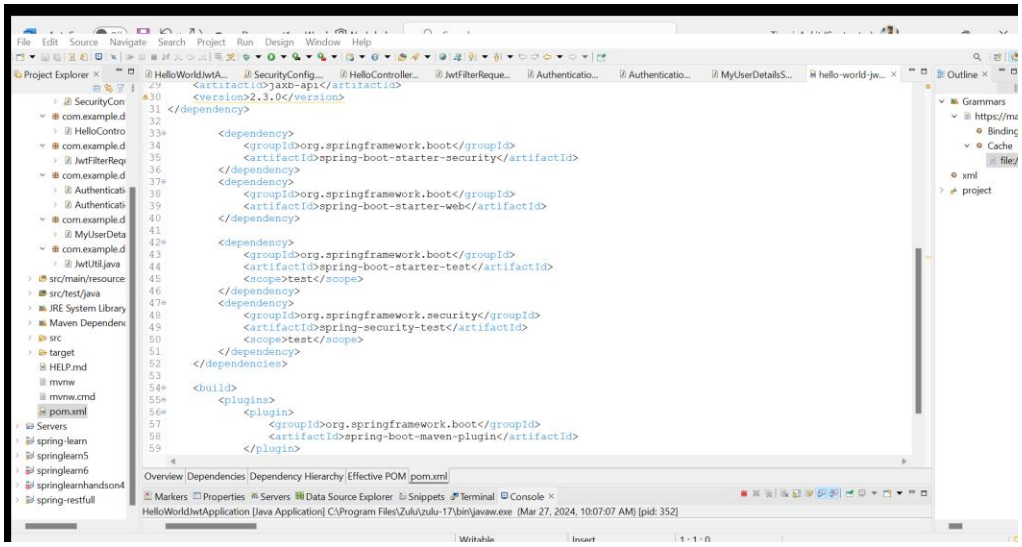
Securing RESTful Web Services with Spring Security

Follow steps below to secure all web services using Spring Security:

- Open spring-learn project in Eclipse
- Include spring security related libraries by adding the below dependency in pom.xml

```
<dependency>

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

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

</dependency>
```



```
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
```

```
HelloWorldJwtApplication [Java Application] C:\Program Files\Zulu\zulu-17\bin\javaw.exe (Mar 27, 2024, 10:07:07 AM) [pid: 352]
```

- Rebuild the project in command line using mvn clean package command (ensure to include proxy details in mvn command).
- To ensure the new libraries are enabled in Eclipse, right click the project and select Maven > Update Project
- Create a new package 'com.cognizant.springlearn.security'
- Create a new class SecurityConfig in the new package created above which extends from WebSecurityConfigurerAdapter
- Include annotations @Configuration and @EnableWebSecurity at class level
- Import appropriate classes using Ctrl + Shift + O

```
package com.example.demo.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.Authen
ticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityC
onfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticatio
nFilter;

import com.example.demo.filter.JwtFilterRequest;
import com.example.demo.service.MyUserDetailsService;

@EnableWebSecurity
public class SecurityConfig  extends WebSecurityConfigurerAdapter
{

	@Autowired
	MyUserDetailsService user;
	@Autowired
	JwtFilterRequest req;
	@Override
	protected void configure(AuthenticationManagerBuilder auth)  throws
Exception
	{
		//to provide access for the user
		auth.userDetailsService(user);
	}
	//to skip encoding of password
	@Bean
	public PasswordEncoder passwordEncoder()
	{
		return NoOpPasswordEncoder.getInstance();
	}
	@Override
	protected void configure(HttpSecurity http)throws Exception
	{

	http.csrf().disable().authorizeRequests().antMatchers("/authenticate").
permitAll().
		anyRequest().authenticated()
```

```
            .and().sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
            http.addFilterBefore(req,
UsernamePasswordAuthenticationFilter.class);
      }
      @Bean
      public AuthenticationManager authenticationManagerBean()throws
Exception
      {
            return super.authenticationManager();
      }
}
```

Create authentication service that returns JWT

As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.

Ideally when the below curl command is executed that calls the new authentication service, the token should be responded. Kindly note that the credentials are passed using -u option.

