* [Home](https://www.javainuse.com/home)
* [Java](https://www.javainuse.com/)
* [Spring](https://www.javainuse.com/)
* [Angular](https://www.javainuse.com/)
* [Full Stack](https://www.javainuse.com/)
* [Apache Camel](https://www.javainuse.com/)
* [Cloud Frameworks](https://www.javainuse.com/)
* [Messaging](https://www.javainuse.com/)
* [Drools](https://www.javainuse.com/)
* [Search Engine](https://www.javainuse.com/)
* [Hazelcast](https://www.javainuse.com/hazelcast)
* [DevOps](https://www.javainuse.com/)

Search Tutorials

Top of Form

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Bottom of Form

* [Home](https://www.javainuse.com/home)
* [Java](https://www.javainuse.com/)
* [Spring](https://www.javainuse.com/)
* [Angular](https://www.javainuse.com/)
* [Full Stack](https://www.javainuse.com/)
* [Apache Camel](https://www.javainuse.com/)
* [Cloud Frameworks](https://www.javainuse.com/)
* [Messaging](https://www.javainuse.com/)
* [Drools](https://www.javainuse.com/)
* [Search Engine](https://www.javainuse.com/)
* [Hazelcast](https://www.javainuse.com/hazelcast)
* [DevOps](https://www.javainuse.com/)

#### Search Tutorials

Top of Form

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Bottom of Form

# What is JWT?

[**Close X**](https://www.javainuse.com/)

In this tutorial we will be learning the basic of JSON Web Token (JWT). In next tutorial we will be implementing [Spring Boot +JWT + MYSQL Hello World Example](https://www.javainuse.com/spring/boot-jwt)  
JWT stands for JSON Web Token. JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. The client will need to authenticate with the server using the credentials only once. During this time the server validates the credentials and returns the client a JSON Web Token(JWT). For all future requests the client can authenticate itself to the server using this JSON Web Token(JWT) and so does not need to send the credentials like username and password.

Spring Boot JSON Web Token- Table of Contents

[**What is JWT(JSON Web Token)**](https://www.javainuse.com/spring/jwt)

[Spring Boot +JSON Web Token(JWT) Hello World Example](https://www.javainuse.com/spring/boot-jwt)

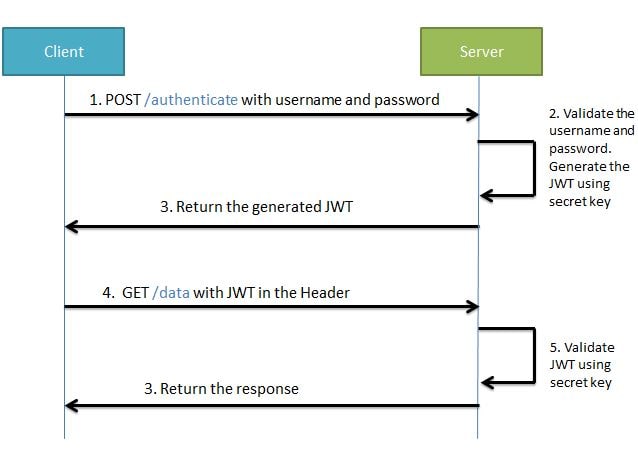
[Spring Boot +JSON Web Token(JWT) + MYSQL Example](https://www.javainuse.com/spring/boot-jwt-mysql)

[Angular 7 + Spring Boot JWT Authentication Hello World Example](https://www.javainuse.com/spring/ang7-jwt)

Video

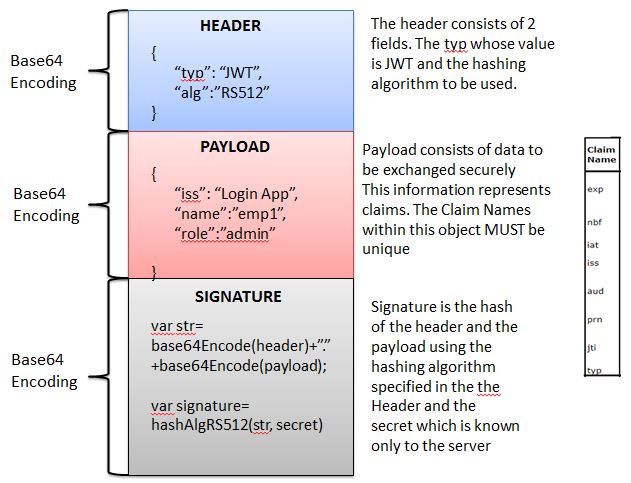
This tutorial is explained in the below Youtube Video.

