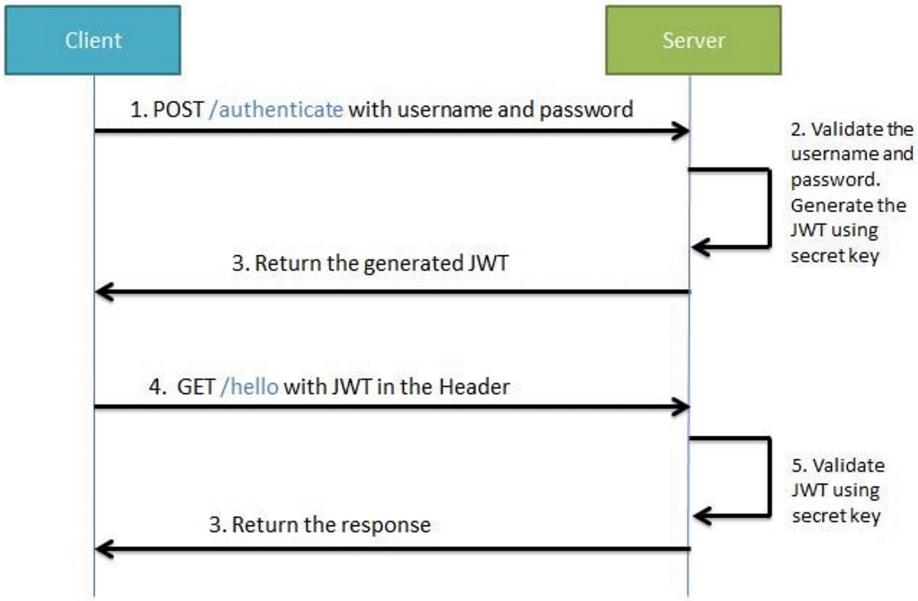
# Springboot Security Jwt

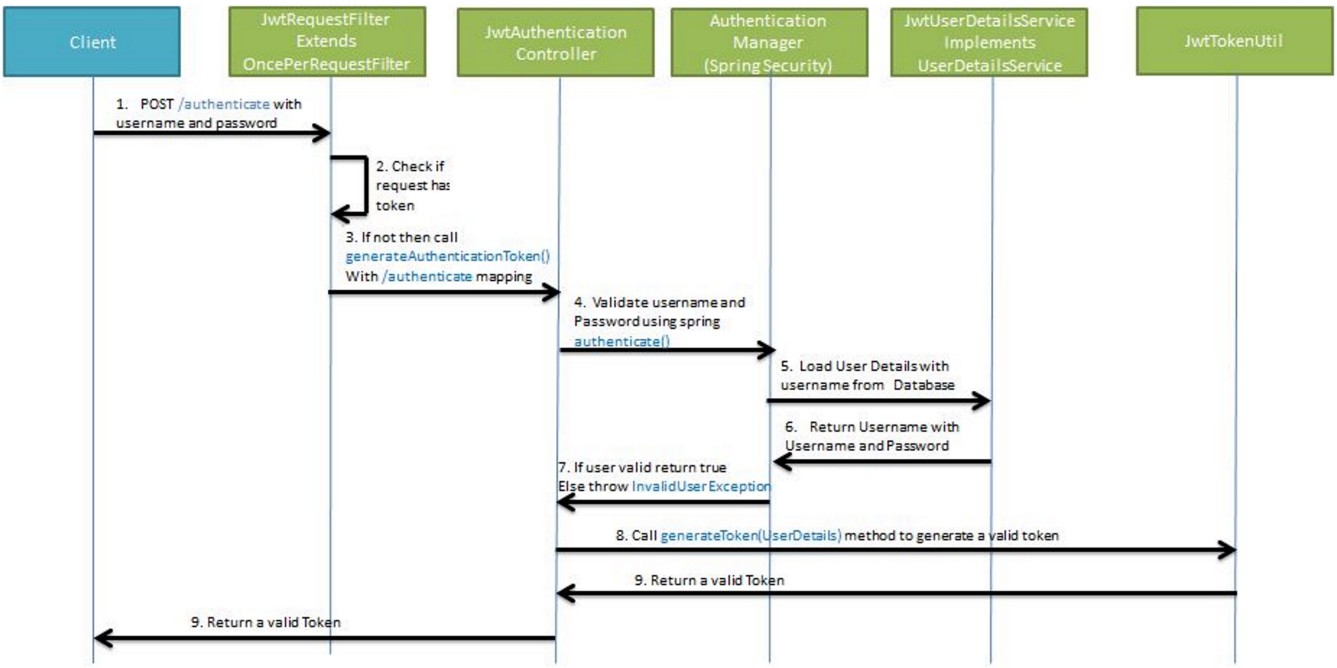
07 January 2022 22:23

