* [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/)
* [Webseries](https://www.javainuse.com/webseries)
* [Crypto Tools](https://www.javainuse.com/tools)
* [Online Certifications](https://www.javainuse.com/cert)

Search Tutorials

Top of Form

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

Bottom of Form

**Spring Boot Security + JWT + MySQL Hello World Example**

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)

[Online JWT Generator](https://www.javainuse.com/jwtgenerator)

[Online JWT Decoder](https://www.javainuse.com/decodeJWT)

[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)

[Spring Boot RestTemplate + JWT Authentication Example](https://www.javainuse.com/webseries/spring-security-jwt/chap6)

[Spring Boot Security - Refresh Expired JSON Web Token](https://www.javainuse.com/webseries/spring-security-jwt/chap7)

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

[Online JWT Generator](https://www.javainuse.com/jwtgenerator)

[Online JWT Decoder](https://www.javainuse.com/decodeJWT)

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-

Play

Unmute

Loaded: 1.18%

Fullscreen

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

* [Spring Boot Interview Questions](https://www.javainuse.com/spring/SpringBootInterviewQuestions)
* [Implement Spring Boot Security and understand Spring Security Architecture](https://www.javainuse.com/webseries/spring-security-jwt/chap3)
* [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 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)
* [Snowflake frequently asked interview questions](https://www.javainuse.com/prep/snowflake)
* [SAP FI - Accounts Receivable frequently asked interview questions](https://www.javainuse.com/prep/sap1)
* [Top SAP ALV Interview Questions](https://www.javainuse.com/prep/sap3)
* [Top SAP Business Objects Administration Interview Questions](https://www.javainuse.com/prep/sap2)
* [EC2 frequently asked interview questions](https://www.javainuse.com/prep/ec2)
* [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)