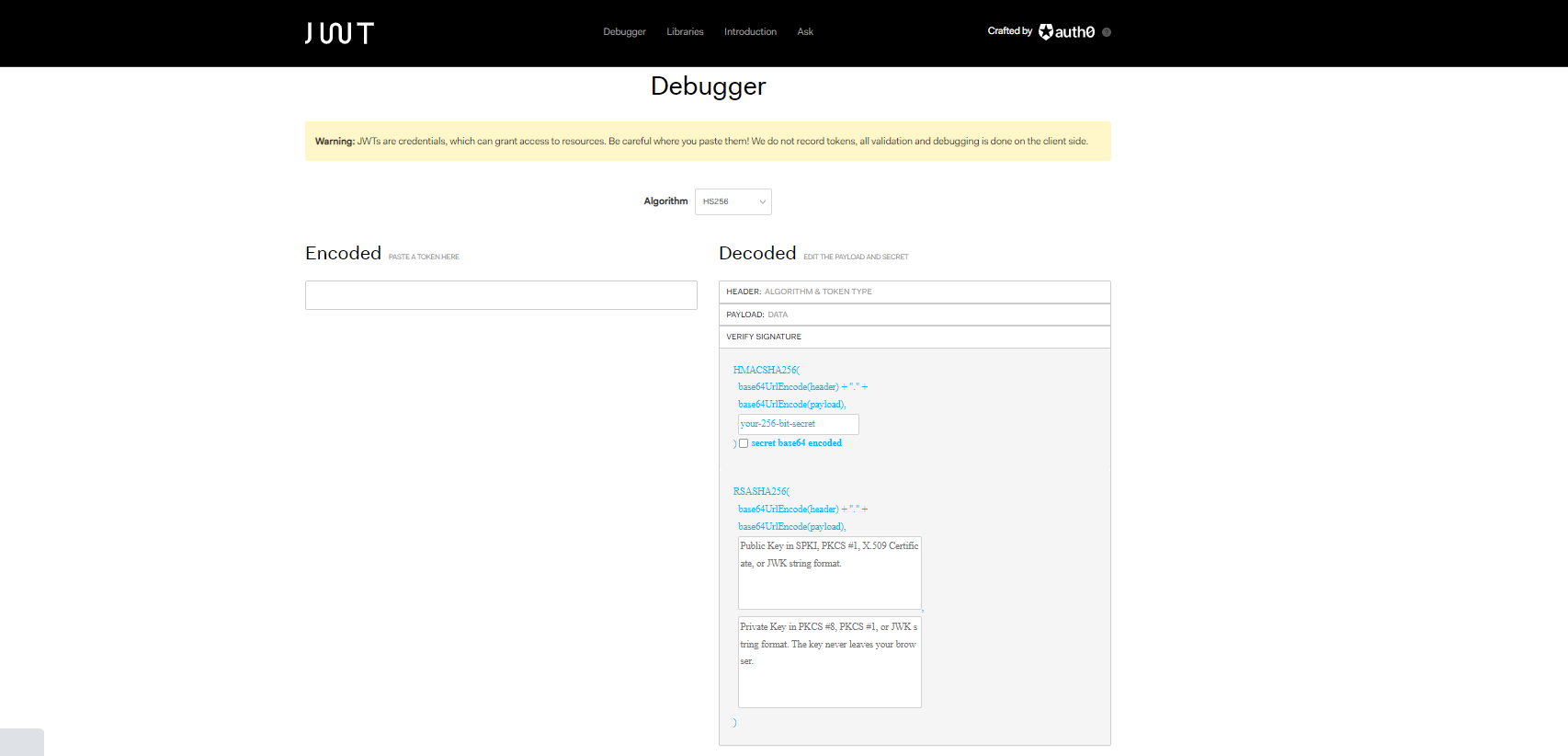


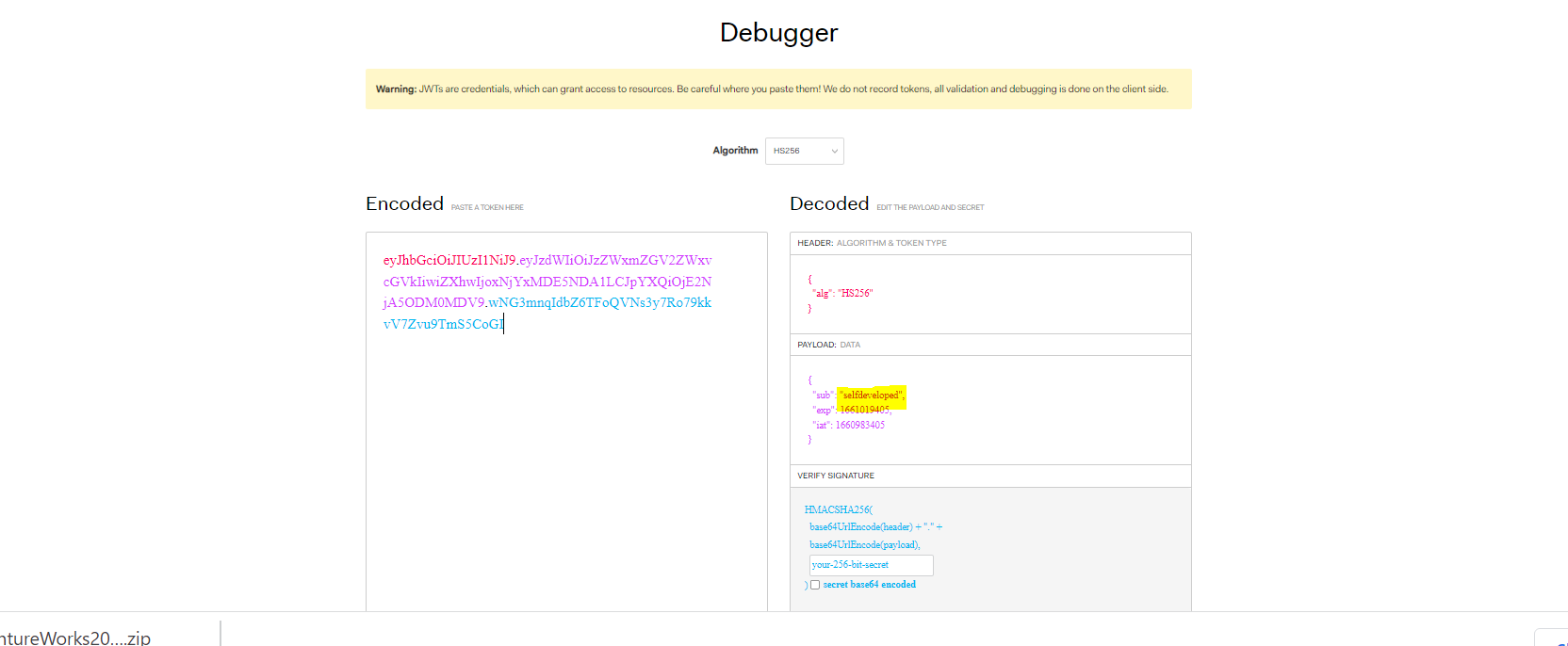
So, if u observed we can see one encrypted string.

**eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzZWxmZGV2ZWxvcGVkIiwiZXhwIjoxNjYxMDE5NDA1LCJpYXQiOjE2NjA5ODM0MDV9.wNG3mnqIdbZ6TFoQVNs3y7Ro79kkvV7Zvu9TmS5CoGI**

so, what we can do let’s copy this string and extract it. So, we can understand what all the fields are presents inside tis encrypted string.

So, go to <http://jwt.io> and copy paste this…



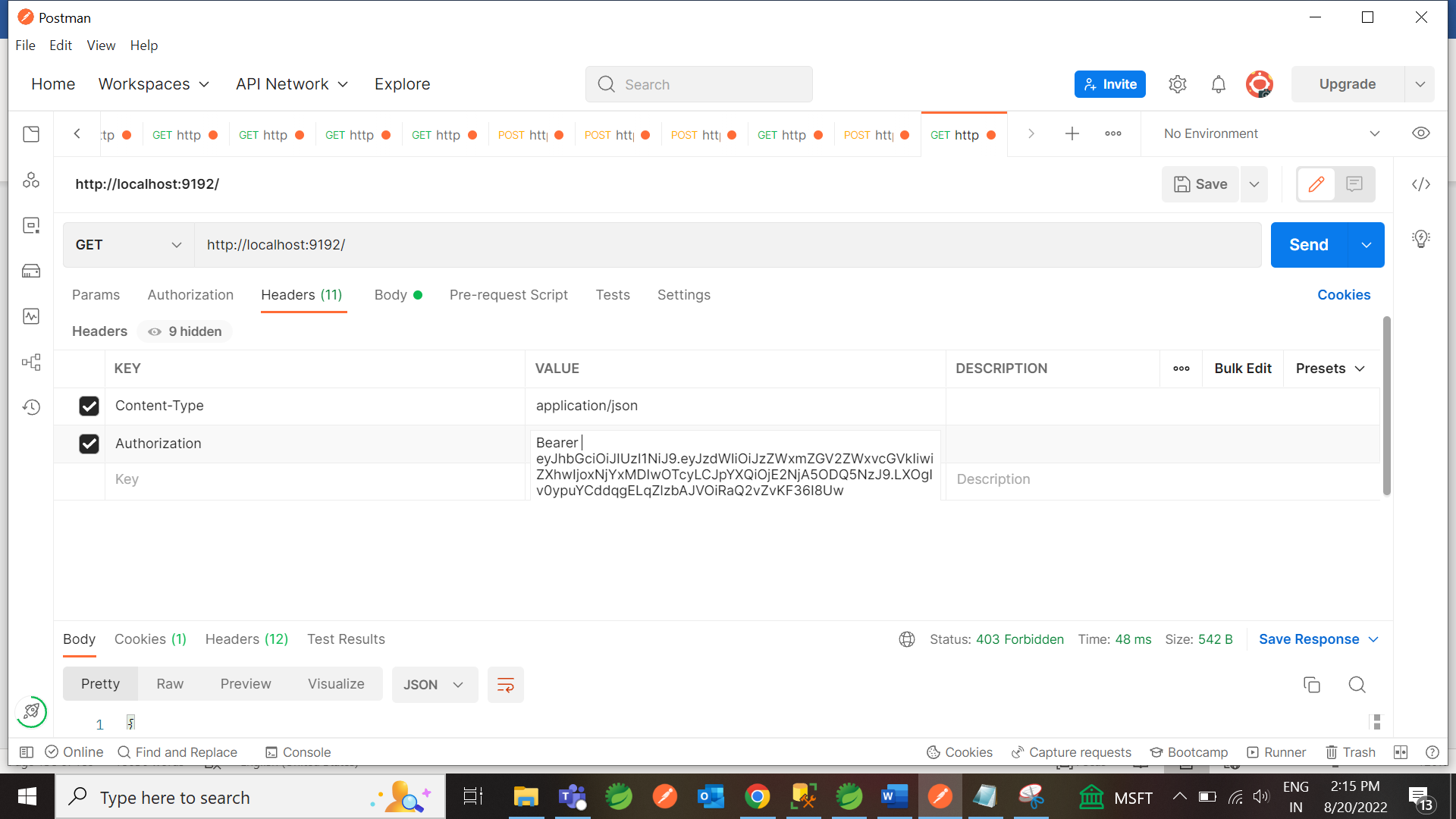


If we see there are 3 parts **Headers**, **payload** and **verify signature**. The header showing our algorithm which we are using that is **HS256.** And subject is nothing our userdetails and token expire date and time which is 10 hours as we configure in our jwt util.

