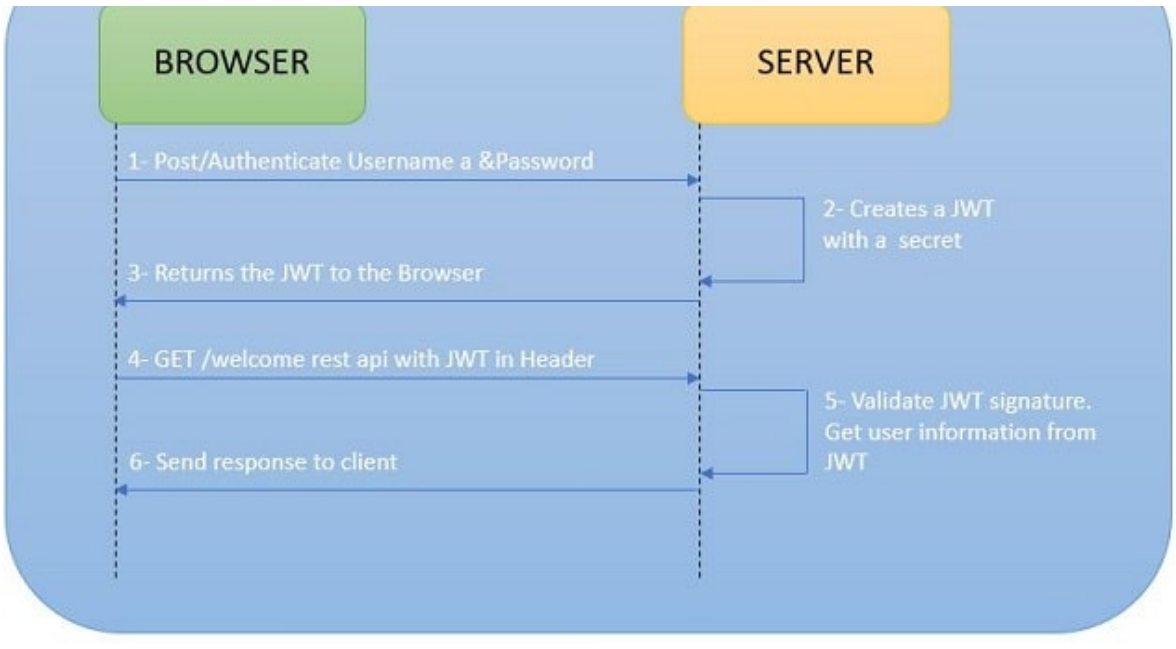
Spring Boot Security + JWT (JSON Web Token) Authentication Example

In this tutorial, we will create a Spring Boot Application that uses JWT authentication to protect an exposed REST API. We will Configure JWT's Spring Security. Use the REST POST API to map / authenticate which user will receive a valid JSON Web Token. And then the user can only access the api / welcome if it has a valid token. We've seen what's JWT, when and how to use it in a previous tutorial.

Spring Boot Security Tutorial :

Spring Boot Rest Authentication with JWT Token Flow



- Customers sign in by submitting their credentials to the provider.
- Upon successful authentication, it generates JWT containing user details and privileges for accessing the services and sets the JWT expiry date in payload.
- The server signs and encrypts the JWT if necessary and sends it to the client as a response with credentials to the initial request.
- Based on the expiration set by the server, the customer/client stores the JWT for a restricted or infinite amount of time.
- The client sends this JWT token in the header for all subsequent requests.
- The client authenticates the user with this token. So we don't need the client to send the user name and password to the server during each authentication process, but only once the server sends the client a JWT.

In next tutorial, we have integrated Angular 8 with Spring Boot JWT Authentication.