Request

```
curl -s -u user:pwd http://localhost:8090/authenticate
```

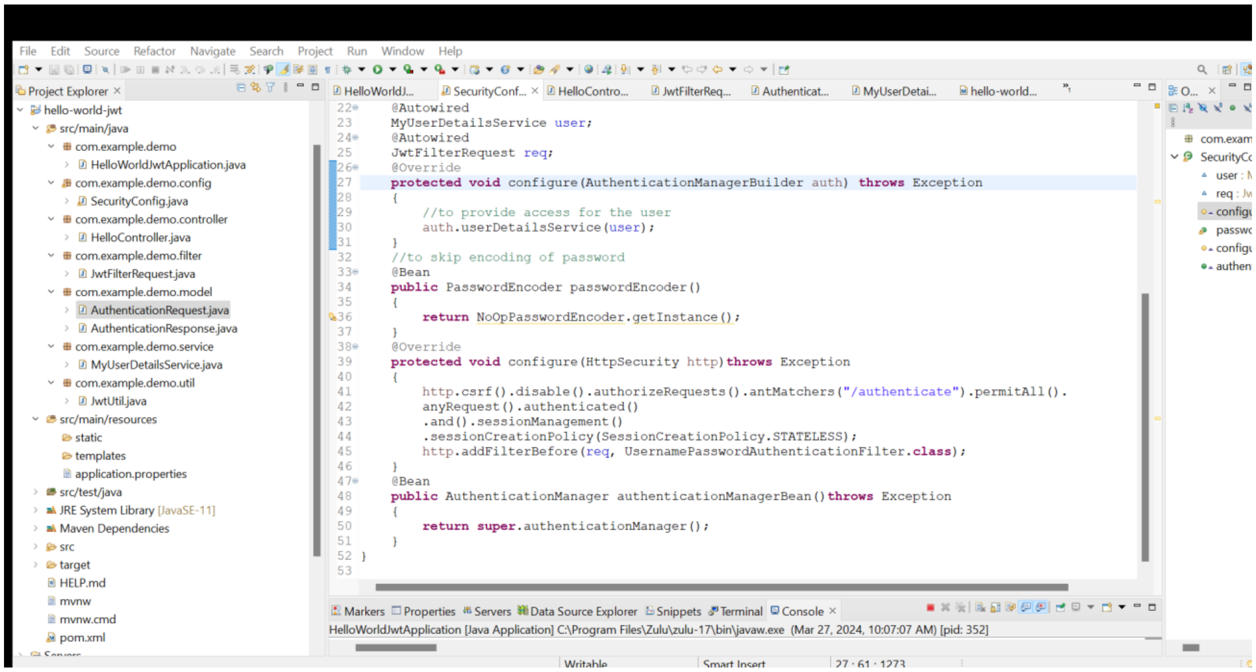
Response

```
{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1Nz
AzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}
```

This can be incorporated as three major steps:

- Create authentication controller and configure it in SecurityConfig
- Read Authorization header and decode the username and password
- Generate token based on the user retrieved in the previous step

Let incorporate the above as separate hands on exercises.



```
    @Autowired
    MyUserDetailsService user;
    @Autowired
    JwtFilterRequest req;
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception
    {
        //to provide access for the user
        auth.userDetailsService(user);
    }
    //to skip encoding of password
    @Bean
    public PasswordEncoder passwordEncoder()
    {
        return NoOpPasswordEncoder.getInstance();
    }
    @Override
    protected void configure(HttpSecurity http)throws Exception
    {
        http.csrf().disable().authorizeRequests().antMatchers("/authenticate").permitAll().
        anyRequest().authenticated()
        .and().sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        http.addFilterBefore(req, UsernamePasswordAuthenticationFilter.class);
    }
    @Bean
    public AuthenticationManager authenticationManager()throws Exception
    {
        return super.authenticationManager();
    }
}
```

```
HelloWorldJwtApplication [Java Application] C:\Program Files\Zulu\zulu-17\bin\javaw.exe (Mar 27, 2024, 10:07:07 AM) [pid: 352]
```

Create authentication controller and configure it in SecurityConfig

AuthenticationController.java

- Create new rest controller named AuthenticationController in controller package with @RestController annotation
- Include method authenticate with "/authenticate" as the URL with @GetMapping.
- To read the Authorization value from HTTP Header, include a parameter for authenticate method as specified below. Spring takes care of reading the Authorization value from HTTP Header and pass it as parameter.

```
@RequestHeader("Authorization") String authHeader
```

- The return type of this method should be Map<String, String>
- Include start and end logger in this method
- Include a debug log for displaying the authHeader parameter
- Create a new HashMap<String, String> and assign it to a map.
- Put a new item into the map with key as "token" and value as empty string.