So now let’s use this token to access our other endpoint. And other endpoint which we have is /(slash) which is default url.

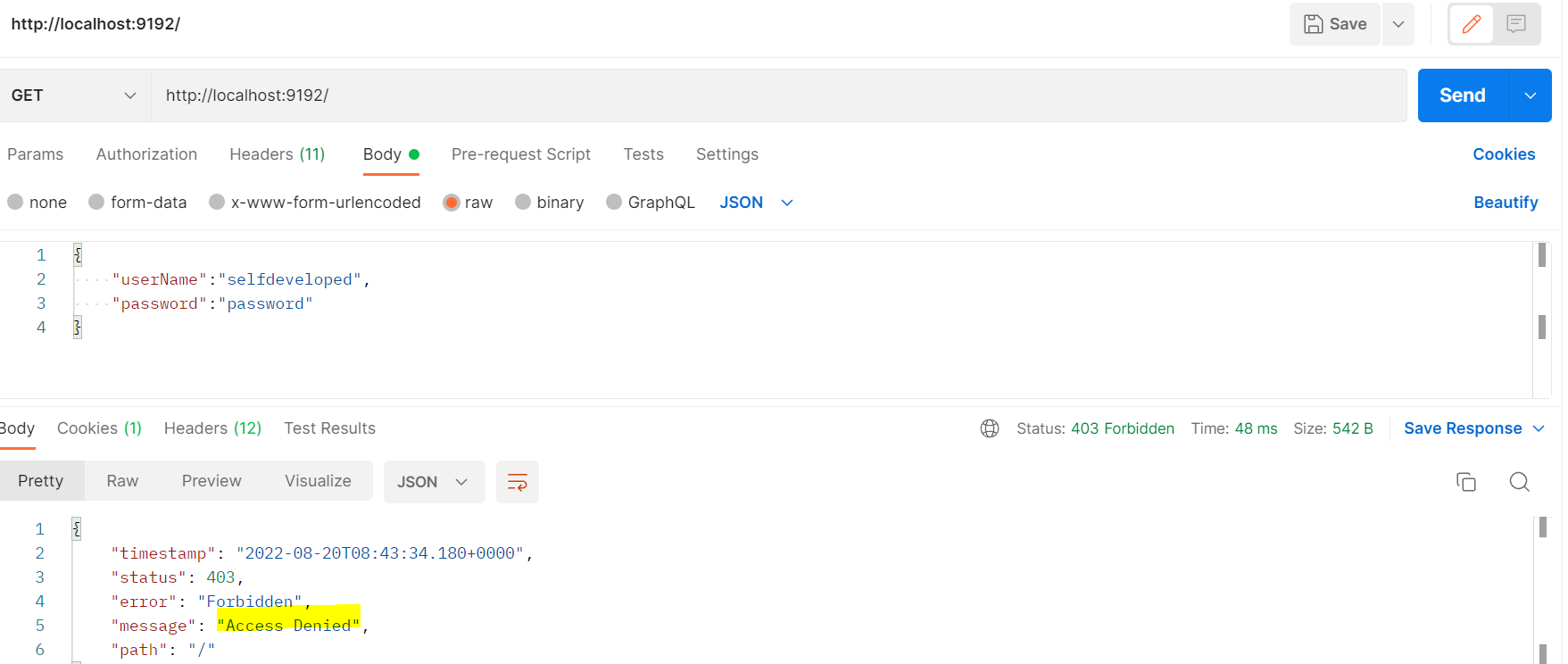
Endpoint- <http://localhost:9192/>

Come to our postman header-



Give key as Authorization and value as **Bearer** <**SPACE>** **Token.**

Now let’s try to access the endpoint it will not work



We are getting **403 Access Denied**.

It means spring boot didn’t understand what we are giving as part of authorization header. So, for that we need to tell to spring boot please get this string and extract and get the username and password. Then authenticate the user if user authentication is succeeded then allow him to access the endpoint.

So, for that what we can do we can one more additional layer so before my request will go to the controller endpoint, I want to authorize these users, or I want to validate these tokens.

So, for that we can write one filter…

Let’s create one more package called filter…

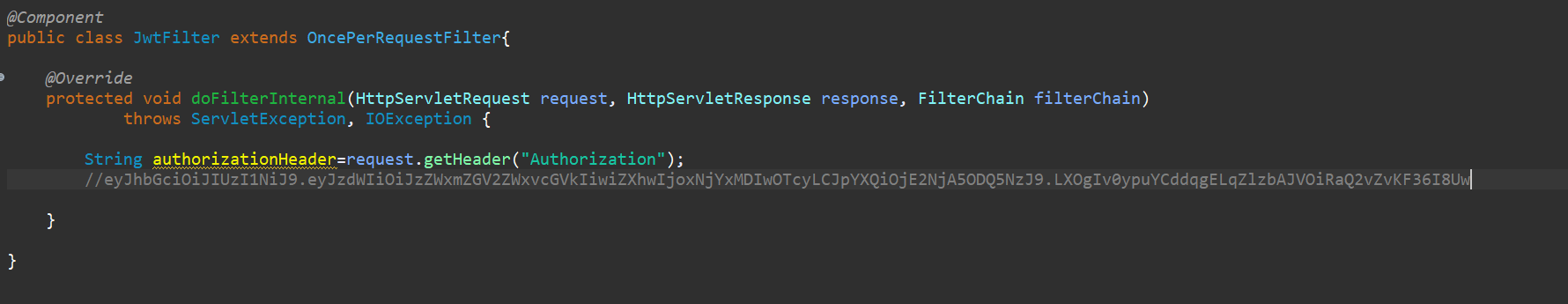


So, here we need to write the logic to authenticate the user and to validate the token.