Take a look at our suggested posts:

Spring Boot Session Management using JDBC Example

Spring Boot Session Management using Redis Example

Spring Boot - Transaction Management

Spring Boot - Hazelcast

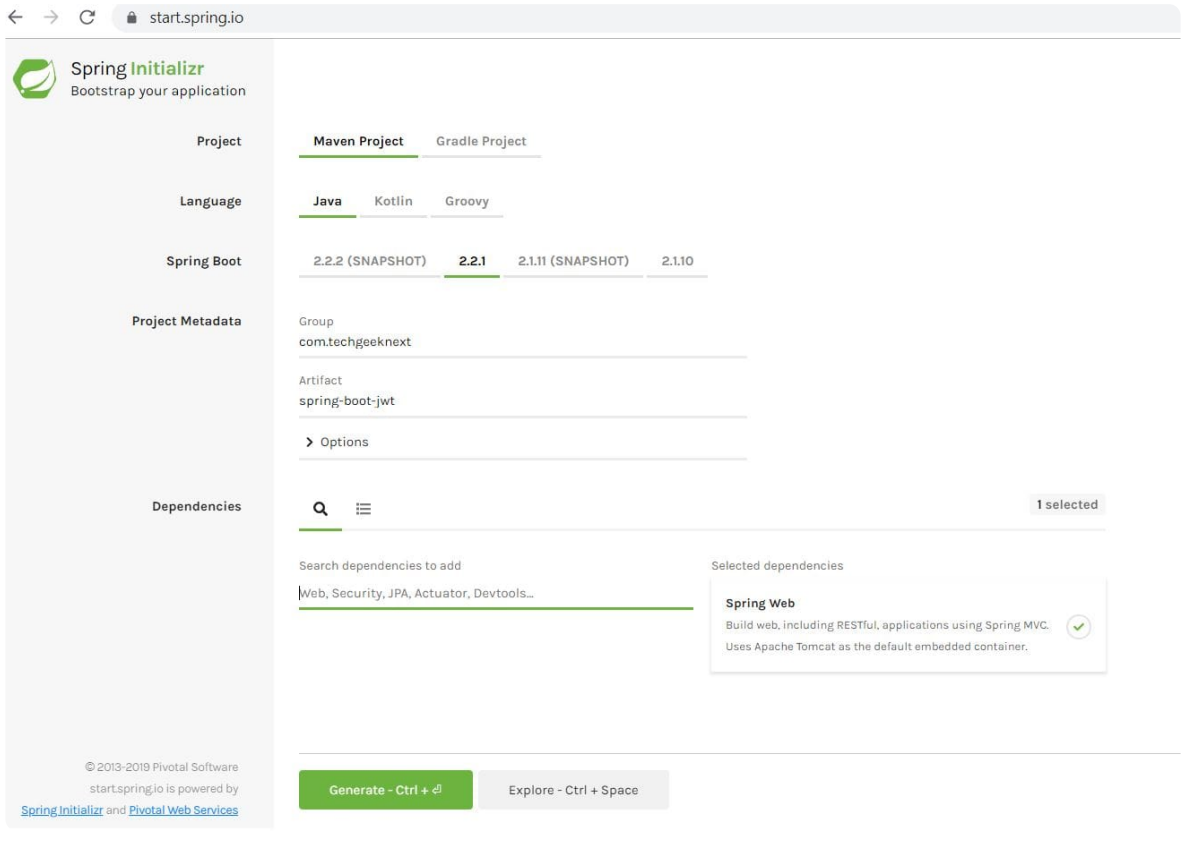
Java 15

Java 14

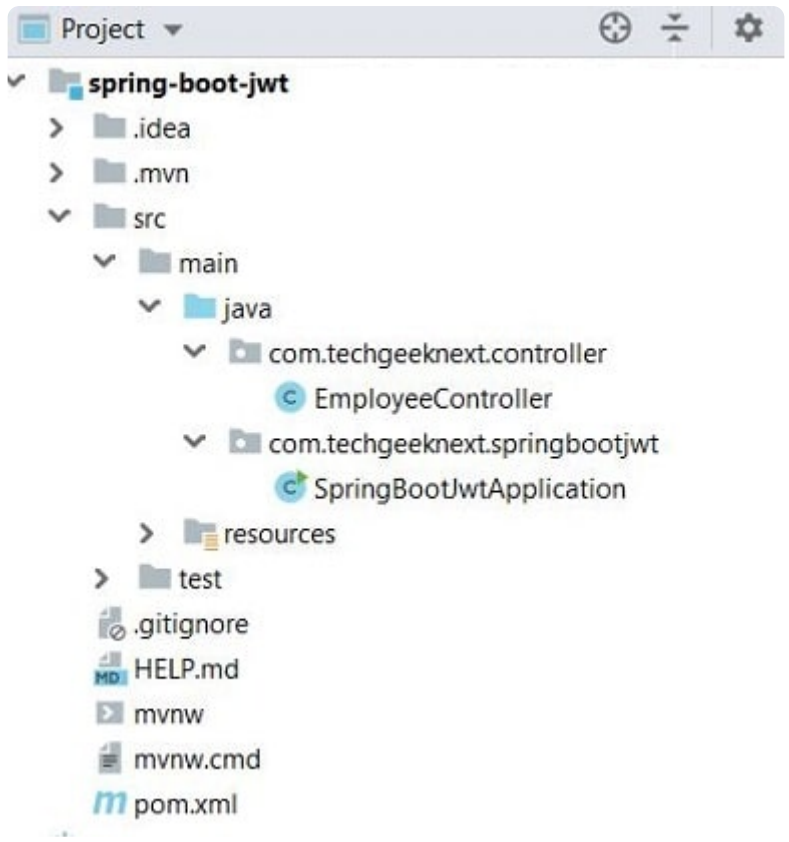
Java 8 Interview Questions and Answers

Create Simple Spring boot with /greeting rest end point

https://start.spring.io/



Project Structure



pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="ht
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https:/
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.techgeeknext</groupId>
    <artifactId>spring-boot-jwt</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-boot-jwt</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <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-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId
            </plugin>
        </plugins>
    </build>

</project>
```

Create Controller with /greeting Rest api

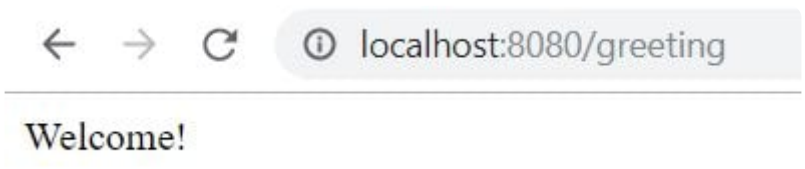
```
package com.techgeeknext.controller;

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

@RestController
public class EmployeeController {
    @RequestMapping({ "/greeting" })
    public String welcomePage() {
        return "Welcome!";
    }
}
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplicat

@SpringBootApplication
public class SpringBootJwtApplication {

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

}
```

Test */greeting* GET Api without JWT

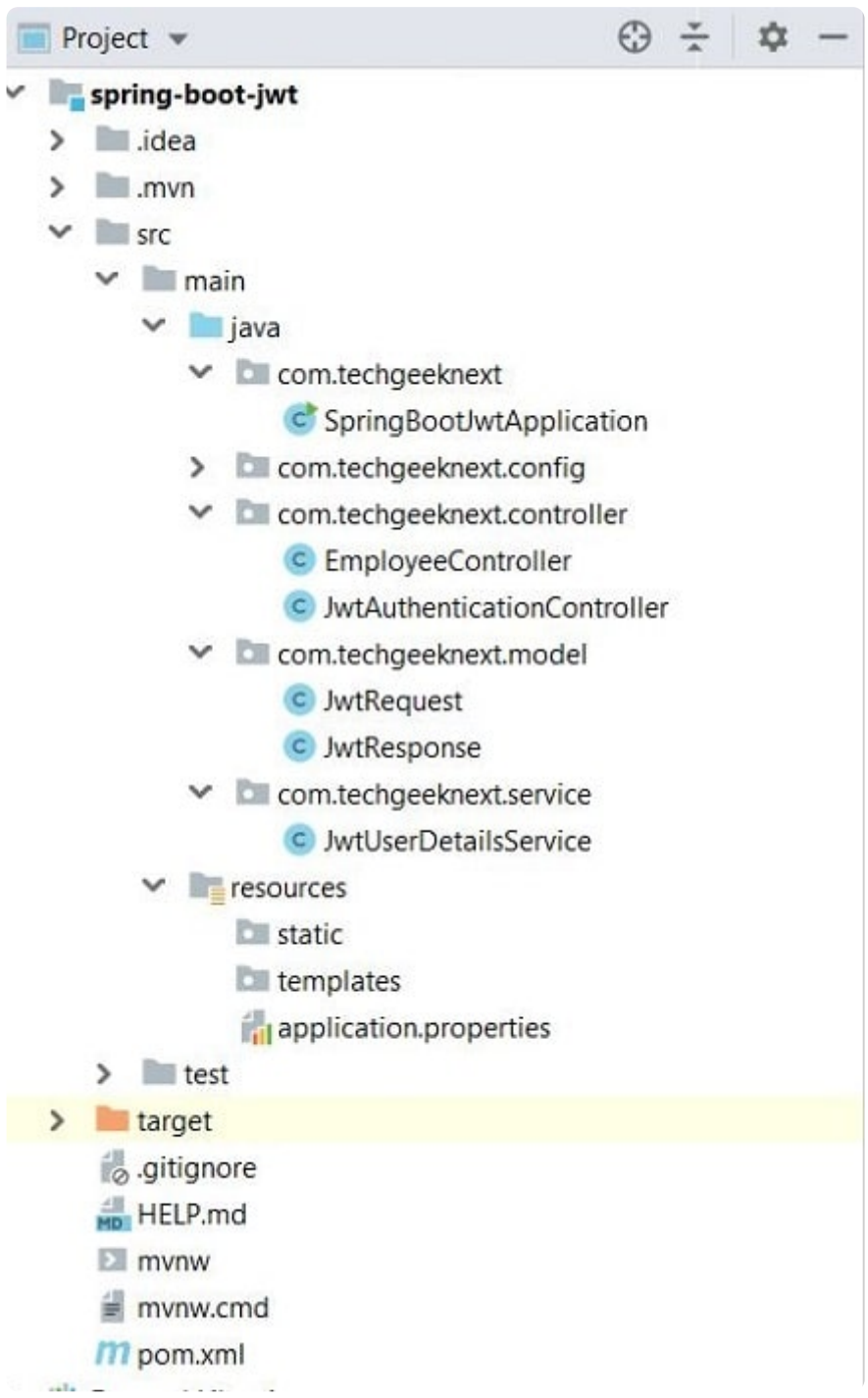
Compile and the run this project by using endpoint

localhost:8080/greeting.



Project Structure

Now will add spring security and JWT into our project.



pom.xml:

Add Spring Security and JWT dependencies as given below.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="ht
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https:/
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.2.1.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.techgeeknext</groupId>
    <artifactId>spring-boot-jwt</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-boot-jwt</name>
    <description>Demo project for Spring Security with JWT</desc

    <properties>
        <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-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId
            </plugin>
        </plugins>
    </build>

</project>
```

application.properties

Provide secret key. As seen in previous JWT introduction, we provided the secret key used by the hashing algorithm . JWT combined this secret key with header and payload data.

Spring Security and JWT Configuration

We will be performing 2 operation to configure spring security and to generate JWT and to validate it.

- Generate JWT : Use */authenticate* POST endpoint by using username and password to generate a JSON Web Token (JWT).
- Validate JWT : User can use */greeting* GET endpoint by using valid JSON Web Token (JWT).