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.

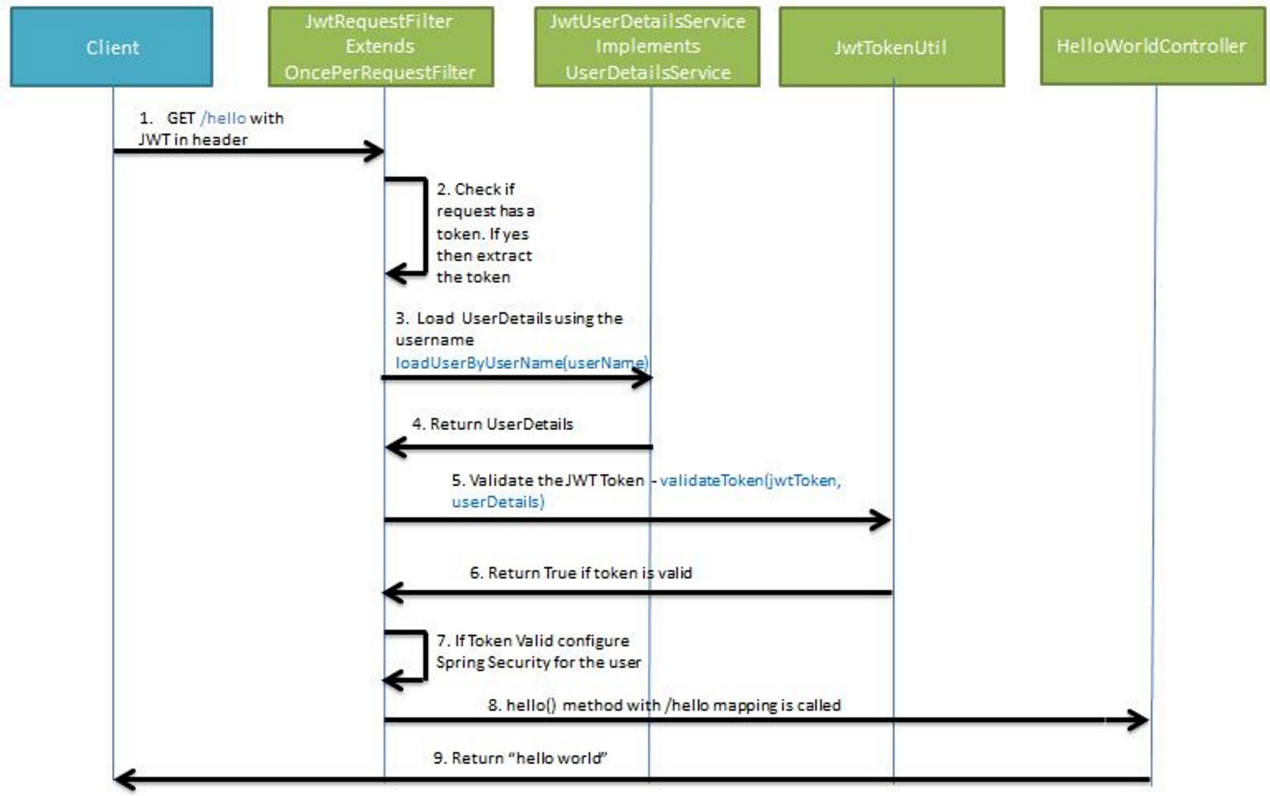
Allow the user to access the api ***/greet*** only if request has a valid token