So, as we have **OncePerRequestFilter,** so it will execute every request for once**.** Then need to override method **doFilterInternal ().**



First annotate this class with **@Component.**  Now first we need to extract Authorization Headers from this **HttpServletRequest**. Bcz this “Authorization” as we mentioned as a key Header in postman.



**Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzZWxmZGV2ZWxvcGVkIiwiZXhwIjoxNjYxMDIwOTcyLCJpYXQiOjE2NjA5ODQ5NzJ9.LXOgIv0ypuYCddqgELqZlzbAJVOiRaQ2vZvKF36I8Uw**

So, if u observe into this entire string first part is **Bearer,** which is the namespace right and the rest part is our **json web token.**

So, in this we just need to extract json web token. So, before that lets validate either this **AuthorizationHeader** first contains this key or not or this is null or not. If this is not null and this contains Bearer as a Key, then get that value.



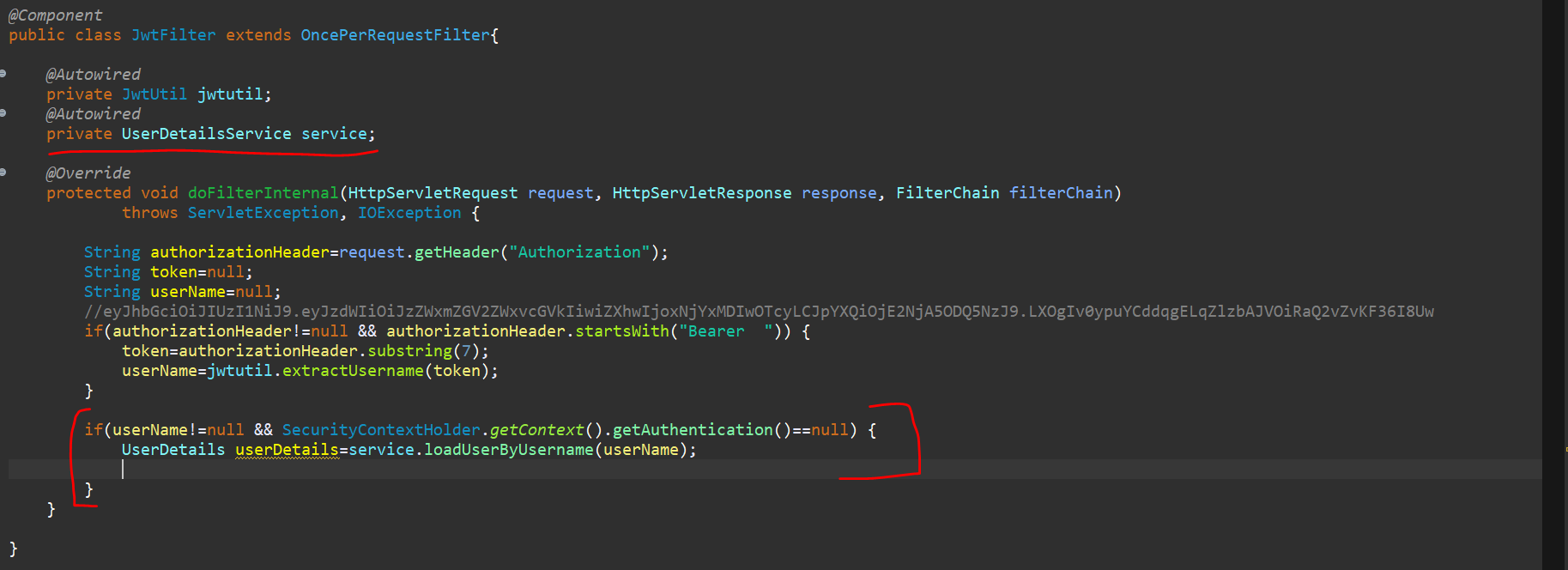
Now from this token we just want to extract the username and password. So, if u remember already, we have written the logic to retrieve username from the token in **JwtUtil.**