## Workflow of how JWT is used



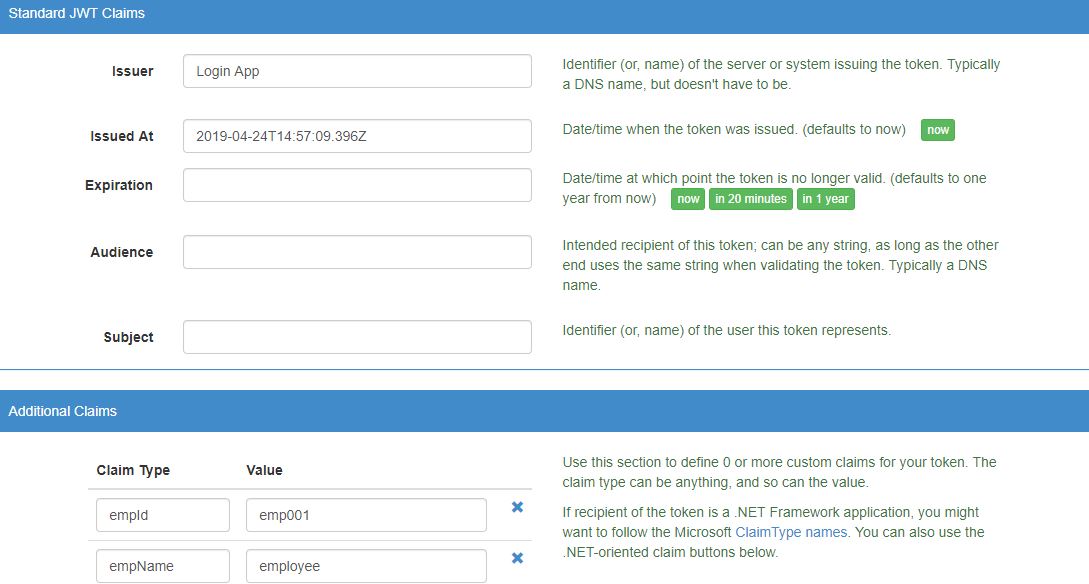
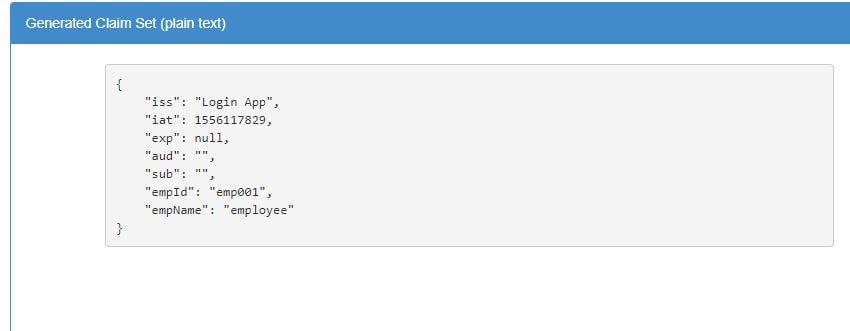
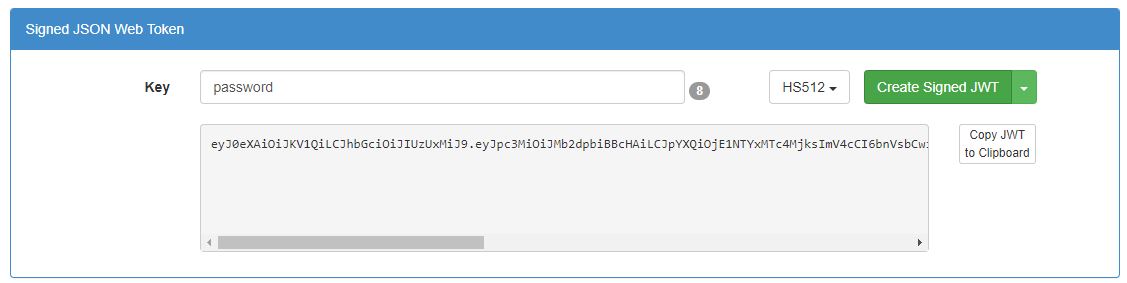
During the first request the client sends a POST request with username and password. Upon successful authentication the server generates the JWT sends this JWT to the client. This JWT can contain a payload of data. On all subsequent requests the client sends this JWT token in the header. Using this token the server authenticates the user. So we don't need the client to send the user name and password to the server during each request for authentication, but only once after which the server issues a JWT to the client. A JWT payload can contain things like user ID so that when the client again sends the JWT, you can be sure that it is issued by you, and you can see to whom it was issued.

## Structure of JWT

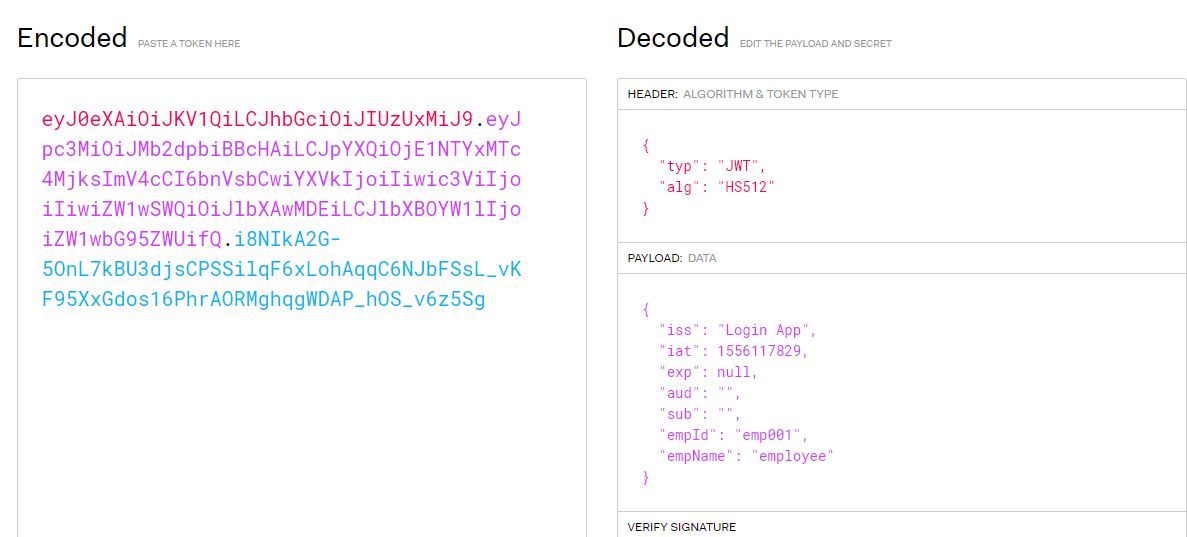
JWT has the following format -**header.payload.signature**  
  
**Structure of JWT-**  


An important point to remember about JWT is that the information in the payload of the JWT is visible to everyone. So we should not pass any sensitive information like passwords in the payload. We can encrypt the payload data if we want to make it more secure. However we can be sure that no one can tamper and change the payload information. If this is done the server will recognize it.

## Creating a JWT Token

We will be creating a JWT token using [JWT Online Token Generator](http://jwtbuilder.jamiekurtz.com/)  
Specify the payload data as folows-  
  
We will be having following claims in the payload-  
  
Sign the payload using the hashing algorithm-  


## Inspect the contents of the created token

We will be inspecting JWT token using [JWT Online Decoder](https://jwt.io/)  
  


## Popular Posts

* [E-commerce Website - Online Book Store using Angular 8 + Spring Boot](https://www.javainuse.com/fullstack/ecommerce)
* [Spring Boot +JSON Web Token(JWT) Hello World Example](https://www.javainuse.com/spring/boot-jwt)
* [Angular 7 + Spring Boot Application Hello World Example](https://www.javainuse.com/spring/ang7-hello)
* [Build a Real Time Chat Application using Spring Boot + WebSocket + RabbitMQ](https://www.javainuse.com/spring/boot-websocket-chat)
* [Pivotal Cloud Foundry Tutorial - Deploy Spring Boot Application Hello World Example](https://www.javainuse.com/pcf/pcf-hello)
* [Deploying Spring Based WAR Application to Docker](https://www.javainuse.com/devOps/docker/docker-war)
* [EIP patterns using Apache Camel](https://www.javainuse.com/camel/camel_EIP)
* [Spring Cloud- Netflix Eureka + Ribbon Simple Example](https://www.javainuse.com/spring/spring_ribbon)
* [Spring Cloud- Netflix Hystrix Circuit Breaker Simple Example](https://www.javainuse.com/spring/spring_hystrix_circuitbreaker)
* [Spring Boot + Swagger Example Hello World Example](https://www.javainuse.com/spring/boot_swagger)
* [Spring Boot Batch Simple example](https://www.javainuse.com/spring/bootbatch)
* [Spring Boot + Apache Kafka Example](https://www.javainuse.com/spring/spring-boot-apache-kafka-hello-world)
* [Spring Boot Admin Simple Example](https://www.javainuse.com/spring/boot-admin)
* [Spring Boot Security - Introduction to OAuth](https://www.javainuse.com/spring/spring-boot-oauth-introduction)
* [Spring Boot OAuth2 Part 1 - Getting The Authorization Code](https://www.javainuse.com/spring/spring-boot-oauth-authorization-code)
* [Spring Boot OAuth2 Part 2 - Getting The Access Token And Using it to Fetch Data.](https://www.javainuse.com/spring/spring-boot-oauth-access-token)
* [JBoss Drools Hello World-Stateful Knowledge Session using KieSession](https://www.javainuse.com/drools_hello_kie)
* [Understand Drools Stateful vs Stateless Knowledge Session](https://www.javainuse.com/drools_states)
* [JBoss Drools- Understanding Drools Decision Table using Simple Example](https://www.javainuse.com/drools/drools_decision)

