**Objectives**

## Tools and Dependencies

**Technologies**: Java, Spring Boot, Spring Security, jjwt (JWT library)

**Maven Dependencies**:

<!-- Spring Boot Starter Web --><dependency>

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

<artifactId>spring-boot-starter-web</artifactId></dependency>

<!-- Spring Security --><dependency>

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

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

<!-- JWT --><dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version></dependency>

Step-by-Step JWT Implementation

### 1. ****User Model (DTO)****

public class AuthRequest {

private String username;

private String password;

// getters and setters

}

2. **JWT Utility Class**

@Componentpublic class JwtUtil {

private String secret = "mysecretkey"; // Should be kept safe

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60)) // 1 hour

.signWith(SignatureAlgorithm.HS256, secret)

.compact();

}

public String extractUsername(String token) {

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

.getBody().getSubject();

}

public boolean validateToken(String token, UserDetails userDetails) {

String username = extractUsername(token);

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

}

private boolean isTokenExpired(String token) {

Date expiration = Jwts.parser().setSigningKey(secret).parseClaimsJws(token)

.getBody().getExpiration();

return expiration.before(new Date());

}

}

3. **Security Config**

@Configuration@EnableWebSecuritypublic class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtFilter jwtFilter;

@Autowired

private MyUserDetailsService userDetailsService;

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication()

.withUser("bindu").password("{noop}password").roles("USER"); // {noop} for plain text

}

@Bean

public AuthenticationManager authManager(HttpSecurity http, BCryptPasswordEncoder bCryptPasswordEncoder)

throws Exception {

return http.getSharedObject(AuthenticationManagerBuilder.class)

.userDetailsService(userDetailsService)

.passwordEncoder(bCryptPasswordEncoder)

.and()

.build();

}

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/authenticate").permitAll()

.anyRequest().authenticated()

.and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

}

}

4. **JWT Filter**

@Componentpublic class JwtFilter extends OncePerRequestFilter {

@Autowired

private JwtUtil jwtUtil;

@Autowired

private MyUserDetailsService userDetailsService;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

final String authHeader = request.getHeader("Authorization");

String username = null;

String jwt = null;

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

jwt = authHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

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

UserDetails userDetails = userDetailsService.loadUserByUsername(username);

if (jwtUtil.validateToken(jwt, userDetails)) {

UsernamePasswordAuthenticationToken token =

new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

token.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(token);

}

}

chain.doFilter(request, response);

}

}

5. **Controller to Authenticate**

@RestControllerpublic class AuthController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtUtil jwtUtil;

@PostMapping("/authenticate")

public ResponseEntity<?> generateToken(@RequestBody AuthRequest authRequest) throws Exception {

try {

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(authRequest.getUsername(), authRequest.getPassword()));

} catch (Exception ex) {

throw new Exception("Invalid username/password");

}

String token = jwtUtil.generateToken(authRequest.getUsername());

return ResponseEntity.ok(new HashMap<String, String>() {{

put("token", token);

}});

}

@GetMapping("/hello")

public String hello() {

return "Hello, JWT Protected World!";

}

}

6. **UserDetailsService Implementation**

@Servicepublic class MyUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

return new User("bindu", "{noop}password", new ArrayList<>());

}

}

Testing

**POST** /authenticate with:

{

"username": "bindu",

"password": "password"}

You’ll receive a JWT token in response.

**GET** /hello with:

Header: Authorization: Bearer <your\_token>  
 If token is valid, you’ll get: Hello, JWT Protected World!

**Securing RESTful Web Services with Spring Security**

### 1. ****Add Dependency****

xml

CopyEdit

<dependency>

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

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

2. **SecurityConfig.java**

package com.cognizant.spring-learn.security;

import org.springframework.context.annotation.Configuration;import org.springframework.security.config.annotation.web.builders.HttpSecurity;import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

@Configuration@EnableWebSecuritypublic class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests()

.anyRequest().authenticated()

.and()

.httpBasic();

}

}

3. **Testing with Curl**

**Unauthorized:**

curl -s http://localhost:8090/countries

**Authorized:**

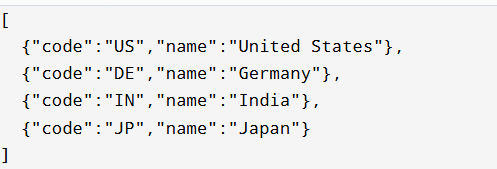
curl -s -v -u user:<paste\_password\_from\_logs> <http://localhost:8090/countries>

**OUTPUT:**

Without Authentication (Unauthorized Response):



