

Spring Boot – Spring Security JSON Web Token

JSON Web Tokens are an Open industry standard RFC 7519 method for representing claims securely between two parties (<https://jwt.io>)

Algorithm HS256

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA


```
{  "sub": "1234567890",  "name": "John Doe",  "iat": 1516239022}
```

VERIFY SIGNATURE

```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret)
```

☐ secret base64 encoded

1. Generate Spring Boot Project (<https://start.spring.io>)



Project
☒ Maven Project ☐ Gradle Project

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 2.5.0 (SNAPSHOT) ☐ 2.4.2 (SNAPSHOT) ☒ 2.4.1 ☐ 2.3.8 (SNAPSHOT)
☐ 2.3.7

Project Metadata

Group

com.distareza

Artifact

springsecurityjwt

Name

springsecurityjwt

Description

Demo project for Spring Boot

Package name

com.distareza.springsecurityjwt

Packaging

☒ Jar ☐ War

Java

☐ 15 ☐ 11 ☒ 8

Dependencies ADD DEPENDENCIES... CTRL + B

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

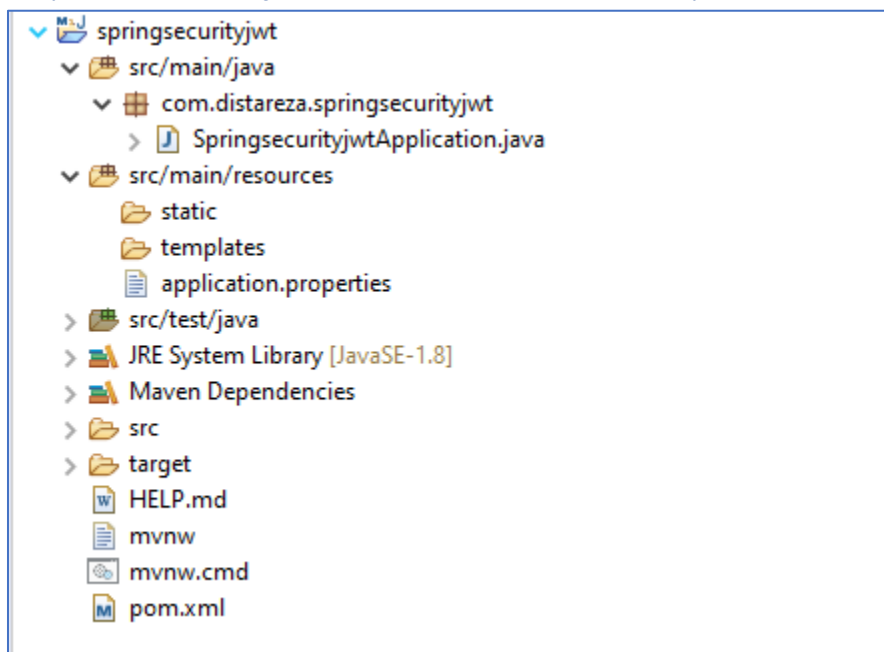
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Security SECURITY
Highly customizable authentication and access-control framework for Spring applications.

H2 Database SOL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Spring Data JPA SOL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

2. Import The Project source code into Eclipse



3. Create User Entity Class

com.distareza.springsecurityjwt.entity.User.java

```
package com.distareza.springsecurityjwt.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "USER_TBL")
public class User {
    @Id
    private int id;
    private String userName;
    private String password;
    private String email;
}
```

Starting with a fresh eclipse installation you, in fact, need to "install" Lombok before being able to use it.

How to install lombok:

Go where you Lombok jar is (e.g. you can find in
~/.m2/repository/org/projectlombok/lombok/1.16.10/lombok-1.16.10.jar),
run it (Example: java -jar lombok-1.16.10.jar). A window should appear, browse to your
eclipse.exe location.

Click on install.

Launch Eclipse, update project configuration on all projects and you ready to go.