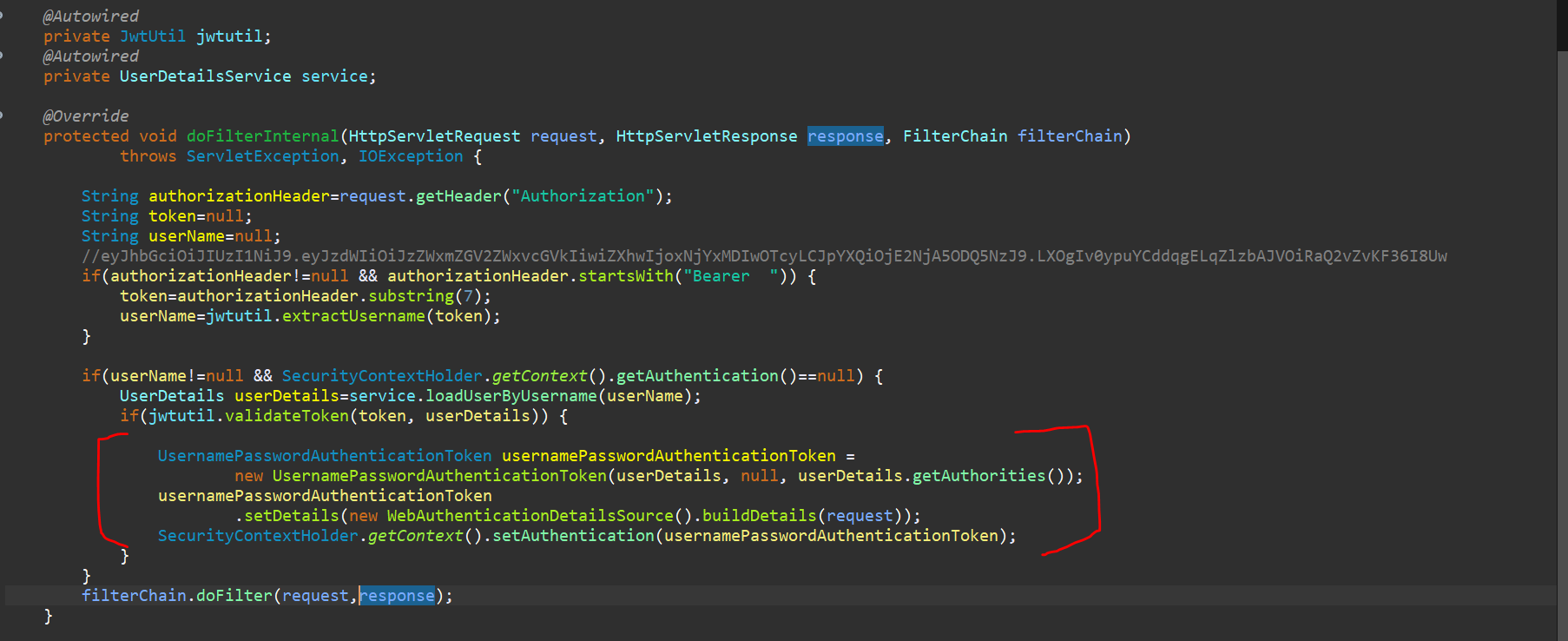
Again, we need to validate the username and security context holder. So, for that let me add one more if statement. Where if **username! =null** and **SecurityContextHolder.getContext(). getAuthorization()==null**

If the userName! = null then give this userName to UserDetailsService and get the UserDetails Object.

So, for that let me Inject the UserDetailsService .

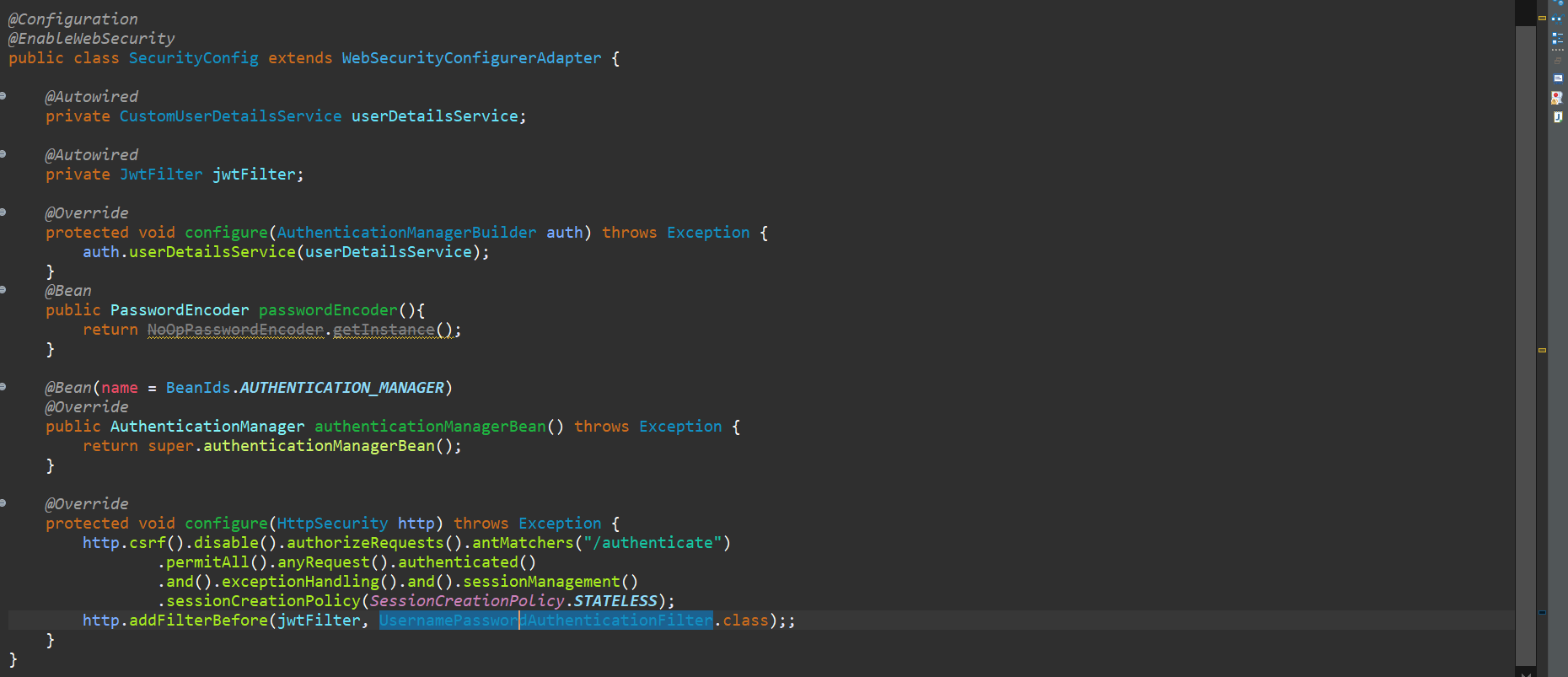


So, we have userName with us, we have Token with us, and we have UserDetails with us. So now we need to validate this token. So, let me add it here…