## See Also

* [Spring Boot Interview Questions](https://www.javainuse.com/spring/SpringBootInterviewQuestions)
* [Spring Batch Interview Questions](https://www.javainuse.com/spring/sprbatch_interview)
* [Spring AOP Interview Questions](https://www.javainuse.com/spring/spring-AOP-interview-quesions)
* [Angular 2 Interview Questions](https://www.javainuse.com/angular/ang2_intvw)
* [Apache Camel Interview Questions](https://www.javainuse.com/camel/Apache_Camel_Questions)
* [JBoss Fuse Interview Questions](https://www.javainuse.com/camel/JBoss_Fuse_Questions)
* [Drools Interview Questions](https://www.javainuse.com/drools/drools_intvw)
* [Java 8 Interview Questions](https://www.javainuse.com/java/java8_intvw)
* [Spring Cloud Interview Questions](https://www.javainuse.com/spring/spring-cloud-interview-questions)
* [Microservices Interview Questions](https://www.javainuse.com/spring/microservices-interview-quesions)
* [Java HashMap and ConcurrentHashMap Interview Questions](https://www.javainuse.com/java/java_map_intvw)
* [Mule ESB frequently asked interview questions](https://www.javainuse.com/misc/muleintvw)
* [Apache Kafka Interview Questions](https://www.javainuse.com/misc/apache-kafka-interview-questions)
* [Tosca Testing Tool Interview Questions](https://www.javainuse.com/misc/tosca-testing-tool-interview-questions)
* [Top Maven Build Tool Interview Questions](https://www.javainuse.com/misc/maven-interview-questions)
* [Top Gradle Build Tool Interview Questions](https://www.javainuse.com/misc/gradle-interview-questions)
* [Miscellaneous Topics](https://www.javainuse.com/misc)

© Copyright JavaInUse. All Rights Reserved.  
[Privacy Policy](https://www.javainuse.com/privacy)

---------------------------------------------------

Spring Boot Security + JWT Hello World Example

In this tutorial we will be developing a Spring Boot Application that makes use of JWT authentication for securing an exposed REST API. In this example we will be making use of hard coded user values for User Authentication. In next tutorial we will be implementing [Spring Boot + JWT + MYSQL JPA for storing and fetching user credentials.](https://www.javainuse.com/spring/boot-jwt-mysql)Any user will be able to consume this API only if it has a valid JSON Web Token(JWT). In a [previous tutorial we have seen what is JWT, when and how to use it.](https://www.javainuse.com/spring/jwt)

Spring Boot JSON Web Token- Table of Contents

[What is JWT(JSON Web Token)](https://www.javainuse.com/spring/jwt)

[**Spring Boot +JSON Web Token(JWT) Hello World Example**](https://www.javainuse.com/spring/boot-jwt)

[Spring Boot +JSON Web Token(JWT) + MYSQL Example](https://www.javainuse.com/spring/boot-jwt-mysql)

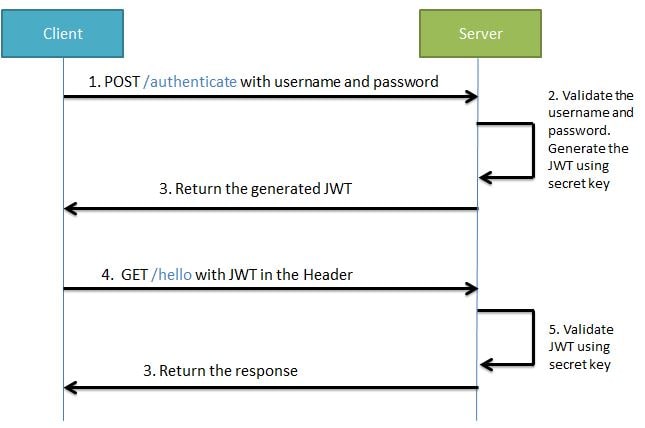
[Angular 7 + Spring Boot JWT Authentication Hello World Example](https://www.javainuse.com/spring/ang7-jwt)

Video

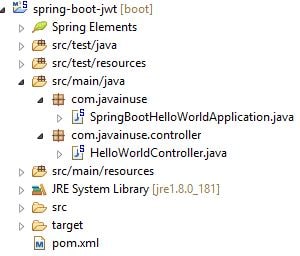
This tutorial is explained in the below Youtube Video.

Lets Begin?

For better understanding we will be developing the project in stages

* Develop a Spring Boot Application to expose a Simple REST GET API with mapping /hello.
* Configure Spring Security for JWT. Expose REST POST API with mapping /authenticate using which User will get a valid JSON Web Token. And then allow the user access to the api /hello only if it has a valid token  
  

Develop a Spring Boot Application to expose a GET REST API

Maven Project will be as follows-  
  


The pom.xml is as follows-

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>spring-boot-jwt</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

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

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

<version>2.1.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

</project>

Create a Controller class for exposing a GET REST API-

package com.javainuse.controller;

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

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

@RestController

public class HelloWorldController {

@RequestMapping({ "/hello" })

public String firstPage() {

return "Hello World";

}

}

Create the bootstrap class with SpringBoot Annotation

Video Player is loading.



package com.javainuse;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

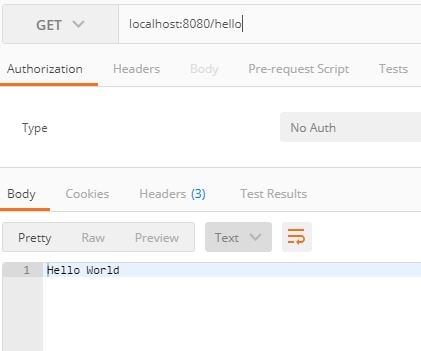
public class SpringBootHelloWorldApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootHelloWorldApplication.class, args);