4. Create User JPA Repository Interface

com.distareza.springsecurityjwt.repository.UserRepository.java

```
package com.distareza.springsecurityjwt.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.distareza.springsecurityjwt.entity.User;

public interface UserRepository extends JpaRepository<User, Integer> {

    public User findByName(String userName);

}
```

5. Create User Service Class

com.distareza.springsecurityjwt.service.UserService.java

```
package com.distareza.springsecurityjwt.service;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.distareza.springsecurityjwt.entity.User;
import com.distareza.springsecurityjwt.repository.UserRepository;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public void initUser() {

        List<User> users = Stream.of(
            new User( 1, "distareza", "password", "distareza@gmail.com"),
            new User( 2, "admin", "nimda", "admin@localhost.com"),
            new User( 3, "guest", "whoami", "guess@localhost.com")
        ).collect(Collectors.toList());
        userRepository.saveAll(users);

    }

    public List<User> getAllUser() {
        return userRepository.findAll();
    }

    public User findtUser(String userName) {
        return userRepository.findByUserName(userName);
    }

}
```

Create Test Class

com.distareza.springsecurityjwt.TestUserService.java

```
package com.distareza.springsecurityjwt;

import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import com.distareza.springsecurityjwt.service.UserService;
import com.fasterxml.jackson.databind.ObjectMapper;

@SpringBootTest
public class TestUserService {

    @Autowired
    private UserService userService;
```

```

@Test
public void testInitUser() throws Exception {
    userService.initUser();

    ObjectMapper objectMapper = new ObjectMapper();
    String jsonText = objectMapper.writeValueAsString(userService.getAllUser());
    System.out.println(jsonText);

    assertTrue(true);
};

@Test
public void testFindUserByName() throws Exception {
    userService.initUser();
    System.out.println( userService.findtUser("admin") );
    assertTrue(true);
}
}

```

6. Update application.properties or application.yml

application.properties

```

spring.h2.console.enabled = true
server.port = 9192

```

application.yml

```

spring:
  h2:
    console:
      enabled: true

server:
  port: 9192

```

7. Create User Authentication Service Class

com.distareza.springsecurityjwt.service.UserAuthenticationService.java

```
package com.distareza.springsecurityjwt.service;

import java.util.ArrayList;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import com.distareza.springsecurityjwt.entity.User;

@Service
public class UserAuthenticationService implements UserDetailsService {

    @Autowired
    private UserService service;

    @Override
    public UserDetails loadUserByUsername(String username)
        throws UsernameNotFoundException {
        User user = service.findtUser(username);
        return new org.springframework.security.core.userdetails.User(
            user.getUserName(),
            user.getPassword(),
            new ArrayList<>());
    }
}
```

8. Create Security Config Class

com.distareza.springsecurityjwt.config.SecurityConfig.java

```
package com.distareza.springsecurityjwt.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.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

import com.distareza.springsecurityjwt.service.UserAuthenticationService;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserAuthenticationService userAuthenticationService;

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

```

@Bean
public PasswordEncoder passwordEncoder(){
    return NoOpPasswordEncoder.getInstance();
}
}

```

9. Create Default Controller Class

com.distareza.springsecurityjwt.controller.WelcomeController.java

```

package com.distareza.springsecurityjwt.controller;

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

@RestController
public class WelcomeController {

    @GetMapping("/")
    public String welcome() {
        return "Welcome";
    }

}

```

10. Initialize Default User on start up Class

com.distareza.springsecurityjwt.SpringsecurityjwtApplication.java

```

package com.distareza.springsecurityjwt;

import javax.annotation.PostConstruct;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.distareza.springsecurityjwt.service.UserService;

@SpringBootApplication
public class SpringsecurityjwtApplication {

    @Autowired
    private UserService userService;