SecurityConfig.java

- In the second configure method, include authenticate URL just after the countries URL defined earlier. Refer code below:

```
            .antMatchers("/countries").hasRole("USER")
            .antMatchers("/authenticate").hasAnyRole("USER", "ADMIN")
```

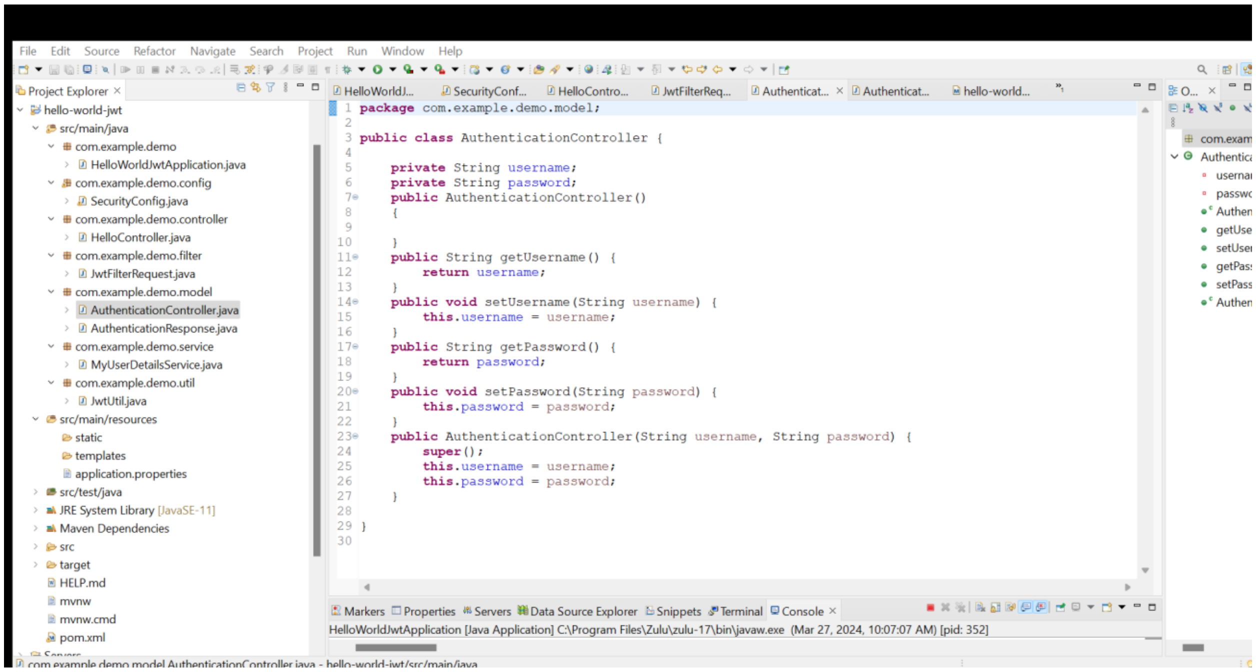
Read Authorization header and decode the username and password

Steps to read and decode header:

- Create a new private method in AuthenticationController with below method signatureGet the Base64 encoded text after "Basic "

- Decode it using the library available in Java 8 API. Refer code below

- The above call returns a byte array, which can be passed as parameter to string constructor to convert to string.
- Get the text until colon on the string created in previous step to get the user
- Return the user obtained in previous step
- Include appropriate debug logs within this method
- Invoke the getUser() method from authenticate method
- Execute the curl command used in the previous step and check the logs if the user information is obtained successfully.