}

}

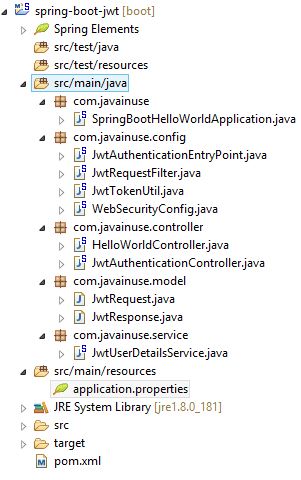
Compile and the run the SpringBootHelloWorldApplication.java as a Java application.  
Go to **localhost:8080/hello**  


Spring Security and JWT Configuration

We will be configuring Spring Security and JWT for performing 2 operations-

* **Generating JWT** - Expose a POST API with mapping **/authenticate**. On passing correct username and password it will generate a JSON Web Token(JWT)
* **Validating JWT** - If user tries to access GET API with mapping **/hello**. It will allow access only if request has a valid JSON Web Token(JWT)

Maven Project will be as follows-

  
The sequence flow for these operations will be as follows-

Generating JWT


Validating JWT

  
Add the Spring Security and JWT dependencies

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>spring-boot-jwt</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

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

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

<version>2.1.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

**<dependency>**

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

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

**</dependency>**

**<dependency>**

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

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

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

**</dependency>**

</dependencies>

</project>

* Define the application.properties. As see in [previous JWT tutorial, we specify the secret key using which we will be using for hashing algorithm.](https://www.javainuse.com/spring/jwt) The secret key is combined with the header and the payload to create a unique hash. We are only able to verify this hash if you have the secret key.
* jwt.secret=javainuse
* JwtTokenUtil

The JwtTokenUtil is responsible for performing JWT operations like creation and validation.It makes use of the io.jsonwebtoken.Jwts for achieving this.

package com.javainuse.config;

import java.io.Serializable;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

import org.springframework.beans.factory.annotation.Value;

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

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Component

public class JwtTokenUtil implements Serializable {

private static final long serialVersionUID = -2550185165626007488L;

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

@Value("${jwt.secret}")

private String secret;

**//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));

}

}

* JWTUserDetailsService

JWTUserDetailsService implements the Spring Security UserDetailsService interface. It overrides the loadUserByUsername for fetching user details from the database using the username. The Spring Security Authentication Manager calls this method for getting the user details from the database when authenticating the user details provided by the user. Here we are getting the **user details from a hardcoded User List**. In the [next tutorial we will be adding the DAO implementation for fetching User Details from the Database.](https://www.javainuse.com/spring/boot-jwt-mysql) Also the password for a user is stored in encrypted format using BCrypt. Previously we have seen [Spring Boot Security - Password Encoding Using Bcrypt.](https://www.javainuse.com/spring/boot_security_jdbc_authentication_bcrypt) Here using the [Online Bcrypt Generator you can generate the Bcrypt for a password.](https://www.javainuse.com/onlineBcrypt)

package com.javainuse.service;

import java.util.ArrayList;

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

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;

@Service

public class JwtUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

**if ("javainuse".equals(username)) {**

**return new User("javainuse", "$2a$10$slYQmyNdGzTn7ZLBXBChFOC9f6kFjAqPhccnP6DxlWXx2lPk1C3G6",**

**new ArrayList<>());**

**}** else {

throw new UsernameNotFoundException("User not found with username: " + username);

}

}

}

* JwtAuthenticationController

Expose a POST API /authenticate using the JwtAuthenticationController. The POST API gets username and password in the body- Using Spring Authentication Manager we authenticate the username and password.If the credentials are valid, a JWT token is created using the JWTTokenUtil and provided to the client.

package com.javainuse.controller;

import java.util.Objects;

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

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.authentication.DisabledException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

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

import org.springframework.web.bind.annotation.CrossOrigin;

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

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

import org.springframework.web.bind.annotation.RequestMethod;

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

import com.javainuse.service.JwtUserDetailsService;

import com.javainuse.config.JwtTokenUtil;

import com.javainuse.model.JwtRequest;

import com.javainuse.model.JwtResponse;

@RestController

@CrossOrigin

public class JwtAuthenticationController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtTokenUtil jwtTokenUtil;

@Autowired

private JwtUserDetailsService userDetailsService;

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

public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest authenticationRequest) throws Exception {

authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());

final UserDetails userDetails = userDetailsService

.loadUserByUsername(authenticationRequest.getUsername());

final String token = jwtTokenUtil.generateToken(userDetails);

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

}

private void authenticate(String username, String password) throws Exception {

try {

**authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));**

} catch (DisabledException e) {

throw new Exception("USER\_DISABLED", e);

} catch (BadCredentialsException e) {

throw new Exception("INVALID\_CREDENTIALS", e);

}

}

}

* JwtRequest

This class is required for storing the username and password we recieve from the client.

package com.javainuse.model;

import java.io.Serializable;

public class JwtRequest implements Serializable {

private static final long serialVersionUID = 5926468583005150707L;

private String username;

private String password;

//need default constructor for JSON Parsing

public JwtRequest()

{

}

public JwtRequest(String username, String password) {

this.setUsername(username);

this.setPassword(password);

}

public String getUsername() {

return this.username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return this.password;

}

public void setPassword(String password) {

this.password = password;

}

}

* JwtResponse

This is class is required for creating a response containing the JWT to be returned to the user.

package com.javainuse.model;

import java.io.Serializable;

public class JwtResponse implements Serializable {

private static final long serialVersionUID = -8091879091924046844L;

private final String jwttoken;

public JwtResponse(String jwttoken) {

this.jwttoken = jwttoken;

}

public String getToken() {

return this.jwttoken;

}

}

* JwtRequestFilter

The JwtRequestFilter extends the Spring Web Filter OncePerRequestFilter class. For any incoming request this Filter class gets executed. It checks if the request has a valid JWT token. If it has a valid JWT Token then it sets the Authentication in the context, to specify that the current user is authenticated.

package com.javainuse.config;

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.javainuse.service.JwtUserDetailsService;

import io.jsonwebtoken.ExpiredJwtException;

@Component

public class JwtRequestFilter extends OncePerRequestFilter {

@Autowired

private JwtUserDetailsService jwtUserDetailsService;

@Autowired

private JwtTokenUtil jwtTokenUtil;

@Override

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

throws ServletException, IOException {

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

String username = null;