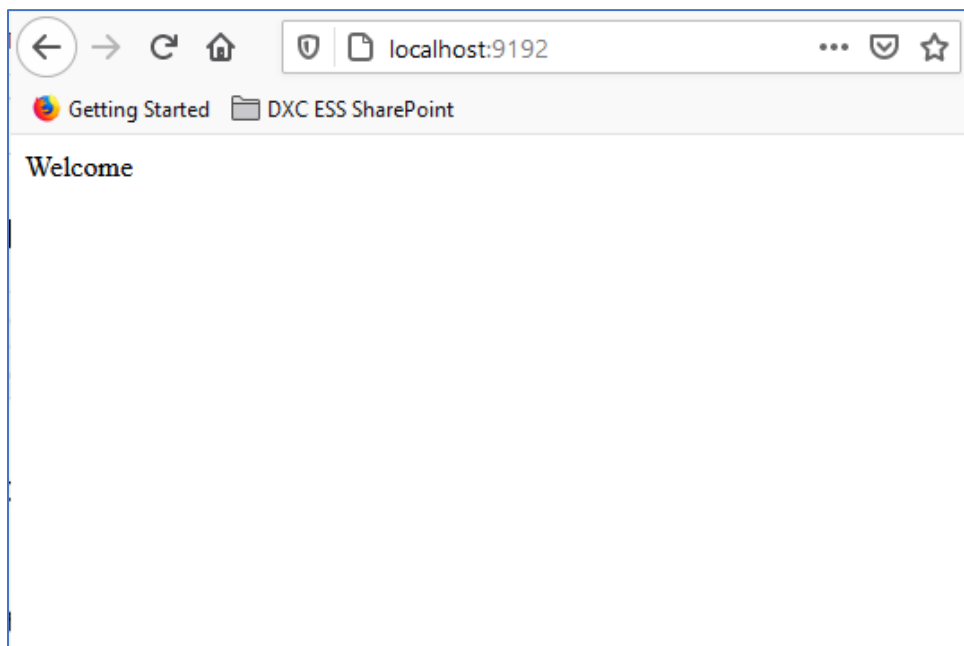
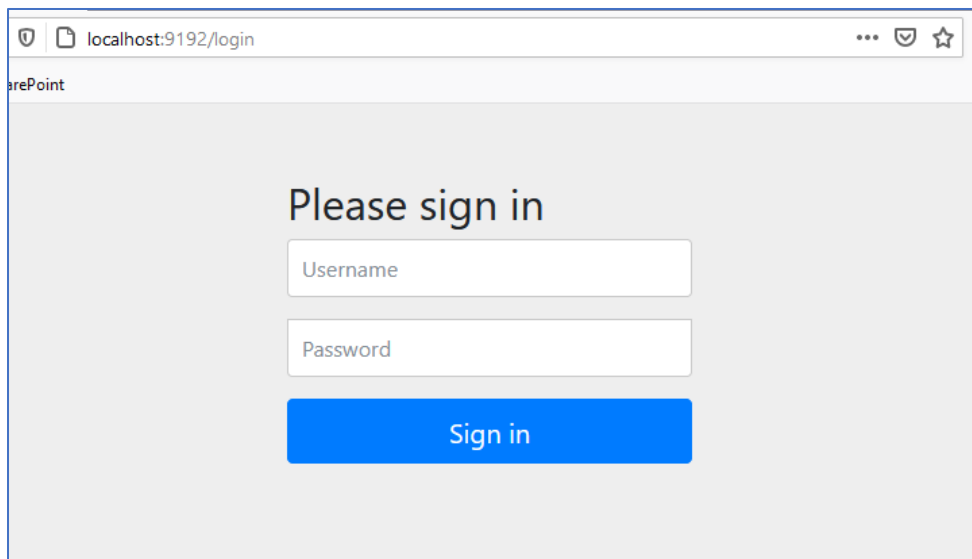
    @PostConstruct
    public void initUser() {
        userService.initUser();
    }

    public static void main(String[] args) {
        SpringApplication.run(SpringsecurityjwtApplication.class, args);
    }

}

```

Try and Test to run spring boot application and open browser <http://localhost:9192>



You have successfully implementing Spring Security on Spring Boot Application

11. Implement JWT Library add Dependency on Pom.xml

Add following dependency on pom.xml

pom.xml

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

12. Create JWT Util Class

com.distareza.springsecurityjwt.util.JWTUtil.java

```
package com.distareza.springsecurityjwt.util;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Service
public class JWTUtil {

    private String secret = "mysecretcode";

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
    }

    private Boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    public String generateToken(String username) {
        Map<String, Object> claims = new HashMap<>();
        return createToken(claims, username);
    }

    private 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 * 10))
        .signWith(SignatureAlgorithm.HS256, secret)
        .compact();
    }

    public Boolean validateToken(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }
}

```

13. Create New Entity Class for Authentication Request

com.distareza.springsecurityjwt.entity.AuthRequest.java

```

package com.distareza.springsecurityjwt.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class AuthRequest {

    private String username;
    private String password;
}

```

14. Create New Controller Class for Authentication

com.distareza.springsecurityjwt.controller.AuthenticateController.java

```

package com.distareza.springsecurityjwt.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.distareza.springsecurityjwt.entity.AuthRequest;
import com.distareza.springsecurityjwt.util.JWTUtil;

@RestController
public class AuthenticateController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JWTUtil jwtUtil;

