==========

How to secure REST APIs using Spring Boot

==========

-> Security is very important for every web application

-> To protect our application & application data we need to implement security logic

-> Spring Security concept we can use to secure our web applications / REST APIs

-> To secure our spring boot application we need to add below starter in pom.xml file

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

Note: When we add this dependency in pom.xml file then by default our application will be secured with basic authentication. It will generate random password to access our application.

Note: Generated Random Password will be printed on console.

-> We need to use below credentials to access our application

Username : user

Password : <copy the pwd from console>

-> When we access our application url in browser then it will display "Login Form" to authenticate our request.

-> To access secured REST API from postman, we need to set Auth values in POSTMAN to send the request

Auth : Basic Auth

Username : user

Password : <copy-from-console>

==========

How to override Spring Security Default Credentials

==========

-> To override Default credentials we can configre security credentials in application.properties file or application.yml file like below

spring.security.user.name=ashokit

spring.security.user.password=ashokit@123

-> After configuring credentials like above, we need to give above credentials to access our application / api.

==========

How to secure specific URL Patterns

==========

-> When we add 'security-starter' in pom.xml then it will apply security filter for all the HTTP methods of our application.

-> But in reality we need to secure only few methods not all methods in our application.

For Example

/ login-page --> security not required

/ transfer ---> security required

/ balance ---> security required

/about-us ---> security not required

-> In order to achieve above requirement we need to Customize Security Configuration in our project like below

```
@Configuration
@EnableWebSecurity
public class SecurityConfigurer {

	@Bean
	public SecurityFilterChain securityFilter(HttpSecurity http) throws Exception{
		// logic tp customized security in our app

		http.authorizeHttpRequests(request ->
request.requestMatchers("/balance","/contact","/swagger-ui.html").permitAll()

.anyRequest().authenticated()
)
			.sessionManagement(sess ->
sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
			.httpBasic(Customizer.withDefaults());
		return http.build();

	}

}
```

==========

Spring Boot Security with JDBC Authentication

==========

Step-1) Setup Database tables with required data

```
-- users table structure

CREATE TABLE `users` (
  `username` VARCHAR(50) NOT NULL,
  `password` VARCHAR(120) NOT NULL,
  `enabled` TINYINT(1) NOT NULL,
 PRIMARY KEY (`username`)
);



-- authorities table structure

CREATE TABLE `authorities` (
  `username` VARCHAR(50) NOT NULL,
  `authority` VARCHAR(50) NOT NULL,
 KEY `username` (`username`),
 CONSTRAINT `authorities_ibfk_1` FOREIGN KEY (`username`)
 REFERENCES `users` (`username`)
);



========== Online Encrypt : https://bcrypt-generator.com/
==============================

-- insert records into table

insert into users values ('admin',
'$2a$12$e9oIZjBeSJDryJ/P5p1Ep.WPzJ3f4.C2vHC/as1E22R25XXGpPYyG',  1);
```

```
insert into users values ('user',
'$2a$12$JQiGAJhdSOoTXAzIpbDxpemXcYHCmxYOnodLNBeNORH8J4FLxHGvK',  1);


insert into authorities values ('admin', 'ROLE_ADMIN');

insert into authorities values ('admin', 'ROLE_USER');

insert into authorities values ('user', 'ROLE_USER');
```

Step-2) Create Boot application with below dependencies

a) web-starter

b) security-starter

c) data-jdbc

d) mysql-connector

e) lombok

f) devtools

Step-3) Configure Data source properties in application.yml file

```
spring:
 application:
  name: 45-Spring-Security-JDBC-Authentication-App
 datasource:
  driver-class-name: com.mysql.cj.jdbc.Driver
  password: password
  url: jdbc:mysql://localhost:3306/sbms
  username: root
```

```
 jpa:
   show-sql: true
```

Step-4) Create Rest Controller with Required methods

```
@RestController
public class UserRestController {

	@GetMapping(value = "/admin")
	public String admin() {
		return "<h3>Welcome Admin :)</h3>";
	}

	@GetMapping(value = "/user")
	public String user() {
		return "<h3>Hello User :)</h3>";
	}

	@GetMapping(value = "/")
	public String welcome() {
		return "<h3>Welcome :)</h3>";
	}

}
```