File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
HelloWorldJ... SecurityConf... HelloContro... JwtFilterReq... Authenticat... Authenticat... hello-world...
Markers Properties Servers Data Source Explorer Snippets Terminal Console

```
hello-world-jwt
  src/main/java
    com.example.demo
      HelloWorldJwtApplication.java
    com.example.demo.config
      SecurityConfig.java
    com.example.demo.controller
      HelloController.java
    com.example.demo.filter
      JwtFilterRequest.java
    com.example.demo.model
      AuthenticationController.java
      AuthenticationResponse.java
    com.example.demo.service
      MyUserDetailsService.java
    com.example.demo.util
      JwtUtil.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-11]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

```java
package com.example.demo.model;

public class AuthenticationController {

    private String username;
    private String password;
    public AuthenticationController()
    {

    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public AuthenticationController(String username, String password) {
        super();
        this.username = username;
        this.password = password;
    }

}
```

```
HelloWorldJwtApplication [Java Application] C:\Program Files\Zulu\zulu-17\bin\javaw.exe (Mar 27, 2024, 10:07:07 AM) [pid: 352]
```

com.example.demo.model.AuthenticationController.java - hello-world-jwt/src/main/java

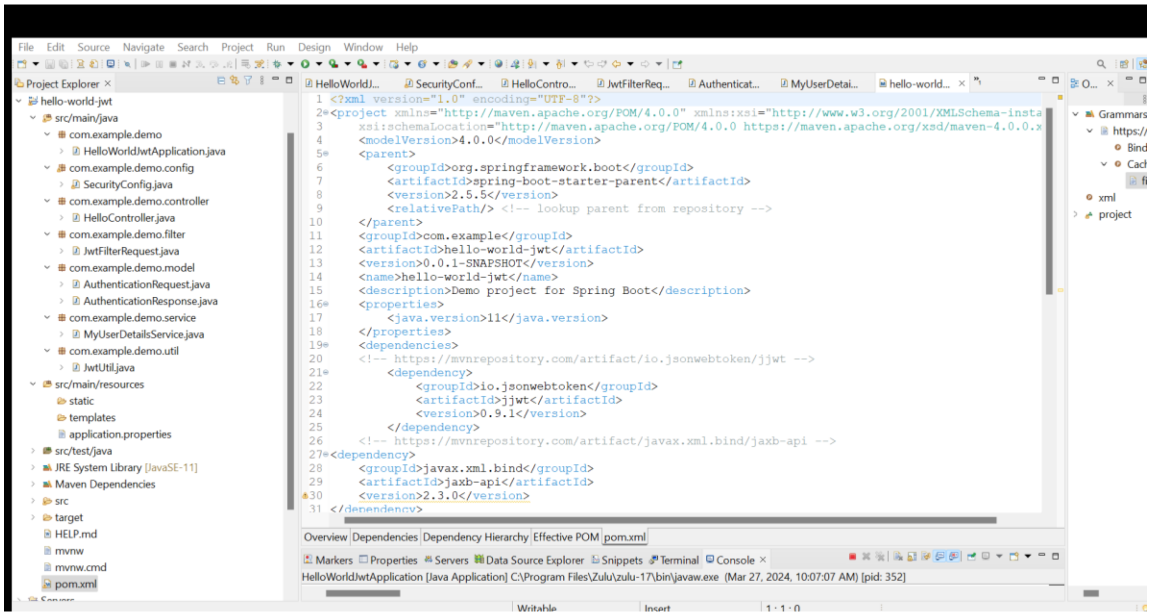
Generate token based on the user

Steps to generate token:

- Include JWT library by including the following maven dependency.