String jwtToken = null;

**// JWT Token is in the form "Bearer token". Remove Bearer word and get**

**// only the Token**

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

jwtToken = requestTokenHeader.substring(7);

try {

username = jwtTokenUtil.getUsernameFromToken(jwtToken);

} catch (IllegalArgumentException e) {

System.out.println("Unable to get JWT Token");

} catch (ExpiredJwtException e) {

System.out.println("JWT Token has expired");

}

} else {

logger.warn("JWT Token does not begin with Bearer String");

}

**// Once we get the token validate it.**

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

UserDetails userDetails = this.jwtUserDetailsService.loadUserByUsername(username);

**// if token is valid configure Spring Security to manually set**

**// authentication**

if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

usernamePasswordAuthenticationToken

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

**// After setting the Authentication in the context, we specify**

**// that the current user is authenticated. So it passes the**

**// Spring Security Configurations successfully.**

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

}

}

chain.doFilter(request, response);

}

}

* JwtAuthenticationEntryPoint

This class will extend Spring's AuthenticationEntryPoint class and override its method commence. It rejects every unauthenticated request and send error code 401

package com.javainuse.config;

import java.io.IOException;

import java.io.Serializable;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint, Serializable {

private static final long serialVersionUID = -7858869558953243875L;

@Override

public void commence(HttpServletRequest request, HttpServletResponse response,

AuthenticationException authException) throws IOException {

response.sendError(HttpServletResponse.SC\_UNAUTHORIZED, "Unauthorized");

}

}

* WebSecurityConfig

This class extends the WebSecurityConfigurerAdapter is a convenience class that allows customization to both WebSecurity and HttpSecurity.

package com.javainuse.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

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

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

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

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

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

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

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

@Autowired

private UserDetailsService jwtUserDetailsService;

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

**// configure AuthenticationManager so that it knows from where to load**

**// user for matching credentials**

**// Use BCryptPasswordEncoder**

auth.userDetailsService(jwtUserDetailsService).passwordEncoder(passwordEncoder());

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Override

protected void configure(HttpSecurity httpSecurity) throws Exception {

**// We don't need CSRF for this example**

httpSecurity.csrf().disable()

**// dont authenticate this particular request**

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

**// all other requests need to be authenticated**

anyRequest().authenticated().and().

**// make sure we use stateless session; session won't be used to**

**// store user's state.**

exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS);

**// Add a filter to validate the tokens with every request**

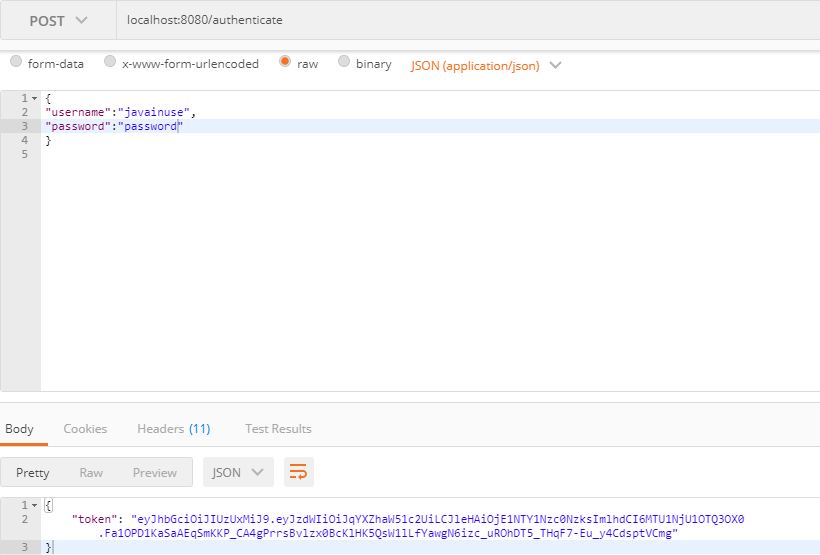
httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

}

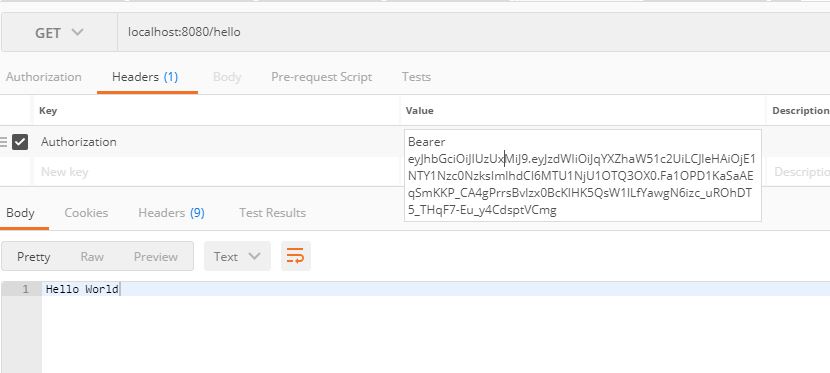
}

Start the Spring Boot Application

* Generate a JSON Web Token -

Create a POST request with url localhost:8080/authenticate. Body should have valid username and password. In our case username is javainuse and password is password.  


* Validate the JSON Web Token

- Try accessing the url localhost:8080/hello using the above generated token in the header as follows  


Download Source Code

Download it -  
[Spring Boot + JWT without JPA](https://www.javainuse.com/zip/spring/sec/spring-boot-jwt-without-JPA.rar)