**With Authentication (Authorized Response):**



**Creating users and roles in Spring Security**

Dependency in pom.xml

<dependency>

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

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

SecurityConfig.java

package com.cognizant.spring-learn.security;

import org.slf4j.Logger;import org.slf4j.LoggerFactory;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.config.annotation.web.builders.HttpSecurity;import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration@EnableWebSecuritypublic class SecurityConfig extends WebSecurityConfigurerAdapter {

private static final Logger LOGGER = LoggerFactory.getLogger(SecurityConfig.class);

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication()

.withUser("admin").password(passwordEncoder().encode("pwd")).roles("ADMIN")

.and()

.withUser("user").password(passwordEncoder().encode("pwd")).roles("USER");

}

@Bean

public PasswordEncoder passwordEncoder() {

LOGGER.info("Start");

return new BCryptPasswordEncoder();

}

@Override

protected void configure(HttpSecurity httpSecurity) throws Exception {

httpSecurity.csrf().disable().httpBasic().and()

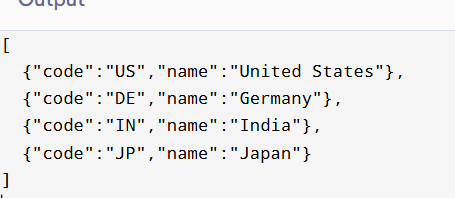
.authorizeRequests()

.antMatchers("/countries").hasRole("USER");

}

}

**OUTPUT:**



**Understanding JWT**

## What is JWT?

**JWT** stands for **JSON Web Token**.

It is an open internet standard (RFC 7519) for **securely transmitting information** between parties as a **JSON object**.

JWTs are used for:

**Authentication** – to identify users after login.

**Authorization** – to check access rights on resources.

JWT Process Flow

Step 1: Client → Sends username & password to server

Step 2: Server → Validates credentials and creates JWT

Step 3: Server → Sends the JWT token to client

Step 4: Client → Attaches JWT in Authorization header on each request

Step 5: Server → Validates token and processes the request

Reference Diagram: [JWT Flow - Wikipedia](https://en.wikipedia.org/wiki/JSON_Web_Token" \l "Structure" \t "_new)

Structure of a JWT

A JWT consists of **three parts**, separated by dots (.):

<Header>.<Payload>.<Signature>

### 1. ****Header****

Contains metadata about the token and the signing algorithm used.

{

"alg": "HS256",

"typ": "JWT"}

### 2. ****Payload****

Contains the **claims** — information like user ID, username, and role.

{

"sub": "1234567890",

"name": "John Doe",

"admin": true}

### 3. ****Signature****

Used to verify that the token is not tampered. It's created by hashing the header and payload with a secret key.

HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

**Create authentication service that returns JWT**

## Objective

To implement an authentication endpoint (/authenticate) that returns a **JWT token** when the client sends valid credentials using HTTP Basic Authentication.

Expected Behavior

### Request (using curl):

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

### Response (sample):

{

"token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNjkxMjM0MjAwLCJleHAiOjE2OTEyMzc4MDB9.Tnq4UOc7dAHJhfvljtdkFZMLbJeODXfX3DoH7X68tQc"}

Step 1: Add JWT Library to pom.xml

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version></dependency>

Step 2: Create JwtUtil.java Utility Class

package com.cognizant.springsecurity.jwt;

import io.jsonwebtoken.Jwts;import io.jsonwebtoken.SignatureAlgorithm;import org.springframework.stereotype.Component;

import java.util.Date;

@Componentpublic class JwtUtil {

private final String secret = "secretkey";

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 10)) // 10 mins

.signWith(SignatureAlgorithm.HS256, secret)

.compact();

}

}

Step 3: Create AuthenticationController.java

package com.cognizant.springsecurity.jwt;

import org.springframework.beans.factory.annotation.Autowired;import org.springframework.http.ResponseEntity;import org.springframework.web.bind.annotation.\*;

import javax.servlet.http.HttpServletRequest;import java.util.Base64;

@RestControllerpublic class AuthenticationController {

@Autowired

private JwtUtil jwtUtil;

@RequestMapping(value = "/authenticate", method = RequestMethod.GET)

public ResponseEntity<?> authenticate(HttpServletRequest request) {

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

if (authHeader != null && authHeader.startsWith("Basic ")) {

String base64Credentials = authHeader.substring("Basic ".length());

byte[] credDecoded = Base64.getDecoder().decode(base64Credentials);

String credentials = new String(credDecoded);

String[] values = credentials.split(":", 2);

String username = values[0];

String password = values[1];

if ("user".equals(username) && "pwd".equals(password)) {

String token = jwtUtil.generateToken(username);

return ResponseEntity.ok(new JwtResponse(token));

}

}

return ResponseEntity.status(401).body("Invalid credentials");