Basically, what we are doing inside if block, we are just validating the token if the token is valid then we are validating the UserDetails…. if UserDetails are valid then just we are setting the SecurityContext. So now we need to call the **doFilter()** method

Now what we need to do we need to register this **jwtFilter** in **SecurityConfig.**  So first we need to Inject **JwtFilter** inside SecurityConfig. Now we need to enable the SessionPolicy which is Stateless as I already mentioned this JWT follows **Stateless Authentication Mechanism**. So, we need to enable that as well as we need to enable the JwtFilter. So go to the configure () method and enable the session policy here.

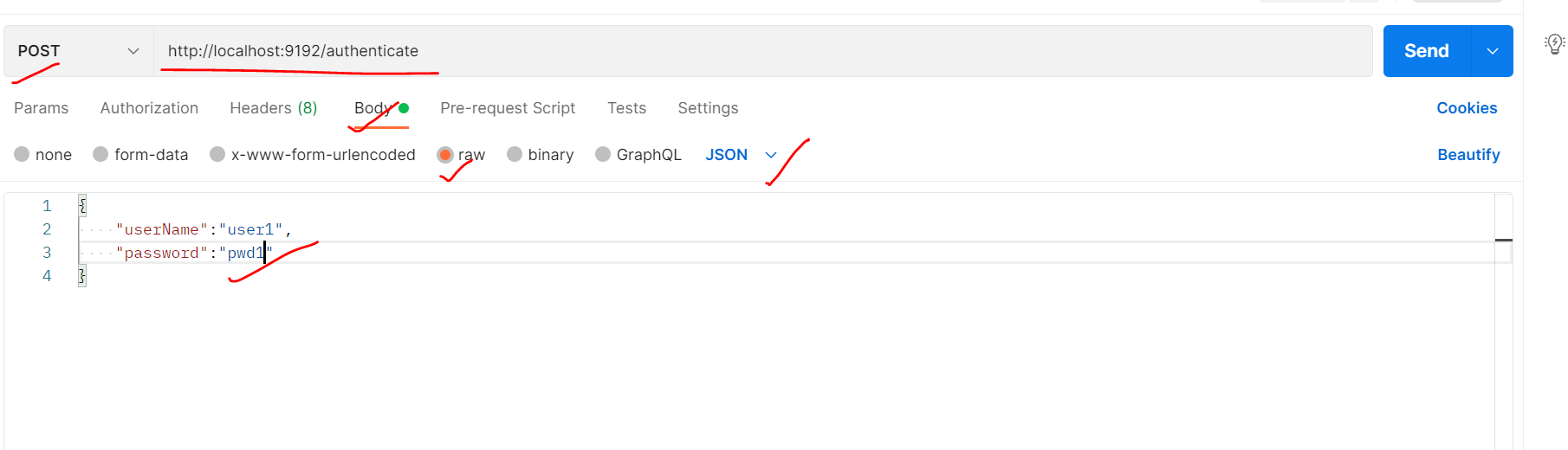


So, here we have just enabled our session creation policy which is STATELESS and we just Register our Filter. So now I think we are going so let’s run our application.

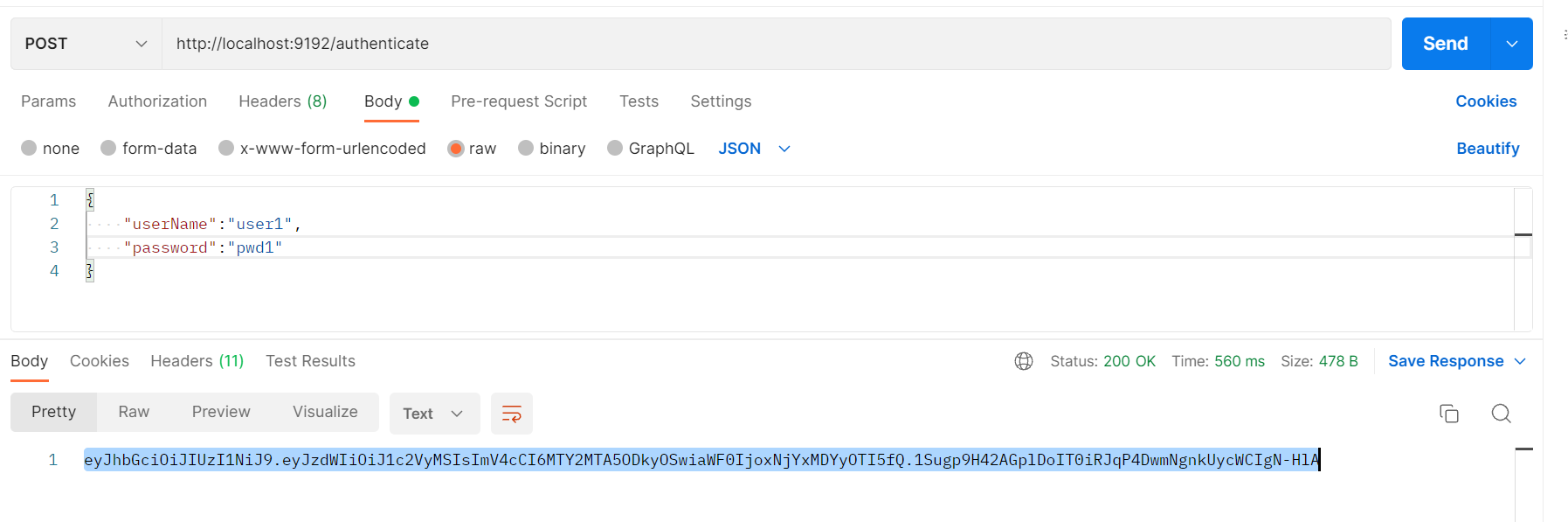
So, we will hit the first endpoint **/authenticate**, we will re-generate the token. Then we will try to access our **/ endpoint.**

So, from db we are going to pass username and password.

