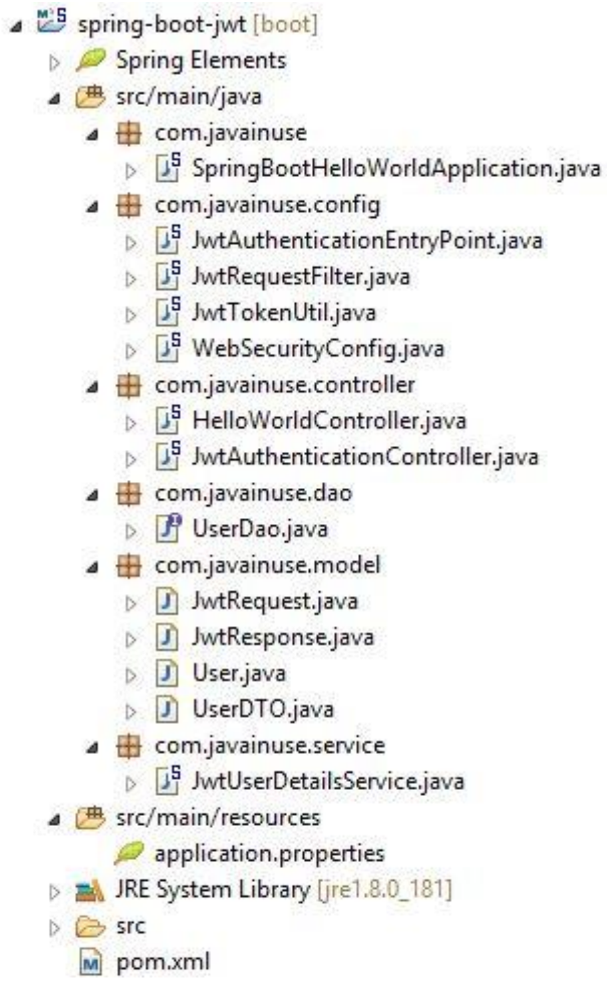
Spring Boot Security + JWT + MySQL

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-



```
spring-boot-jwt [boot]
    Spring Elements
    src/main/java
        com.javainuse
            SpringBootHelloWorldApplication.java
        com.javainuse.config
            JwtAuthenticationEntryPoint.java
            JwtRequestFilter.java
            JwtTokenUtil.java
            WebSecurityConfig.java
        com.javainuse.controller
            HelloWorldController.java
            JwtAuthenticationController.java
        com.javainuse.dao
            UserDao.java
        com.javainuse.model
            JwtRequest.java
            JwtResponse.java
            User.java
            UserDTO.java
        com.javainuse.service
            JwtUserDetailsService.java
    src/main/resources
        application.properties
    JRE System Library [jre1.8.0_181]
    src
    pom.xml
```

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.output
Encoding>
                <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
```

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.Auth
enticationManagerBuilder;
```

```
import org.springframework.security.config.annotation.method.configuration.EnableG
lobalMethodSecurity;

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

import org.springframework.security.config.annotation.web.configuration.EnableWebS
ecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurit
yConfigurerAdapter;

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.UsernamePasswordAuthenticat
ionFilter;


@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 Exc
eption {

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

                // user for matching credentials

                // Use BCryptPasswordEncoder

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

        }
        @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 w
on't be used to
                                // store user's state.
                                exceptionHandling().authenticationEntryPoint(jwt
AuthenticationEntryPoint).and().sessionManagement()
                                .sessionCreationPolicy(SessionCreationPolicy.STA
TELESS);

                // Add a filter to validate the tokens with every request
                httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAu
thenticationFilter.class);
        }
}
```

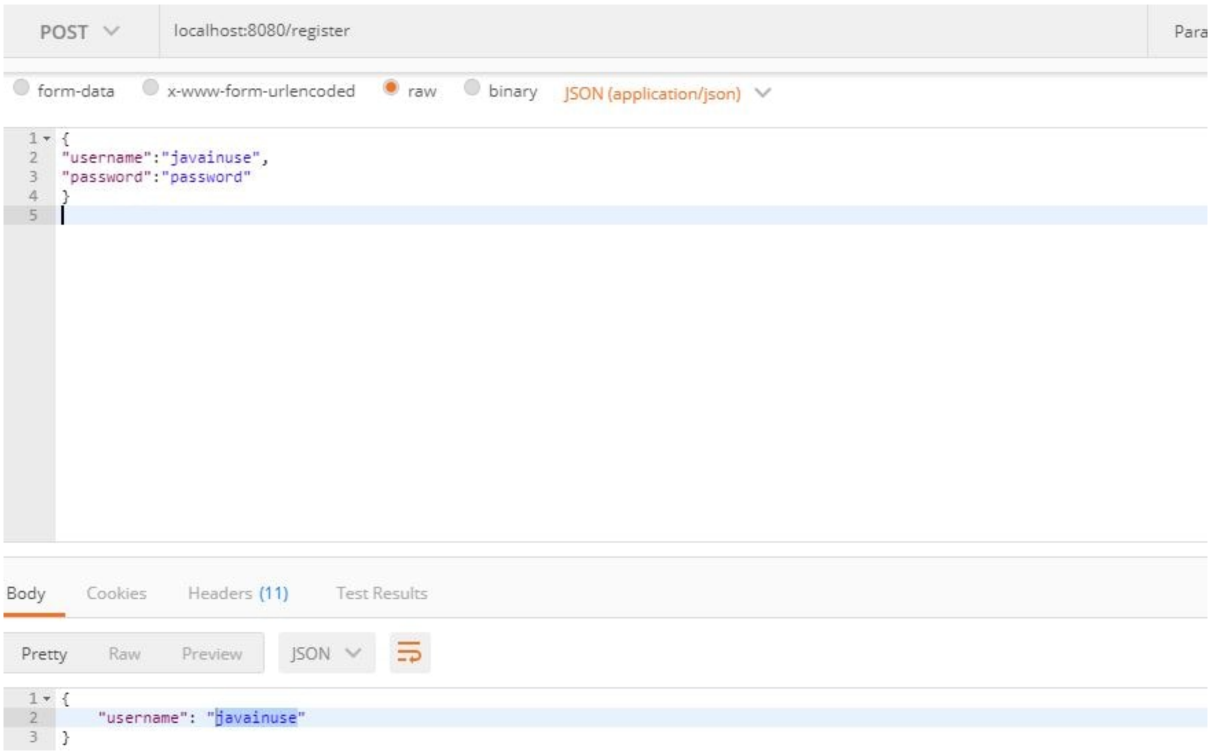
In the JwtUserDetailsService, autowire the UserDao bean and the BcryptEncoder bean. Also define the saveUser function for inserting user details-

```
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 UsernameNot
FoundException {
                if ("javainuse".equals(username)) {
                        return new User("javainuse", "$2a$10$slYQmyNdGzTn7ZLBXBCh
FOC9f6kFjAqPhccnP6DxlWXx2lPk1C3G6",
                                        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.UsernamePasswordAuthenticationT
oken;
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 JwtReques
t authenticationRequest) throws Exception {


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


                final UserDetails userDetails = userDetailsService.loadUserByUser
name(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 Excep
tion {
                return ResponseEntity.ok(userDetailsService.save(user));
        }


        private void authenticate(String username, String password) throws Except
ion {
                try {
                        authenticationManager.authenticate(new UsernamePasswordAu
thenticationToken(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



POST localhost:8080/register

form-data x-www-form-urlencoded [✓ raw] binary JSON (application/json)