JWT Token Utility

We will define the utilities method for generating and validating JWT token.

```java
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 = -255018516562600

    public static final long JWT_TOKEN_VALIDITY = 5*60*60;

    @Value("")
    private String secret;

    public String getUsernameFromToken(String token) {
        return getClaimFromToken(token, Claims::getSubject);
    }

    public Date getIssuedAtDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getIssuedAt);
    }

    public Date getExpirationDateFromToken(String token) {
        return getClaimFromToken(token, Claims::getExpiration);
    }

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

    private Claims getAllClaimsFromToken(String token) {
        return Jwts.parser().setSigningKey(secret).parseClaimsJw
    }

    private Boolean isTokenExpired(String token) {
        final Date expiration = getExpirationDateFromToken(token
        return expiration.before(new Date());
    }

    private Boolean ignoreTokenExpiration(String token) {
        // here you specify tokens, for that the expiration is ig
        return false;
    }

    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        return doGenerateToken(claims, userDetails.getUsername()
    }

    private String doGenerateToken(Map<String, Object> claims, St

        return Jwts.builder().setClaims(claims).setSubject(subje
                .setExpiration(new Date(System.currentTimeMillis
    }
```

```
    }

    public Boolean validateToken(String token, UserDetails userD
        final String username = getUsernameFromToken(token);
        return (username.equals(userDetails.getUsername()) && !is
    }
}
```

Load Username and Password

We used UserDetailsService interface from `org.springframework.security.core.userdetails.UserDetailsService` package provided by the spring security.

UserDetailsService interface is used in order to search the username, password and *GrantedAuthorities* for given user.

This interface provide only one method called `loadUserByUsername`. Authentication Manager calls this method for getting the user details from the database when authenticating the user details provided by the user.

NOTE: In this tutorial will be using hard coded username password. Use BCrypt password, can use any online tool to BCrypt the password. In Next coming tutorial will integrate with MySql database to get username and password.

```
package com.techgeeknext.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.UserDetailsS
import org.springframework.security.core.userdetails.UsernameNotF
import org.springframework.stereotype.Service;

@Service
public class JwtUserDetailsService implements UserDetailsService

    @Override
    public UserDetails loadUserByUsername(String username) throws
        if ("techgeeknext".equals(username)) {
            return new User("techgeeknext", "$2a$10$slYQmyNdGzTn
                    new ArrayList<>());
        } else {
            throw new UsernameNotFoundException("User not found w
        }
    }

}
```

Jwt Authentication Controller

We will use */authenticate POST API* to authenticate username and password. Username and Password will passed in body and using Authentication Manager will authenticate the credentials. If credentials

to client.

```
package com.techgeeknext.controller;

import java.util.Objects;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.Authenticatio
import org.springframework.security.authentication.BadCredential
import org.springframework.security.authentication.DisabledExcep
import org.springframework.security.authentication.UsernamePassw
import org.springframework.security.core.userdetails.UserDetails
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.techgeeknext.config.JwtTokenUtil;
import com.techgeeknext.model.JwtRequest;
import com.techgeeknext.model.JwtResponse;

@RestController
@CrossOrigin
public class JwtAuthenticationController {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Autowired
    private UserDetailsService jwtInMemoryUserDetailsService;

    @RequestMapping(value = "/authenticate", method = RequestMet
    public ResponseEntity<?> generateAuthenticationToken(@Reques
            throws Exception {

        authenticate(authenticationRequest.getUsername(), authen

        final UserDetails userDetails = jwtInMemoryUserDetailsSe
                .loadUserByUsername(authenticationRequest.getUse

        final String token = jwtTokenUtil.generateToken(userDeta

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

    private void authenticate(String username, String password)
        Objects.requireNonNull(username);
        Objects.requireNonNull(password);
        try {
            authenticationManager.authenticate(new UsernamePassw
        } catch (DisabledException e) {
            throw new Exception("USER_DISABLED", e);
        } catch (BadCredentialsException e) {
            throw new Exception("INVALID_CREDENTIALS", e);
        }
    }
}
```