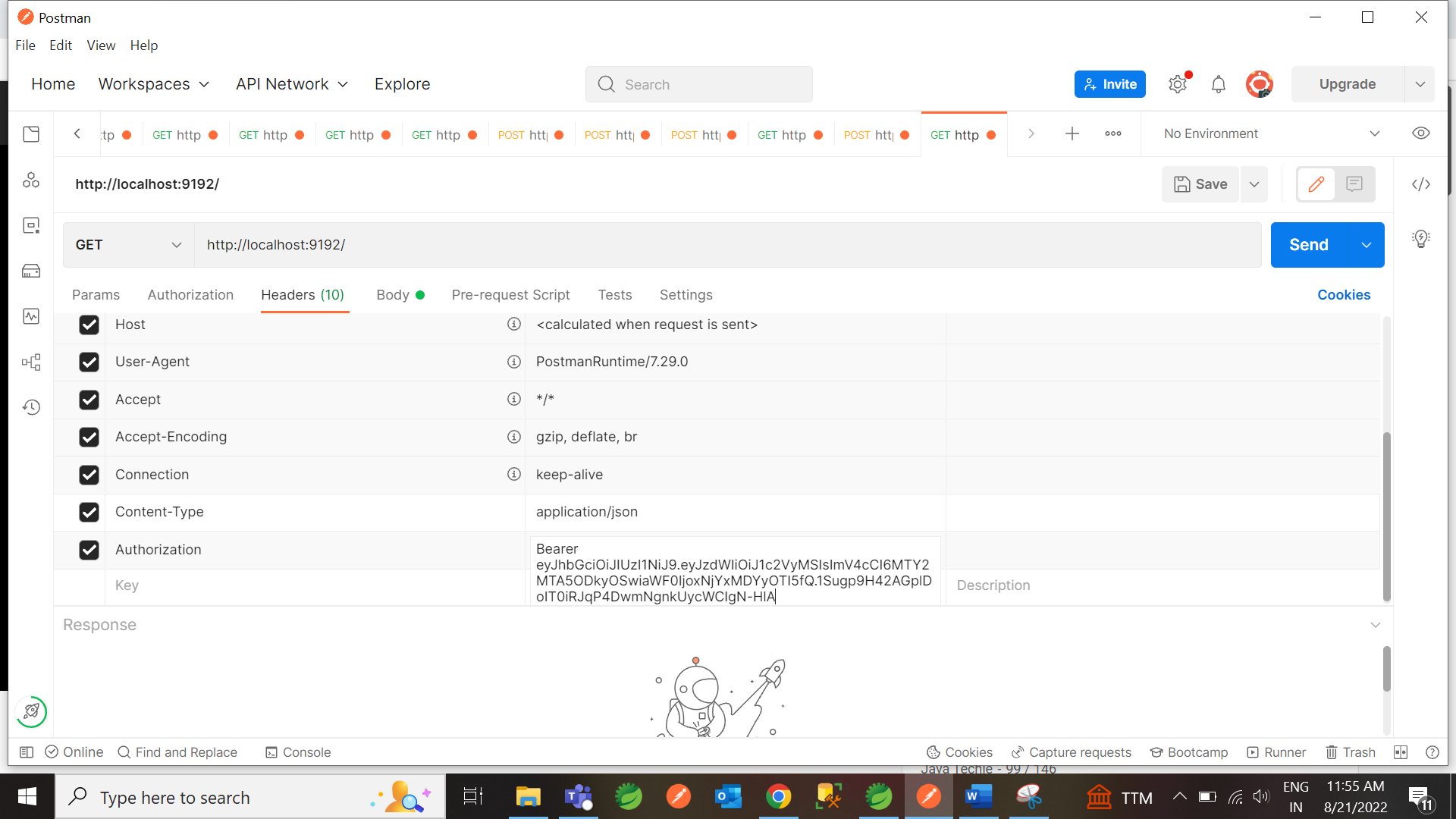


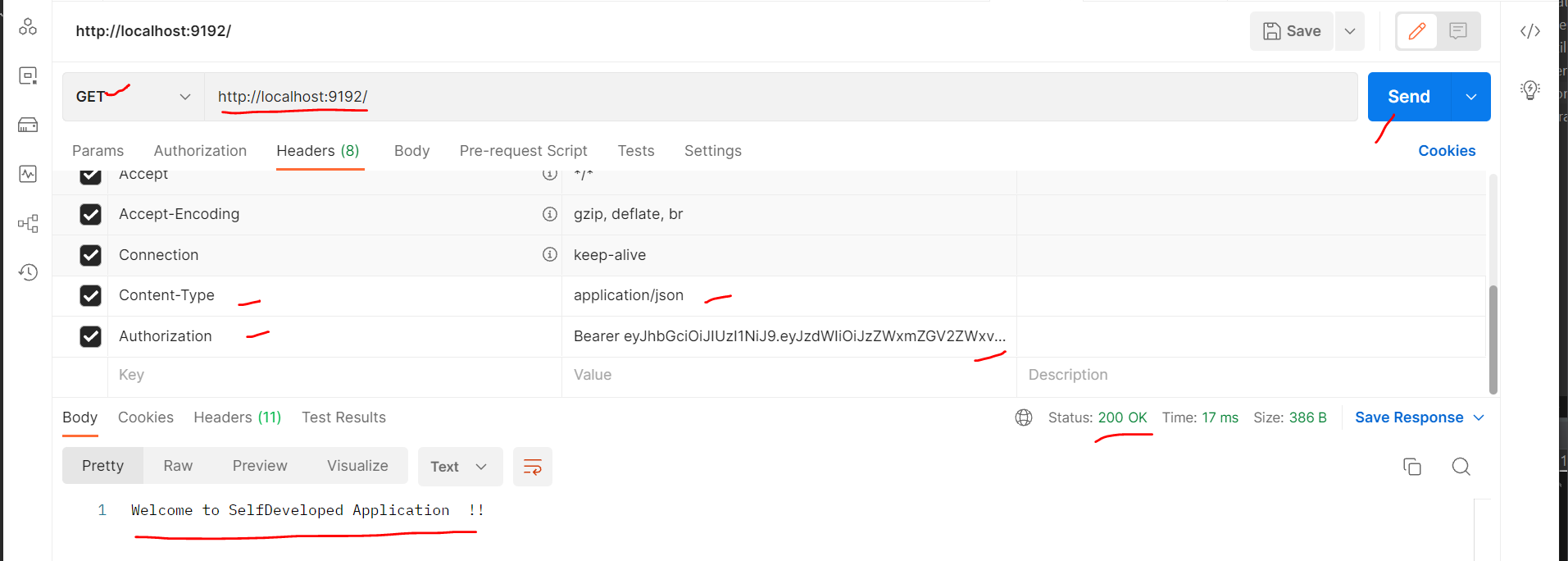


Generated-> **eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMSIsImV4cCI6MTY2MTA5ODkyOSwiaWF0IjoxNjYxMDYyOTI5fQ.1Sugp9H42AGplDoIT0iRJqP4DwmNgnkUycWCIgN-HlA**

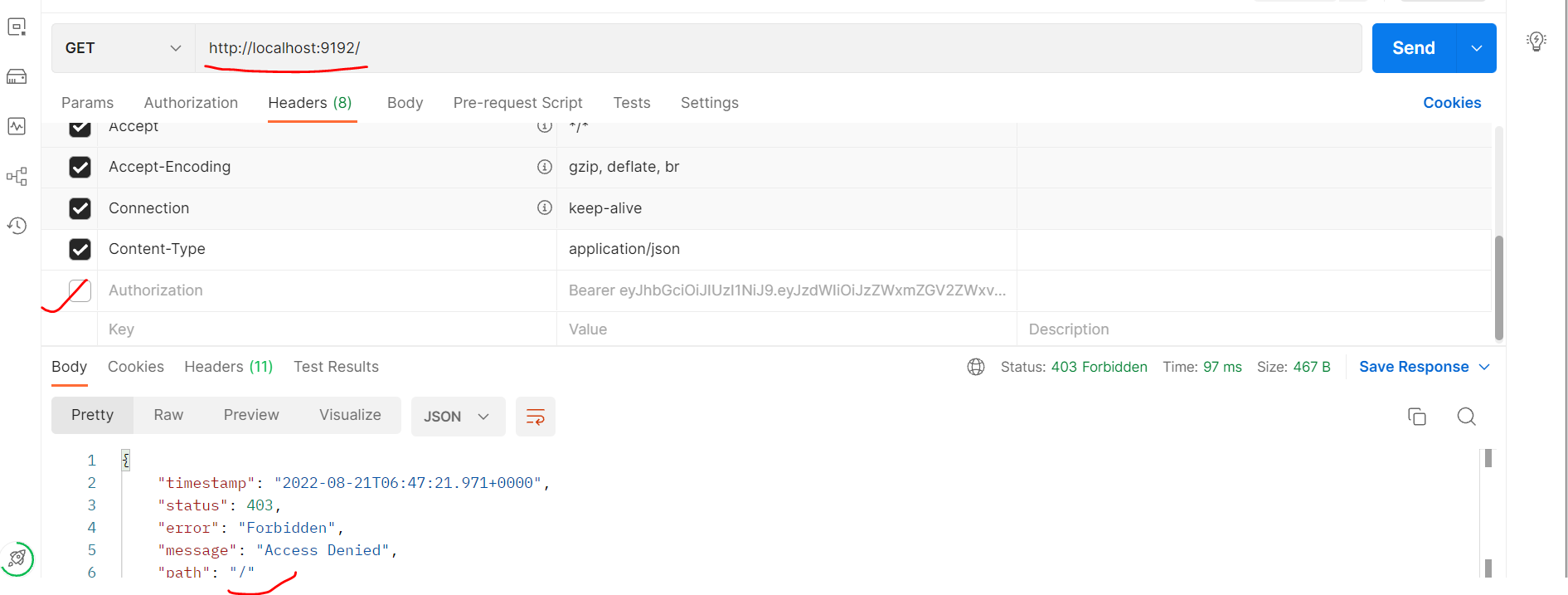


Now copy token and go to **GET api go to header and change the token**





Now just Disable the Authorization tab then it will give **403** **Forbidden Access**, **Access Denied**…



**Hence it proved it following the Stateless Authentication. means our Input is not stored in any server Memory or Cookies. So, this is how we can play with JWT and in Real Time Scenario we can Implement in our Application.**