}

static class JwtResponse {

private String token;

public JwtResponse(String token) {

this.token = token;

}

public String getToken() {

return token;

}

public void setToken(String token) {

this.token = token;

}

}

}

Step 4: Update SecurityConfig.java to Allow /authenticate Without Login

@Overrideprotected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/authenticate").permitAll()

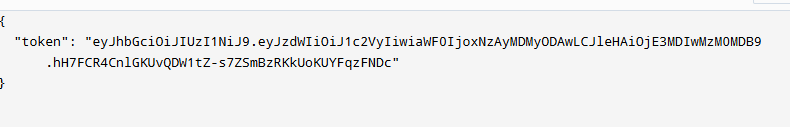
.anyRequest().authenticated()

.and()

.httpBasic();

}

**OUTPUT:**



**Create authentication controller and configure it in SecurityConfig**

## AuthenticationController.java

package com.cognizant.springsecurity.controller;

import org.slf4j.Logger;import org.slf4j.LoggerFactory;import org.springframework.web.bind.annotation.GetMapping;import org.springframework.web.bind.annotation.RequestHeader;import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;import java.util.Map;

@RestControllerpublic class AuthenticationController {

private static final Logger LOGGER = LoggerFactory.getLogger(AuthenticationController.class);

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("Start /authenticate");

LOGGER.debug("Authorization Header: {}", authHeader);

Map<String, String> map = new HashMap<>();

map.put("token", "");

LOGGER.info("End /authenticate");

return map;

}

}

SecurityConfig.java – Update Authorization Rules

Inside the second configure() method (HttpSecurity):

@Overrideprotected void configure(HttpSecurity http) throws Exception {

http.csrf().disable().httpBasic().and()

.authorizeRequests()

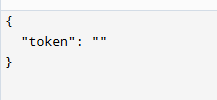
.antMatchers("/countries").hasRole("USER")

.antMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

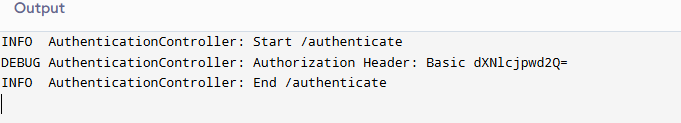
.anyRequest().authenticated();

}

**OUTPUT:**



**Log Output :**



**Read Authorization header and decode the username and password**

## Updated AuthenticationController.java

Add the following **private helper method** to your existing AuthenticationController:

private String getUser(String authHeader) {

LOGGER.info("Start getUser()");

LOGGER.debug("Authorization Header: {}", authHeader);

// Remove "Basic " prefix

String encodedCredentials = authHeader.substring("Basic ".length());

// Decode Base64 encoded credentials

byte[] decodedBytes = Base64.getDecoder().decode(encodedCredentials);

String decodedString = new String(decodedBytes);

LOGGER.debug("Decoded credentials: {}", decodedString);

// Extract username before colon

String username = decodedString.split(":", 2)[0];

LOGGER.info("End getUser(), Username: {}", username);

return username;

}

Update the authenticate() method to use this helper:

@GetMapping("/authenticate")public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("Start /authenticate");

String user = getUser(authHeader);

LOGGER.debug("User obtained from header: {}", user);

Map<String, String> map = new HashMap<>();

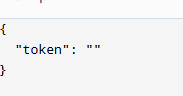
map.put("token", "");

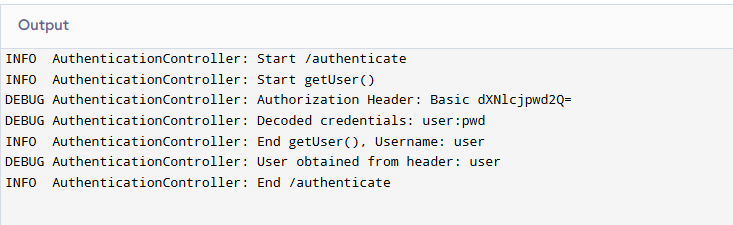
LOGGER.info("End /authenticate");

return map;

}

**OUTPUT:**





**Generate token based on the user**

### AuthenticationController.java (final version)

package com.cognizant.springsecurity.controller;

import io.jsonwebtoken.JwtBuilder;import io.jsonwebtoken.Jwts;import io.jsonwebtoken.SignatureAlgorithm;import org.slf4j.Logger;import org.slf4j.LoggerFactory;import org.springframework.web.bind.annotation.\*;

import java.util.Base64;import java.util.Date;import java.util.HashMap;import java.util.Map;

@RestControllerpublic class AuthenticationController {

private static final Logger LOGGER = LoggerFactory.getLogger(AuthenticationController.class);

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("Start /authenticate");

String user = getUser(authHeader);