    @PostMapping("/authenticate")
    public String generateToken(@RequestBody AuthRequest authRequest)
        throws Exception {
    }
}

```

```

        try {
            authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                    authRequest.getUserName(),
                    authRequest.getPassword())
            );
        } catch (Exception ex) {
            throw new Exception("Invalid username/password");
        }

        return jwtUtil.generateToken(authRequest.getUserName());
    }
}

```

Add following method in Security Config Class to enable authenticationManager bean and exclude path “/authenticate” from authentication when user want to generate token.

com.distareza.springsecurityjwt.config.SecurityConfig.java

```

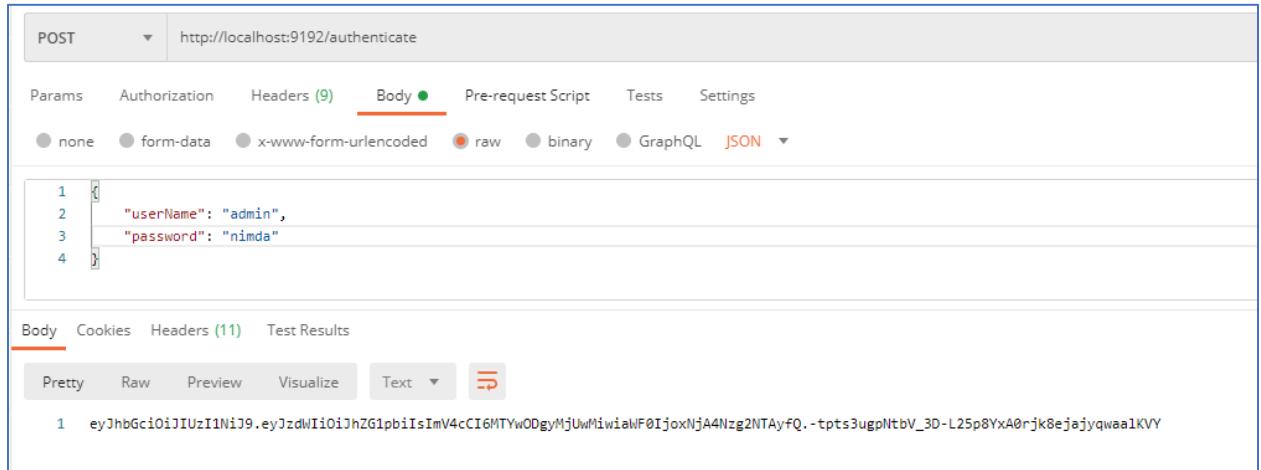
@Bean(name = BeanIds.AUTHENTICATION_MANAGER)
@Override
public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf()
        .disable()
        .authorizeRequests()
        .antMatchers("/authenticate")
        .permitAll()
        .anyRequest()
        .authenticated();
}