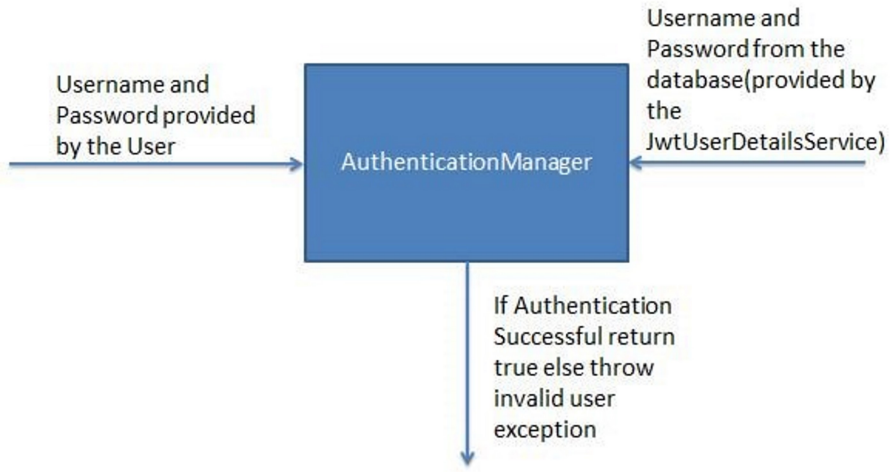



## Generating JWT




## Validating JWT




The **pom.xml** is as follows-

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.2</version>
        <relativePath />
```

```
    </parent>
    <groupId>com.demo</groupId>
    <artifactId>springboot-security-jwt-example</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>p01-springboot-basic</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <!-- API, java.xml.bind module -->
        <dependency>
            <groupId>jakarta.xml.bind</groupId>
            <artifactId>jakarta.xml.bind-api</artifactId>
        </dependency>
        <!-- Runtime, com.sun.xml.bind module -->
        <dependency>
            <groupId>org.glassfish.jaxb</groupId>
            <artifactId>jaxb-runtime</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-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

Create a **DemoController** class for exposing GET REST APIs

```
package com.demo.controllers;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class DemoController {
```

```
    @GetMapping("/greet")
    public String welcome() {
        return "Welcome user";
    }

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello user";
    }

    @GetMapping("/greet/{username}")
    public String welcome(@PathVariable String username) {
        return "Welcome "+username;
    }

}
```

Define a secret key in the **application.properties** file. 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=some@secret#key
```

The **JwtTokenUtil** class uses io.jsonwebtoken.Jwts for performing JWT operations like creation and validation.

```
package com.demo.utils;

import java.io.Serializable;
import java.time.LocalDateTime;
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 TOKEN_VALIDITY = 5 * 60 * 60 * 1000;

    @Value("${jwt.secret}")
    private String secret;


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


    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);
    }


    private Claims getAllClaimsFromToken(String token) {
        return Jwts
                .parser()
                .setSigningKey(secret)
                .parseClaimsJws(token)
                .getBody();
    }

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

   public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();

        claims.put("role", "user");
        claims.put("date", LocalDateTime.now().toString());
        claims.put("message", "some other message");

        return doGenerateToken(claims, userDetails.getUsername());
    }

    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() + TOKEN_VALIDITY))
                    .signWith(SignatureAlgorithm.HS512, secret).compact();
    }