Popular Posts

* [E-commerce Website - Online Book Store using Angular 8 + Spring Boot](https://www.javainuse.com/fullstack/ecommerce)
* [Spring Boot +JSON Web Token(JWT) Hello World Example](https://www.javainuse.com/spring/boot-jwt)
* [Angular 7 + Spring Boot Application Hello World Example](https://www.javainuse.com/spring/ang7-hello)
* [Build a Real Time Chat Application using Spring Boot + WebSocket + RabbitMQ](https://www.javainuse.com/spring/boot-websocket-chat)
* [Pivotal Cloud Foundry Tutorial - Deploy Spring Boot Application Hello World Example](https://www.javainuse.com/pcf/pcf-hello)
* [Deploying Spring Based WAR Application to Docker](https://www.javainuse.com/devOps/docker/docker-war)
* [EIP patterns using Apache Camel](https://www.javainuse.com/camel/camel_EIP)
* [Spring Cloud- Netflix Eureka + Ribbon Simple Example](https://www.javainuse.com/spring/spring_ribbon)
* [Spring Cloud- Netflix Hystrix Circuit Breaker Simple Example](https://www.javainuse.com/spring/spring_hystrix_circuitbreaker)
* [Spring Boot + Swagger Example Hello World Example](https://www.javainuse.com/spring/boot_swagger)
* [Spring Boot Batch Simple example](https://www.javainuse.com/spring/bootbatch)
* [Spring Boot + Apache Kafka Example](https://www.javainuse.com/spring/spring-boot-apache-kafka-hello-world)
* [Spring Boot Admin Simple Example](https://www.javainuse.com/spring/boot-admin)
* [Spring Boot Security - Introduction to OAuth](https://www.javainuse.com/spring/spring-boot-oauth-introduction)
* [Spring Boot OAuth2 Part 1 - Getting The Authorization Code](https://www.javainuse.com/spring/spring-boot-oauth-authorization-code)
* [Spring Boot OAuth2 Part 2 - Getting The Access Token And Using it to Fetch Data.](https://www.javainuse.com/spring/spring-boot-oauth-access-token)
* [JBoss Drools Hello World-Stateful Knowledge Session using KieSession](https://www.javainuse.com/drools_hello_kie)
* [Understand Drools Stateful vs Stateless Knowledge Session](https://www.javainuse.com/drools_states)
* [JBoss Drools- Understanding Drools Decision Table using Simple Example](https://www.javainuse.com/drools/drools_decision)

See Also

* [Spring Boot Interview Questions](https://www.javainuse.com/spring/SpringBootInterviewQuestions)
* [Spring Batch Interview Questions](https://www.javainuse.com/spring/sprbatch_interview)
* [Spring AOP Interview Questions](https://www.javainuse.com/spring/spring-AOP-interview-quesions)
* [Angular 2 Interview Questions](https://www.javainuse.com/angular/ang2_intvw)
* [Apache Camel Interview Questions](https://www.javainuse.com/camel/Apache_Camel_Questions)
* [JBoss Fuse Interview Questions](https://www.javainuse.com/camel/JBoss_Fuse_Questions)
* [Drools Interview Questions](https://www.javainuse.com/drools/drools_intvw)
* [Java 8 Interview Questions](https://www.javainuse.com/java/java8_intvw)
* [Spring Cloud Interview Questions](https://www.javainuse.com/spring/spring-cloud-interview-questions)
* [Microservices Interview Questions](https://www.javainuse.com/spring/microservices-interview-quesions)
* [Java HashMap and ConcurrentHashMap Interview Questions](https://www.javainuse.com/java/java_map_intvw)
* [Mule ESB frequently asked interview questions](https://www.javainuse.com/misc/muleintvw)
* [Apache Kafka Interview Questions](https://www.javainuse.com/misc/apache-kafka-interview-questions)
* [Tosca Testing Tool Interview Questions](https://www.javainuse.com/misc/tosca-testing-tool-interview-questions)
* [Top Maven Build Tool Interview Questions](https://www.javainuse.com/misc/maven-interview-questions)
* [Top Gradle Build Tool Interview Questions](https://www.javainuse.com/misc/gradle-interview-questions)
* [Miscellaneous Topics](https://www.javainuse.com/misc)

© Copyright JavaInUse. All Rights Reserved.  
[Privacy Policy](https://www.javainuse.com/privacy)

----------------------------------------------------------------------------------------------------------------------------------

* [Home](https://www.javainuse.com/home)
* [Java](https://www.javainuse.com/)
* [Spring](https://www.javainuse.com/)
* [Angular](https://www.javainuse.com/)
* [Full Stack](https://www.javainuse.com/)
* [Apache Camel](https://www.javainuse.com/)
* [Cloud Frameworks](https://www.javainuse.com/)
* [Messaging](https://www.javainuse.com/)
* [Drools](https://www.javainuse.com/)
* [Search Engine](https://www.javainuse.com/)
* [Hazelcast](https://www.javainuse.com/hazelcast)
* [DevOps](https://www.javainuse.com/)

#### Search Tutorials

Top of Form

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Bottom of Form

# Spring Boot Security + JWT + MySQL Hello World Example

[**Close X**](https://www.javainuse.com/)

In a [previous tutorial we had implemented Spring Boot + JWT Authentication Example](https://www.javainuse.com/spring/boot-jwt)  
We were making use of hard coded user values for User Authentication. In this tutorial we will be implementing MYSQL JPA for storing and fetching user credentials.

Spring Boot JSON Web Token- Table of Contents

[What is JWT(JSON Web Token)](https://www.javainuse.com/spring/jwt)

[Spring Boot +JSON Web Token(JWT) Hello World Example](https://www.javainuse.com/spring/boot-jwt)

[**Spring Boot +JSON Web Token(JWT) + MYSQL Example**](https://www.javainuse.com/spring/boot-jwt-mysql)

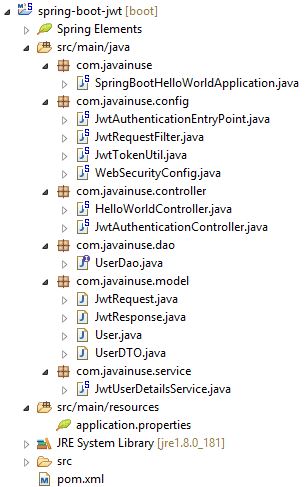
[Angular 7 + Spring Boot JWT Authentication Hello World Example](https://www.javainuse.com/spring/ang7-jwt)

Video

This tutorial is explained in the below Youtube Video.

Lets Begin?

## Implement MYSQL JPA for storing and fetching user details

The starting code for this tutorial will be the [Spring Boot + JWT Hello World Example](https://www.javainuse.com/spring/boot-jwt) we had implemented previously. Currently using JwtUserDetailsService we are validating the user. We are doing this using hard coded values for username and password. Now we will be using Spring Data JPA to validate user credentials by fetching username and password from the mysql db. The maven project will be as follows-  
  
Define the pom.xml as follows-

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>spring-boot-jwt-tr</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

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

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

<version>2.1.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

**<dependency>**

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

**<artifactId>spring-boot-starter-data-jpa</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>mysql</groupId>**

**<artifactId>mysql-connector-java</artifactId>**

**</dependency>**

</dependencies>

</project>

### Inserting a user

Define the database properties as follows-

jwt.secret=javainuse

spring.datasource.url=jdbc:mysql://localhost/bootdb?createDatabaseIfNotExist=true&autoReconnect=true&useSSL=false

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.platform=mysql

spring.jpa.hibernate.ddl-auto=create-drop

In a previous tutorial we had implemented [Spring Boot + Spring Data JPA Hello World Example](https://www.javainuse.com/spring/SpringBoot_DataJPA). Create the Entity class as follows. It will be used while performing database operations-

package com.javainuse.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import javax.persistence.\*;