```
    <dependency>
       <groupId>io.jsonwebtoken</groupId>
       <artifactId>jjwt</artifactId>
       <version>0.9.0</version>
   </dependency>
```

- After inclusion in pom.xml, run the maven package command line and update the project in Eclipse. View the dependency tree and check if the library is added.



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.x
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.5</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>hello-world-jwt</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>hello-world-jwt</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>11</java.version>
    </properties>
    <dependencies>
    <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>
    <!-- https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api -->
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.0</version>
</dependency>
```

```
HelloWorldJwtApplication [Java Application] C:\Program Files\Zulu\zulu-17\bin\javaw.exe  (Mar 27, 2024, 10:07:07 AM) [pid: 352]
```

- Create a new method in AuthenticationController with below method signature:

```
private String generateJwt(String user)
```

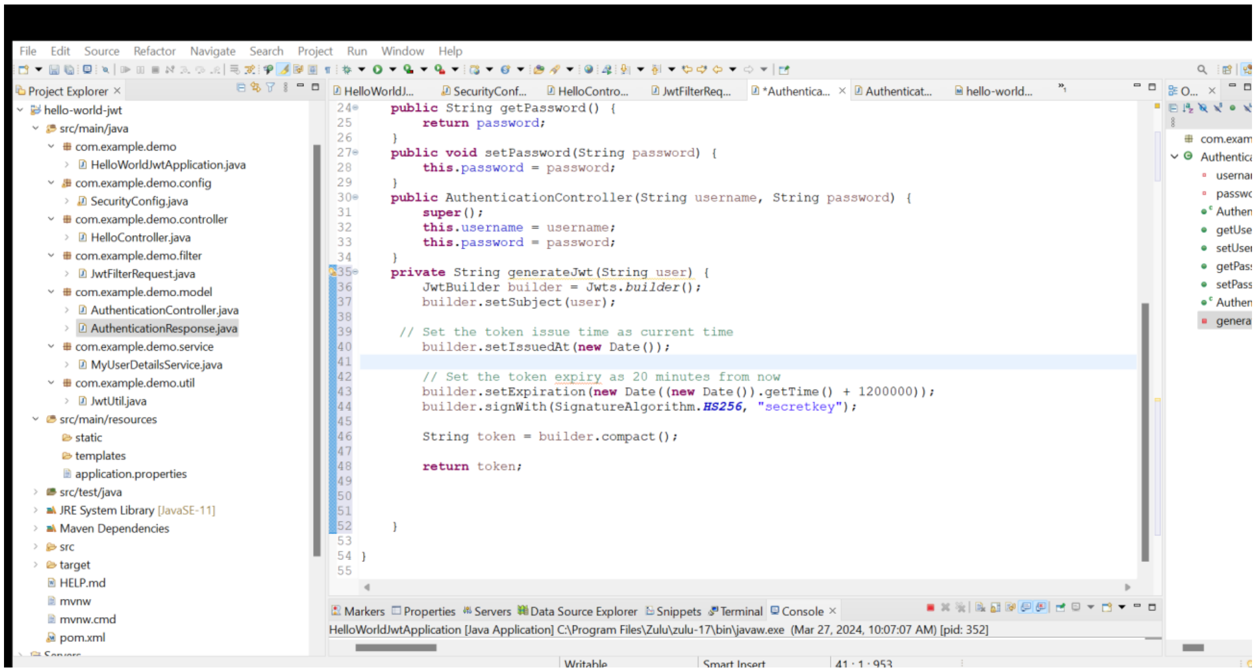
- Generate the token based on the code specified below.

```
        JwtBuilder builder = Jwts.builder();
        builder.setSubject(user);
        // Set the token issue time as current time
        builder.setIssuedAt(new Date());

        // Set the token expiry as 20 minutes from now
        builder.setExpiration(new Date((new Date()).getTime() + 1200000));
        builder.signWith(SignatureAlgorithm.HS256, "secretkey");

        String token = builder.compact();

        return token;
```

- Import reference for the above code

```
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
```

- Invoke this method from authenticate() method passing the user obtained from getUser() method.
- Add the token into the map using put method.
- Include appropriate logs
- Execute the curl command for authenticate and check if the generated token is returned.