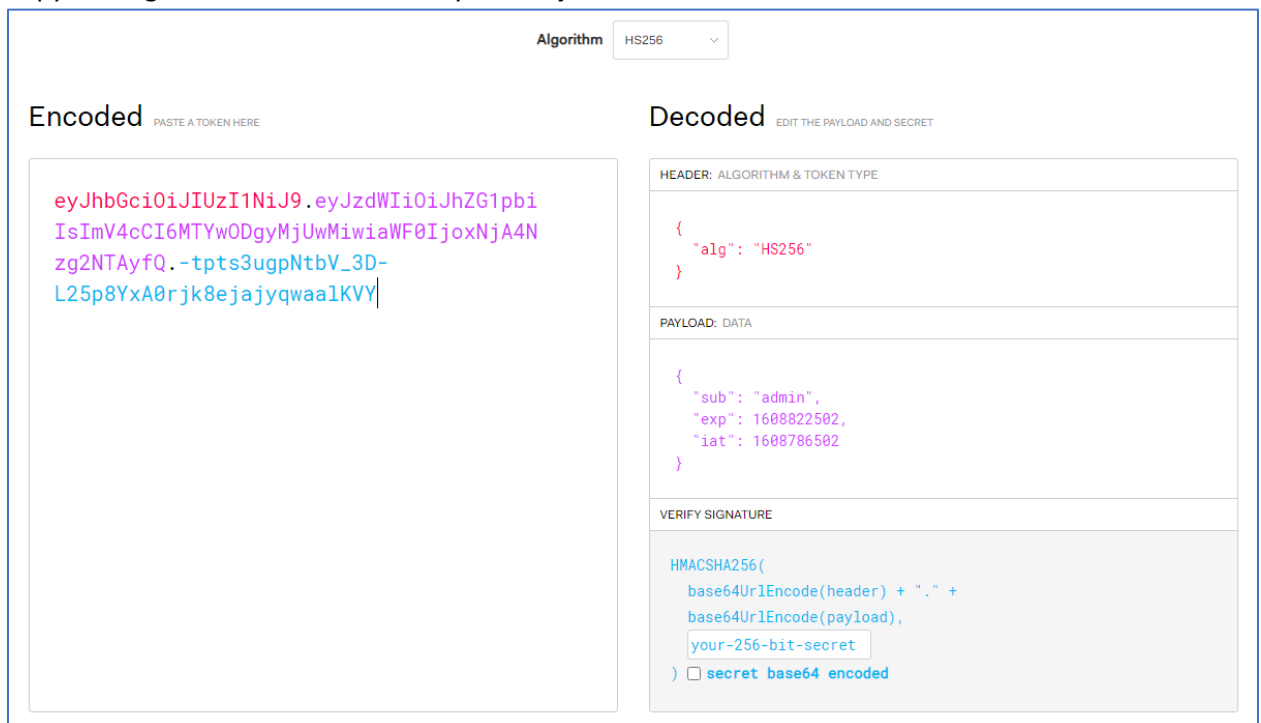
```

15. Generate JSON Web Token

Run spring boot application and generate JWT from postman



Copy those generated Token lets analyze it on jwt.io as follows



16. Create Filter Class to Enable JWT for API Call

com.distareza.springsecurityjwt.filter.JwtFilter.java

```
package com.distareza.springsecurityjwt.filter;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import com.distareza.springsecurityjwt.service.UserAuthenticationService;
import com.distareza.springsecurityjwt.util.JWTUtil;

@Component
public class JwtFilter extends OncePerRequestFilter {

    @Autowired
    private JWTUtil jwtUtil;

    @Autowired
    private UserAuthenticationService userAuthService;

    @Override
    protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain filterChain)
        throws ServletException, IOException {

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

        String token = null;
        String userName = null;

        if (authorizationHeader != null
            && authorizationHeader.startsWith("Bearer ")) {
            token = authorizationHeader.substring(7);
            userName = jwtUtil.extractUsername(token);
        }

        if (userName != null
            && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = userAuthService.loadUserByUsername(userName);

            if (jwtUtil.validateToken(token, userDetails)) {
                UsernamePasswordAuthenticationToken authToken =
                    new UsernamePasswordAuthenticationToken(
                        userDetails,
                        null,
                        userDetails.getAuthorities());
                authToken.setDetails(
                    new WebAuthenticationDetailsSource()
                        .buildDetails(request));

                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }

        filterChain.doFilter(request, response);
    }
}
```

```

        }
    }

    filterChain.doFilter(request, response);
}
}

```

Update Security Config class

com.distareza.springsecurityjwt.config.SecurityConfig.java

```

@Autowired
private JwtFilter jwtFilter;

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable().authorizeRequests().antMatchers("/authenticate")
        .permitAll().anyRequest().authenticated()
        .and().exceptionHandling().and().sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);
}

```

Test Run spring boot application and call API with JWT authentication from postman

The screenshot shows a Postman interface for a GET request to `http://localhost:9192/`. The 'Headers' tab is active, showing an 'Authorization' header with a Bearer token. The 'Body' tab is also active, displaying the response 'Welcome'.

GET `http://localhost:9192/`

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWllOiJhZG1pbilslmV4cCI6MTYwODgyMjU
Key	Value

Body Cookies (1) Headers (11) Test Results

Pretty Raw Preview Visualize Text

```
1 Welcome
```

Test call API without authentication

The screenshot shows a Postman interface for a GET request to `http://localhost:9192/`. The 'Headers' tab is active, showing an empty header list. The 'Body' tab is also active, displaying a JSON response with a 403 status and an error message.

GET `http://localhost:9192/`

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Headers 8 hidden

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies (1) Headers (11) Test Results Status: 403

Pretty Raw Preview Visualize JSON

```
1 {
2   "timestamp": "2020-12-24T05:43:18.208+00:00",
3   "status": 403,
4   "error": "Forbidden",
5   "message": "",
6   "path": "/"
7 }
```