@Entity

@Table(name = "user")

public class DAOUser {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private long id;

@Column

private String username;

@Column

@JsonIgnore

private String password;

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;

}

}

Define the UserDTO model class as follows. It is responsible for getting values from user and passing it to the DAO layer for inserting in database.

package com.javainuse.model;

public class UserDTO {

private String username;

private String password;

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;

}

}

Next we define the UserDao which is an interface that extends the Spring Framework class CrudRepository. CrudRepository class is a generics and takes the following two parameters as arguments- What type of Object will this repository be working with- In our case DAOUser and Id will be what type of object- Integer(since id defined in the UserDao class is Integer) Thats the only configuration required for the repository class. The required operation of inserting user details in DB will now be handled. Define the DAO class as follows.

package com.javainuse.dao;

import org.springframework.data.repository.CrudRepository;

import org.springframework.stereotype.Repository;

import com.javainuse.model.DAOUser;

@Repository

public interface UserDao extends CrudRepository<DAOUser, Integer> {

}

Allow the url /register to be allowed without applying spring security-

package com.javainuse.config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

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

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

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

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

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

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

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

@Autowired

private UserDetailsService jwtUserDetailsService;

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

// configure AuthenticationManager so that it knows from where to load

// user for matching credentials

// Use BCryptPasswordEncoder

auth.userDetailsService(jwtUserDetailsService).passwordEncoder(passwordEncoder());

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Override

protected void configure(HttpSecurity httpSecurity) throws Exception {

// We don't need CSRF for this example

httpSecurity.csrf().disable()

// dont authenticate this particular request

.authorizeRequests().antMatchers**("/authenticate", "/register")**.permitAll().

// all other requests need to be authenticated

anyRequest().authenticated().and().

// make sure we use stateless session; session won't be used to

// store user's state.

exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint).and().sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS);

// Add a filter to validate the tokens with every request

httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

}

}

In the JwtUserDetailsService, autowire the UserDao bean and the BcryptEncoder bean. Also define the saveUser function for inserting user details-

package com.javainuse.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.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import com.javainuse.dao.UserDao;

import com.javainuse.model.DAOUser;

import com.javainuse.model.UserDTO;

public class JwtUserDetailsService implements UserDetailsService {

@Autowired

private UserDao userDao;

@Autowired

private PasswordEncoder bcryptEncoder;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

if ("javainuse".equals(username)) {

return new User("javainuse", "$2a$10$slYQmyNdGzTn7ZLBXBChFOC9f6kFjAqPhccnP6DxlWXx2lPk1C3G6",

new ArrayList<>());

} else {

throw new UsernameNotFoundException("User not found with username: " + username);

}

}

**public UserDao save(UserDTO user) {**

**DAOUser newUser = new DAOUser();**

**newUser.setUsername(user.getUsername());**

**newUser.setPassword(bcryptEncoder.encode(user.getPassword()));**

**return userDao.save(newUser);**

**}**

}

Finally modify the Controller class for adding a POST request for adding user details to database.

package com.javainuse.controller;

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

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.authentication.DisabledException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

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

import org.springframework.web.bind.annotation.CrossOrigin;

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

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

import org.springframework.web.bind.annotation.RequestMethod;

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

import com.javainuse.config.JwtTokenUtil;

import com.javainuse.model.JwtRequest;

import com.javainuse.model.JwtResponse;

import com.javainuse.model.UserDTO;

import com.javainuse.service.JwtUserDetailsService;

@RestController

@CrossOrigin

public class JwtAuthenticationController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtTokenUtil jwtTokenUtil;

@Autowired

private JwtUserDetailsService userDetailsService;

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

public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest authenticationRequest) throws Exception {

authenticate(authenticationRequest.getUsername(), authenticationRequest.getPassword());

final UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());

final String token = jwtTokenUtil.generateToken(userDetails);

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

}

**@RequestMapping(value = "/register", method = RequestMethod.POST)**

**public ResponseEntity<?> saveUser(@RequestBody UserDTO user) throws Exception {**

**return ResponseEntity.ok(userDetailsService.save(user));**

**}**

private void authenticate(String username, String password) throws Exception {

try {

authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));

} catch (DisabledException e) {

throw new Exception("USER\_DISABLED", e);

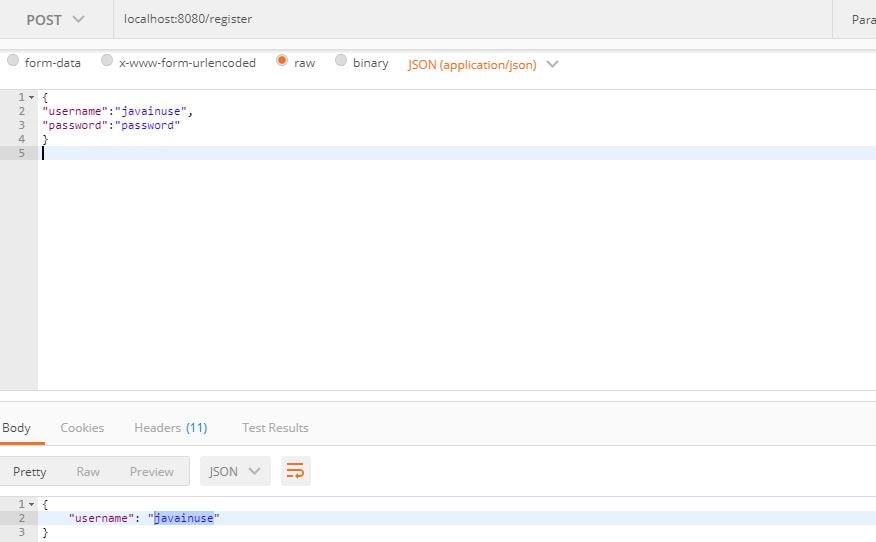
} catch (BadCredentialsException e) {

throw new Exception("INVALID\_CREDENTIALS", e);

}

}

}

Start the Spring Boot Application- Register a new user by creating a post request to url /register and the body having username and password  


### Make use of Database credentials for authentication

In the UserDao interface add a method findByUsername(String username)

package com.javainuse.dao;

import org.springframework.data.repository.CrudRepository;

import org.springframework.stereotype.Repository;

import com.javainuse.model.DAOUser;

@Repository

public interface UserDao extends CrudRepository<DAOUser, Integer> {

UserDao findByUsername(String username);

}

In the loadUserByUsername method, we fetch the user records from the database instead of using hardcoded value.

package com.javainuse.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.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import com.javainuse.dao.UserDao;

import com.javainuse.model.DAOUser;

import com.javainuse.model.UserDTO;