File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
HelloWorldJ... SecurityConf... HelloContro... JwtFilterReq... *Authentica... Authenticat... hello-world...
Markers Properties Servers Data Source Explorer Snippets Terminal Console
Writable Smart Insert 41 : 1 : 953

```
hello-world-jwt
  src/main/java
    com.example.demo
      HelloWorldJwtApplication.java
    com.example.demo.config
      SecurityConfig.java
    com.example.demo.controller
      HelloController.java
    com.example.demo.filter
      JwtFilterRequest.java
    com.example.demo.model
      AuthenticationController.java
      AuthenticationResponse.java
    com.example.demo.service
      MyUserDetailsService.java
    com.example.demo.util
      JwtUtil.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-11]
  Maven Dependencies
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

```
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public AuthenticationController(String username, String password) {
        super();
        this.username = username;
        this.password = password;
    }
    private String generateJwt(String user) {
        JwtBuilder builder = Jwts.builder();
        builder.setSubject(user);

      // Set the token issue time as current time
        builder.setIssuedAt(new Date());

        // Set the token expiry as 20 minutes from now
        builder.setExpiration(new Date((new Date()).getTime() + 1200000));
        builder.signWith(SignatureAlgorithm.HS256, "secretkey");

        String token = builder.compact();

        return token;



    }

}
```

```
HelloWorldJwtApplication [Java Application] C:\Program Files\Zulu\zulu-17\bin\javaw.exe  (Mar 27, 2024, 10:07:07 AM) [pid: 352]
```

Authorize based on JWT

Let us recollect the JWT Process

- Client sends username and password to server
- Servers validates credentials, creates token (JWT) and reponds it back
- Client attaches the token in the subsequent requests to server
- Server validates the token (JWT) on each client request

The points highlighted in blue above are already implemented.

Now all the application related requested coming in should send the token received and the server needs to be incorporate this.

So far, whatever we have implemented are service specific and we introduced respective controller methods, but now the requirment is the validate all the other services provided by this application to be validated for JWT, hence we cannot use a controller here. The ideal solution would be to use a filter as it can intercept all the requests received by this application.

Follow steps below to get this incorporated:

Create UserDetailsService and generate UserDetail

AppUserService.java

- Create AppUserService class and implements UserDetailsService interface and use annotation @Service
- Use @Autowired and inject the EmployeeDao instance in this class
- Override loadUserByUsername() method

```
package com.example.demo.service;

import java.util.ArrayList;

import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
importorg.springframework.security.core.userdetails.UserDetailsService;
importorg.springframework.security.core.userdetails.UsernameNotFoundExc
eption;
 import org.springframework.stereotype.Service;

 @Service
 public class AppUserService implements UserDetailsService
 {
```

```
 @Override
 public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
 // TODO Auto-generated method stub
 return new User("arthi","arthi@123",new ArrayList<>());
        }

    }
```

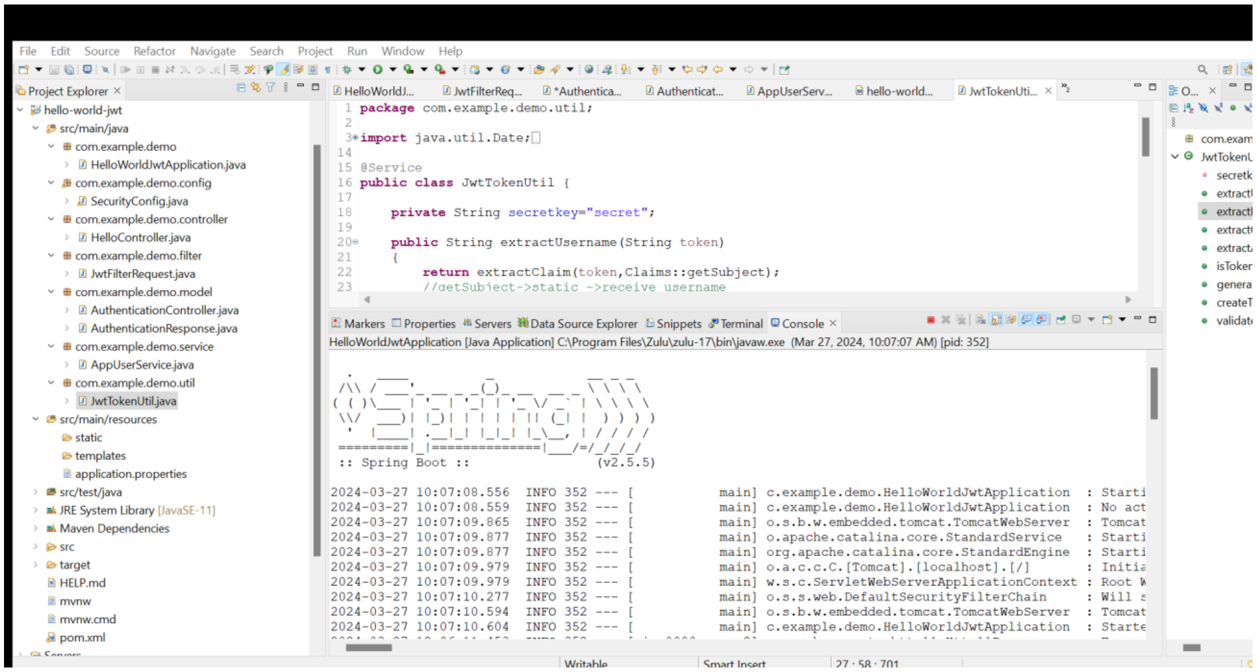
Create a utility class and filter class

JwtTokenUtil.java

- Create a class JwtTokenUtil class, implements Serialization interface and use annotation @Component
- The util class provides various methods to get claims, username, expiration date from basic token and generate JWT token,
- Code given as part of the template