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

**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.

```
package com.demo.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 {

     // find user from database where user = :username
     // userRepo.findByUsername(username);// username, password, roles

        if ("demo".equals(username)) {
            return new User(
                "demo",
//              "{noop}demo@123",
                "$2a$10$.KHH/tBblriAEr8xYyAhSOah0kg.v.XaZoIZOc8wFT5urnwqMCpim",
                new ArrayList<>()
            );
        } else {
            throw new UsernameNotFoundException("User not found with username: " + username);
        }
    }
}
```

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

```
package com.demo.models;

import java.io.Serializable;

public class JwtRequest implements Serializable {
    private static final long serialVersionUID = 5926468583005150707L;

    private String username;
    private String password;

    //JSON Parsing needs default constructor
    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;
    }

    @Override
    public String toString() {
        return "{" +
            " username='" + getUsername() + "'" +
            ", password='" + getPassword() + "'" +
            "}";
    }
```

```
}
```

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

```
package com.demo.models;

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;
    }
}
```

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.demo.controllers;

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.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.demo.models.JwtRequest;
import com.demo.models.JwtResponse;
import com.demo.service.JwtUserDetailsService;
import com.demo.utils.JwtTokenUtil;

@RestController
@CrossOrigin
public class JwtAuthenticationController {
    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Autowired
    private JwtUserDetailsService userDetailsService;

    @PostMapping("/authenticate")
    public ResponseEntity<?> authenticate(@RequestBody JwtRequest req) throws Exception {
        authenticate(req.getUsername(), req.getPassword());
```

```
        final UserDetails userDetails =
                        userDetailsService.loadUserByUsername(req.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);
        }
    }
}
```

The **JwtRequestFilter** class 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.demo.filters;

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.demo.service.JwtUserDetailsService;
import com.demo.utils.JwtTokenUtil;
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;
```

```
        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 {
            System.out.println("JWT Token does not begin with Bearer String");
        }


        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);
        System.out.println("post process");
    }
}
```

## 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.demo.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.demo.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.AuthenticationManagerB
uilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurit
y;
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.web.authentication.UsernamePasswordAuthenticationFilter;
import com.demo.filters.JwtRequestFilter;

@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 {
//      auth.userDetailsService(jwtUserDetailsService);

        auth.userDetailsService(jwtUserDetailsService)
                        .passwordEncoder(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", "/greet").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
        );
    }
}
```

## RestClient

```
POST http://localhost:8080/authenticate
content-type: application/json

{
    "username": "demo",
    "password": "demo@123"
}
###

GET http://localhost:8080/greet
###

GET http://localhost:8080/greet/mark
###

GET http://localhost:8080/greet/mark
Authorization: Bearer valid-token
```

## Use DB

```
<dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
		</dependency>
```

```
spring.datasource.url=jdbc:mysql://localhost/dbname?
createDatabaseIfNotExist=true&autoReconnect=true&useSSL=false
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.platform=mysql
spring.jpa.hibernate.ddl-auto=update
```

```
package com.demo.models;

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;
    }
}
```

```
package com.demo.models

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;
    }
}
```

```
package com.demo.repositories;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import com.javainuse.model.DAOUser;
public interface UserDao extends CrudRepository<DAOUser, Integer> {

    UserDao findByUsername(String username);

}
```

```
public class JwtUserDetailsService implements UserDetailsService {
    public UserDao save(UserDTO user) {
                DAOUser newUser = new DAOUser();
                newUser.setUsername(user.getUsername());
                newUser.setPassword(bcryptEncoder.encode(user.getPassword()));
                return userDao.save(newUser);
            }
}
```

```
@RestController
@CrossOrigin
public class JwtAuthenticationController {

@RequestMapping(value = "/register", method = RequestMethod.POST)
        public ResponseEntity<?> saveUser(@RequestBody UserDTO user) throws Exception {
                return ResponseEntity.ok(userDetailsService.save(user));
        }
}
```

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

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

}
```

```
@Service
public class JwtUserDetailsService implements UserDetailsService {

	@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<>());
			}
}
```