@Service

public class JwtUserDetailsService implements UserDetailsService {

@Autowired

private UserDao userDao;

@Autowired

private PasswordEncoder bcryptEncoder;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

**DAOUser user = userDao.findByUsername(username);**

if (user == null) {

throw new UsernameNotFoundException("User not found with username: " + username);

}

return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(),

new ArrayList<>());

}

public User save(UserDTO user) {

DAOUser newUser = new DAOUser();

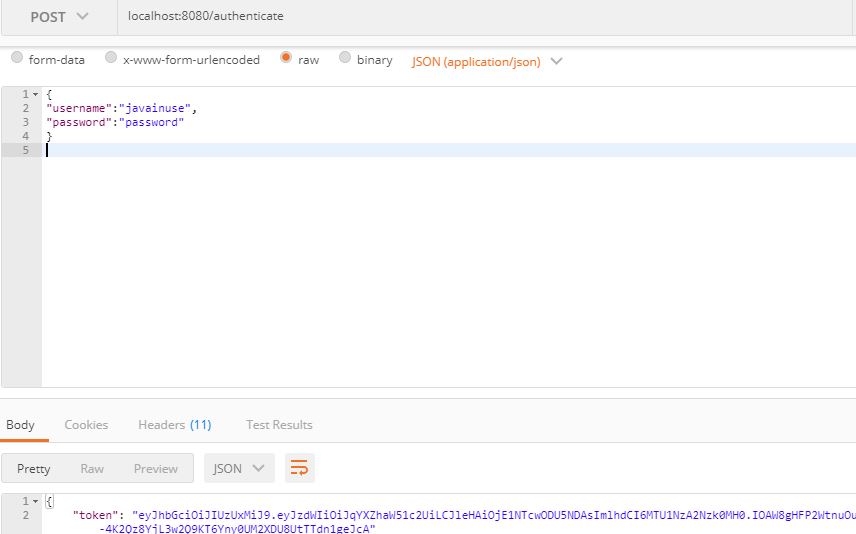
newUser.setUsername(user.getUsername());

newUser.setPassword(bcryptEncoder.encode(user.getPassword()));

return userDao.save(newUser);

}

}

Generate a new Token by creating a post request to url /authenticate and the body having username and password  


## Download Source Code

Download it -  
[Spring Boot + JWT with JPA](https://www.javainuse.com/zip/spring/sec/spring-boot-jwt-JPA.rar)

## Popular Posts

* [E-commerce Website - Online Book Store using Angular 8 + Spring Boot](https://www.javainuse.com/fullstack/ecommerce)
* [Spring Boot +JSON Web Token(JWT) Hello World Example](https://www.javainuse.com/spring/boot-jwt)
* [Angular 7 + Spring Boot Application Hello World Example](https://www.javainuse.com/spring/ang7-hello)
* [Build a Real Time Chat Application using Spring Boot + WebSocket + RabbitMQ](https://www.javainuse.com/spring/boot-websocket-chat)
* [Pivotal Cloud Foundry Tutorial - Deploy Spring Boot Application Hello World Example](https://www.javainuse.com/pcf/pcf-hello)
* [Deploying Spring Based WAR Application to Docker](https://www.javainuse.com/devOps/docker/docker-war)
* [EIP patterns using Apache Camel](https://www.javainuse.com/camel/camel_EIP)
* [Spring Cloud- Netflix Eureka + Ribbon Simple Example](https://www.javainuse.com/spring/spring_ribbon)
* [Spring Cloud- Netflix Hystrix Circuit Breaker Simple Example](https://www.javainuse.com/spring/spring_hystrix_circuitbreaker)
* [Spring Boot + Swagger Example Hello World Example](https://www.javainuse.com/spring/boot_swagger)
* [Spring Boot Batch Simple example](https://www.javainuse.com/spring/bootbatch)
* [Spring Boot + Apache Kafka Example](https://www.javainuse.com/spring/spring-boot-apache-kafka-hello-world)
* [Spring Boot Admin Simple Example](https://www.javainuse.com/spring/boot-admin)
* [Spring Boot Security - Introduction to OAuth](https://www.javainuse.com/spring/spring-boot-oauth-introduction)
* [Spring Boot OAuth2 Part 1 - Getting The Authorization Code](https://www.javainuse.com/spring/spring-boot-oauth-authorization-code)
* [Spring Boot OAuth2 Part 2 - Getting The Access Token And Using it to Fetch Data.](https://www.javainuse.com/spring/spring-boot-oauth-access-token)
* [JBoss Drools Hello World-Stateful Knowledge Session using KieSession](https://www.javainuse.com/drools_hello_kie)
* [Understand Drools Stateful vs Stateless Knowledge Session](https://www.javainuse.com/drools_states)
* [JBoss Drools- Understanding Drools Decision Table using Simple Example](https://www.javainuse.com/drools/drools_decision)

## See Also

* [Spring Boot Interview Questions](https://www.javainuse.com/spring/SpringBootInterviewQuestions)
* [Spring Batch Interview Questions](https://www.javainuse.com/spring/sprbatch_interview)
* [Spring AOP Interview Questions](https://www.javainuse.com/spring/spring-AOP-interview-quesions)
* [Angular 2 Interview Questions](https://www.javainuse.com/angular/ang2_intvw)
* [Apache Camel Interview Questions](https://www.javainuse.com/camel/Apache_Camel_Questions)
* [JBoss Fuse Interview Questions](https://www.javainuse.com/camel/JBoss_Fuse_Questions)
* [Drools Interview Questions](https://www.javainuse.com/drools/drools_intvw)
* [Java 8 Interview Questions](https://www.javainuse.com/java/java8_intvw)
* [Spring Cloud Interview Questions](https://www.javainuse.com/spring/spring-cloud-interview-questions)
* [Microservices Interview Questions](https://www.javainuse.com/spring/microservices-interview-quesions)
* [Java HashMap and ConcurrentHashMap Interview Questions](https://www.javainuse.com/java/java_map_intvw)
* [Mule ESB frequently asked interview questions](https://www.javainuse.com/misc/muleintvw)
* [Apache Kafka Interview Questions](https://www.javainuse.com/misc/apache-kafka-interview-questions)
* [Tosca Testing Tool Interview Questions](https://www.javainuse.com/misc/tosca-testing-tool-interview-questions)
* [Top Maven Build Tool Interview Questions](https://www.javainuse.com/misc/maven-interview-questions)
* [Top Gradle Build Tool Interview Questions](https://www.javainuse.com/misc/gradle-interview-questions)
* [Miscellaneous Topics](https://www.javainuse.com/misc)

© Copyright JavaInUse. All Rights Reserved.  
[Privacy Policy](https://www.javainuse.com/privacy)