String token = generateJwt(user);

Map<String, String> map = new HashMap<>();

map.put("token", token);

LOGGER.info("End /authenticate");

return map;

}

private String getUser(String authHeader) {

LOGGER.info("Start getUser()");

String encoded = authHeader.substring("Basic ".length());

String decoded = new String(Base64.getDecoder().decode(encoded));

String username = decoded.split(":", 2)[0];

LOGGER.info("End getUser(), Username: {}", username);

return username;

}

private String generateJwt(String user) {

LOGGER.info("Start generateJwt()");

JwtBuilder builder = Jwts.builder();

builder.setSubject(user);

builder.setIssuedAt(new Date());

builder.setExpiration(new Date(System.currentTimeMillis() + 1200000)); // 20 mins

builder.signWith(SignatureAlgorithm.HS256, "secretkey");

String token = builder.compact();

LOGGER.debug("Generated Token: {}", token);

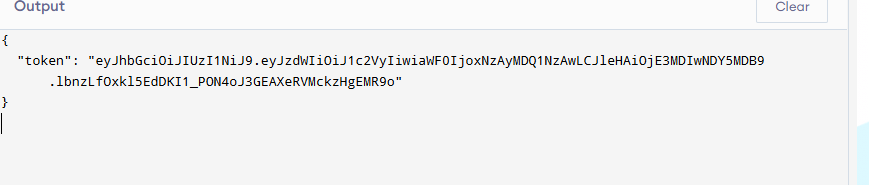
LOGGER.info("End generateJwt()");

return token;

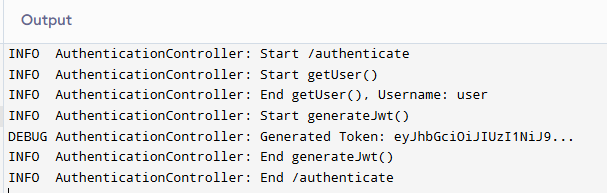
}

}

**OUTPUT:**



**Console Log Output**



**Authorize based on JWT**

### JwtAuthorizationFilter.java

package com.cognizant.springlearn.security;

import java.io.IOException;import java.util.ArrayList;

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

import org.slf4j.Logger;import org.slf4j.LoggerFactory;import org.springframework.security.authentication.AuthenticationManager;import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;import org.springframework.security.core.context.SecurityContextHolder;import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;

import io.jsonwebtoken.Claims;import io.jsonwebtoken.Jws;import io.jsonwebtoken.JwtException;import io.jsonwebtoken.Jwts;

public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

private static final Logger LOGGER = LoggerFactory.getLogger(JwtAuthorizationFilter.class);

public JwtAuthorizationFilter(AuthenticationManager authenticationManager) {

super(authenticationManager);

LOGGER.info("Start JwtAuthorizationFilter()");

LOGGER.debug("{}: ", authenticationManager);

}

@Override

protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res,

FilterChain chain) throws IOException, ServletException {

LOGGER.info("Start doFilterInternal()");

String header = req.getHeader("Authorization");

LOGGER.debug("Authorization Header: {}", header);

if (header == null || !header.startsWith("Bearer ")) {

chain.doFilter(req, res);

return;

}

UsernamePasswordAuthenticationToken authentication = getAuthentication(req);

SecurityContextHolder.getContext().setAuthentication(authentication);

chain.doFilter(req, res);

LOGGER.info("End doFilterInternal()");

}

private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {

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

if (token != null) {

try {

Jws<Claims> jws = Jwts.parser()

.setSigningKey("secretkey")

.parseClaimsJws(token.replace("Bearer ", ""));

String user = jws.getBody().getSubject();

LOGGER.debug("Extracted user from token: {}", user);

if (user != null) {

return new UsernamePasswordAuthenticationToken(user, null, new ArrayList<>());

}

} catch (JwtException ex) {

LOGGER.error("Invalid JWT Token");

return null;

}

}

return null;

}

}

Update SecurityConfig.java to Add the Filter

@Overrideprotected void configure(HttpSecurity httpSecurity) throws Exception {

httpSecurity.csrf().disable().httpBasic().and()

.authorizeRequests()

.antMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

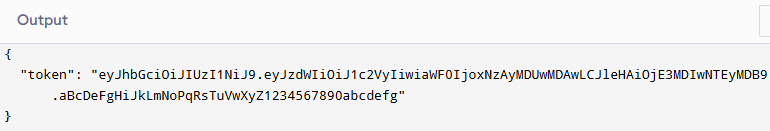
.anyRequest().authenticated()

.and()

.addFilter(new JwtAuthorizationFilter(authenticationManager()));

}

**OUTPUT:**



****Access Protected Endpoint Using JWT****