Request Object

password from the client.

```
package com.techgeeknext.model;

import java.io.Serializable;

public class JwtRequest implements Serializable {

    private static final long serialVersionUID = 5926468583005150

    private String username;
    private String password;

    //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;
    }
}
```

Response Object

JwtResponse used to create the response object send to the client.

```
package com.techgeeknext.model;

import java.io.Serializable;

public class JwtResponse implements Serializable {

    private static final long serialVersionUID = -8091879091924046

    private final String jwttoken;

    public JwtResponse(String jwttoken) {
        this.jwttoken = jwttoken;
    }

    public String getToken() {
        return this.jwttoken;
    }
}
```

JWT Request Filter

validate JWT from the request and sets it in the context to indicate that logged in user is authenticated.

```
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.UsernamePassw
import org.springframework.security.core.context.SecurityContextH
import org.springframework.security.core.userdetails.UserDetails
import org.springframework.security.web.authentication.WebAuthen
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import com.techgeeknext.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, H
            throws ServletException, IOException {

        final String requestTokenHeader = request.getHeader("Aut

        String username = null;
        String jwtToken = null;
        // JWT Token is in the form "Bearer token". Remove Bearer
        if (requestTokenHeader != null && requestTokenHeader.sta
            jwtToken = requestTokenHeader.substring(7);
            try {
                username = jwtTokenUtil.getUsernameFromToken(jwt
            } 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 Str
        }

        //Once we get the token validate it.
        if (username != null && SecurityContextHolder.getContext

            UserDetails userDetails = this.jwtUserDetailsService

            // if token is valid configure Spring Security to ma
            if (jwtTokenUtil.validateToken(jwtToken, userDetails

                UsernamePasswordAuthenticationToken usernamePass
                        userDetails, null, userDetails.getAuthor
                usernamePasswordAuthenticationToken
                        .setDetails(new WebAuthenticationDetails
                // After setting the Authentication in the conte
                // that the current user is authenticated. So it
                SecurityContextHolder.getContext().setAuthentica
```

```
    }

}
```

JwtAuthenticationEntryPoint

This class rejects unauthenticated request and send error code 401

```
package com.techgeeknext.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 Authenticati

    private static final long serialVersionUID = -785886955895324

    @Override
    public void commence(HttpServletRequest request, HttpServlet
            AuthenticationException authException) throws IOExce

        response.sendError(HttpServletResponse.SC_UNAUTHORIZED,
    }
}
```

WebSecurityConfig

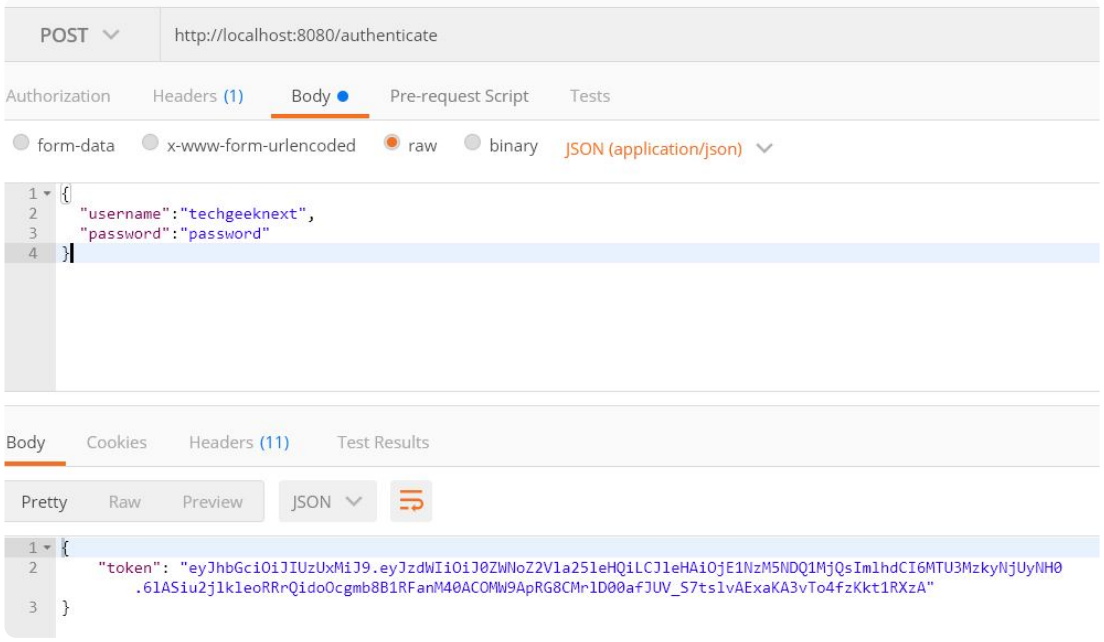
This class extends the WebSecurityConfigurerAdapter that enables both WebSecurity and HTTPSecurity to be customized.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.Authenticatio
import org.springframework.security.config.annotation.authentica
import org.springframework.security.config.annotation.method.con
import org.springframework.security.config.annotation.web.builde
import org.springframework.security.config.annotation.web.config
import org.springframework.security.config.annotation.web.config
import org.springframework.security.config.http.SessionCreationP
import org.springframework.security.core.userdetails.UserDetails
import org.springframework.security.crypto.bcrypt.BCryptPassword
import org.springframework.security.crypto.password.PasswordEnco
import org.springframework.security.web.authentication.UsernameP