```
{
"username":"javainuse",
"password":"password"
}
```

```
{
    "username": "javainuse"
}
```

Make use of Database credentials for authentication

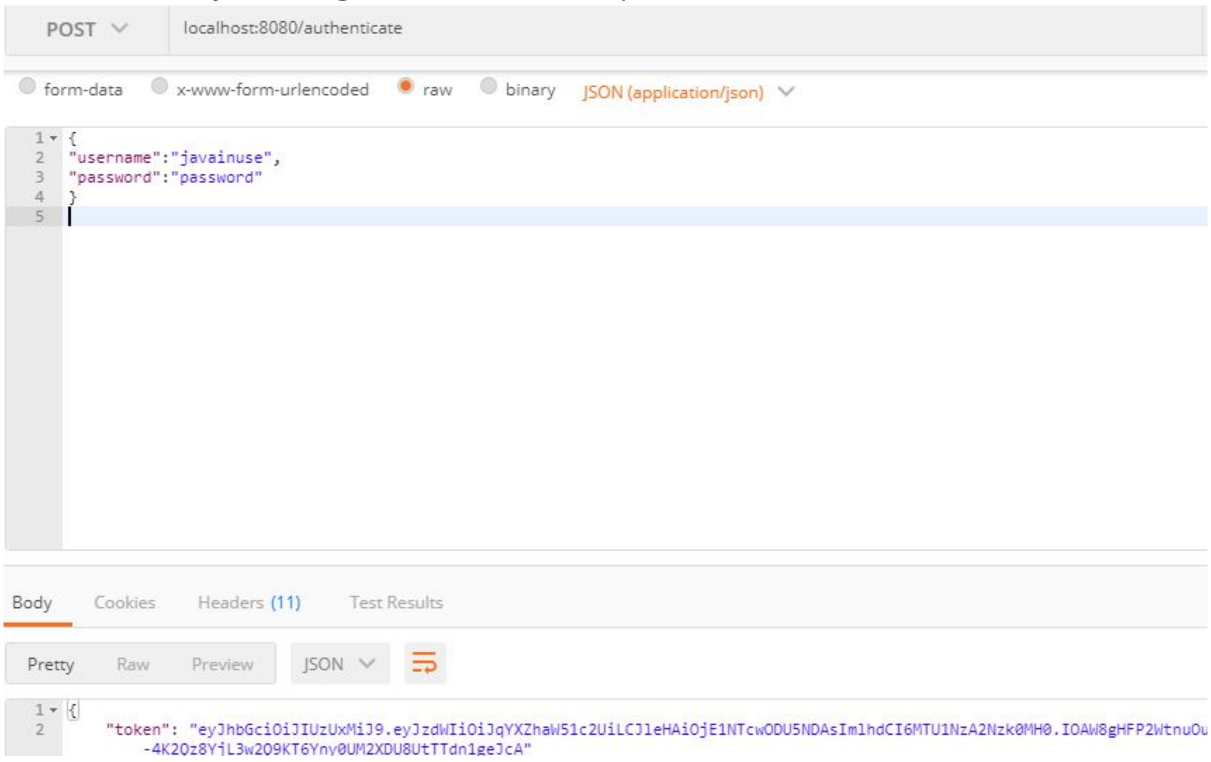
In the UserDao interface add a method findByUsername(String username)

```
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 UsernameNot
FoundException {


                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



localhost:8080/authenticate

JSON (application/json)

```
{
"username":"javainuse",
"password":"password"
}
```

```
{
    "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJqYXZhaW51c2UiLCJleHAiOjE1NTcwODU5NDAsImlhdCI6MTU1NzA2Nzk0MH0.IOAW8gHFP2WtnuOu
      -4K2Oz8YjL3w2O9KT6Yny0UM2XDU8UtTTdn1geJcA"
```