```
package com.example.demo.util;



import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;



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

import org.springframework.stereotype.Service;



import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;



@Service
```

```
}

public class JwtTokenUtil {

    private String secretkey="secret";

    public String extractUsername(String token)
    {
            return extractClaim(token,Claims::getSubject);
            //getSubject->static ->receive username
    }
    public Date extractExpiration(String token)
    {
            return extractClaim(token,Claims::getExpiration);
            //getExpiration->set/extract the time limit for the token to get expired
    }
    public <T> T extractClaim(String token, Function<Claims,T> claimsResolver)
    {
            final Claims claims=extractAllClaims(token);
            return claimsResolver.apply(claims);
    }
    public Claims extractAllClaims(String token)
    {
            return Jwts.parser().setSigningKey(secretkey).parseClaimsJws(token).getBody();
    }
    public Boolean isTokenExpired(String token)
    {
            return extractExpiration(token).before(new Date());

    }

    public String generateToken(UserDetails userdetails)

    {

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

            return createToken(claims,userdetails.getUsername());

    }

    public String createToken(Map<String,Object> claims, String subject)

    {

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

                            .setExpiration(new
Date(System.currentTimeMillis()+1000*60*60*100))

                            .signWith(SignatureAlgorithm.HS256, secretkey).compact();

            //header.payload.signature

    }

    public Boolean validateToken(String token, UserDetails userDetails)

    {

            final String username=extractUsername(token);

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

    }

}
```



```java
package com.example.demo.util;

import java.util.Date;

@Service
public class JwtTokenUtil {

    private String secretkey="secret";

    public String extractUsername(String token)
    {
        return extractClaim(token,Claims::getSubject);
        //getSubject->static ->receive username
```

HelloWorldJwtApplication [Java Application] C:\Program Files\Zulu\zulu-17\bin\javaw.exe (Mar 27, 2024, 10:07:07 AM) [pid: 352]

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.5)

2024-03-27 10:07:08.556  INFO 352 --- [           main] c.example.demo.HelloWorldJwtApplication  : Starti
2024-03-27 10:07:08.559  INFO 352 --- [           main] c.example.demo.HelloWorldJwtApplication  : No act
2024-03-27 10:07:09.865  INFO 352 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat
2024-03-27 10:07:09.877  INFO 352 --- [           main] o.apache.catalina.core.StandardService   : Starti
2024-03-27 10:07:09.877  INFO 352 --- [           main] org.apache.catalina.core.StandardEngine  : Starti
2024-03-27 10:07:09.979  INFO 352 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initia
2024-03-27 10:07:09.979  INFO 352 --- [           main] w.s.c.ServletWebServerApplicationContext : Root W
2024-03-27 10:07:10.277  INFO 352 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will s
2024-03-27 10:07:10.594  INFO 352 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat
2024-03-27 10:07:10.604  INFO 352 --- [           main] c.example.demo.HelloWorldJwtApplication  : Starte
```