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdap

    @Autowired
    private JwtAuthenticationEntryPoint jwtAuthenticationEntryPo

    @Autowired
    private UserDetailsService jwtUserDetailsService;

    @Autowired
    private JwtRequestFilter jwtRequestFilter;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder aut
        // configure AuthenticationManager so that it knows from
        // user for matching credentials
        // Use BCryptPasswordEncoder
        auth.userDetailsService(jwtUserDetailsService).passwordE
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() thr
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws E
        // We don't need CSRF for this example
        httpSecurity.csrf().disable()
                // dont authenticate this particular request
                .authorizeRequests().antMatchers("/authenticate"
                // all other requests need to be authenticated
                anyRequest().authenticated().and().
                // make sure we use stateless session; session w
                // store user's state.
                exceptionHandling().authenticationEntryPoint(jwt
                .sessionCreationPolicy(SessionCreationPolicy.STA

        // Add a filter to validate the tokens with every reques
        httpSecurity.addFilterBefore(jwtRequestFilter, UsernameP
    }
}
```

Now finally will test this by using below steps:

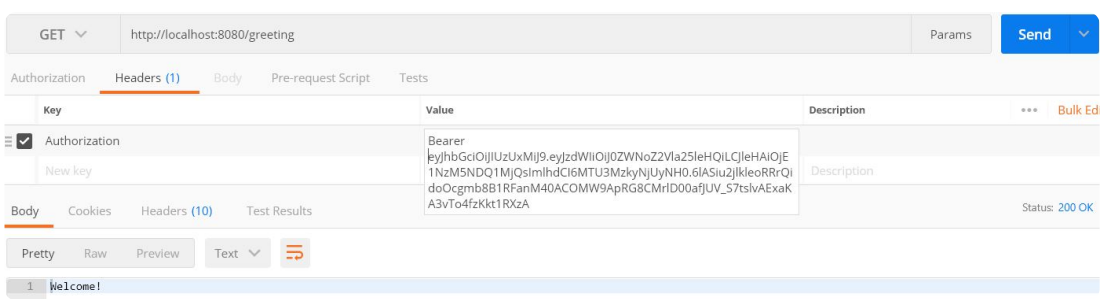
- Generate JSON Web Token (JWT)

  Create POST request (localhost:8080/authenticate) and provide username and password in request body as given below.



- Validate JSON Web Token (JWT)

  Now use GET request localhost:8080/greeting with above generated JWT Token in header request.



Download Source Code

The full source code for this article can be found on below.

Download it here - Spring Boot Security with JWT Token