Step-5) Create Security Configuration class like below with Jdbc Authentication Manager

```
package in.ashokit;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBu
ilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfigurer {

	@Autowired
	private DataSource dataSource;

	@Autowired
	public void authManager(AuthenticationManagerBuilder auth) throws Exception{
		auth.jdbcAuthentication()
			.dataSource(dataSource)
```

```
                .passwordEncoder(new BCryptPasswordEncoder())
                .usersByUsernameQuery("select username,password,enabled from users
where username=?")
                .authoritiesByUsernameQuery("select username,authority from authorities
where username=?");
        }

        @Bean
        public SecurityFilterChain securityConfig(HttpSecurity http) throws Exception {

                http.authorizeHttpRequests(request ->
request.requestMatchers("/admin").hasRole("ADMIN")

.requestMatchers("/user").hasAnyRole("ADMIN","USER")

.requestMatchers("/").permitAll()

.anyRequest().authenticated()
                                        )
                                        .sessionManagement(sess-
>sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                                        .httpBasic(Customizer.withDefaults());

                return http.build();
        }



}
```

==========

OAuth 2.0 (https://oauth.net/2/)

==========

1) Create Spring Boot application with below dependencies

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

2) Create OAuth app in Github.com

	(Login --> Settings --> Developer Settings --> OAuth Apps --> Create App --> Copy Client ID & Client Secret)

3) Configure GitHub OAuth App client id & client secret in application.yml file like below

```
spring:
  security:
    oauth2:
      client:
        registration:
          github:
            clientId: Ov23liovJyJ25Oq5xgXc
            clientSecret: e37750b18085284ed102b9d7ce668caaed30cfa3
```

4) Create Rest Controller with method

```
@RestController
public class WelcomeRestController {

	@GetMapping("/")
	public String welcome() {
		return "Welcome to Ashok IT";
	}
}
```

5) Run the application and test it.

==========

Spring Boot with JWT

==========

-> JWT stands for JSON Web Tokens

-> JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.

-> JWT official Website : https://jwt.io/

-> Below is the sample JWT Token

token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ

.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

-> JWT contains below 3 parts

1) Header
2) Payload
3) Signature

Note: JWT 3 parts will be seperated by using dot(.)

JWT Authentication

* JWT stands for JSON Web Token
* JWT mostly used for securing REST apis
* Best way to communicate security between client and server securely
* JWT follows a stateless authentication mechanism (IMPORTANT)

JWT Architecture :

1) Header -> Algo+Type

2) Payload -> Information about claims

3) Signature -> encoded header + encoded payload + key

8 Steps for JWT authentication:

1) Add dependency(io.jsonwebtoken)

2) Create JWT AuthenticationEntryPoint implements AuthenticationEntryPoint

3) Create JWTTokenHelper

4) JWtAuthenticationFilter extends OncePerRequestFilter

    1. Get JWT token from request

    2. Validate

    3. Get User from token

    4. Load user associated with token

    5. Set spring security ---------> Set Authentication

5) Create JWTAuthResponse

6) Configure JWT in spring security config

7) Create Login api to return token

8) Test the Application

==========

1) Create Spring Boot appliation with below dependencies

==========

```
<dependencies>
    <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>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
```

```
</dependency>

        <!-- Spring Boot Starter Security -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<!-- Spring Boot Starter Validation -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-validation</artifactId>
</dependency>

<!-- JWT Dependencies -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.5</version>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.11.5</version>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.11.5</version>
</dependency>
```

```
        <!-- MYSQL Connector -->
            <!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
            <dependency>
              <groupId>com.mysql</groupId>
              <artifactId>mysql-connector-j</artifactId>
              <!--<version>9.0.0</version>-->
            </dependency>
        </dependencies>
```

==========

2) Create Request and Response Binding Classes

==========

```
public class AuthenticationResponse implements Serializable {

  private final String jwt;

  public AuthenticationResponse(String jwt) {
    this.jwt = jwt;
  }

  public String getJwt() {
    return jwt;
  }
}


import lombok.AllArgsConstructor;
import lombok.Data;
```

```
import lombok.NoArgsConstructor;


@Data
@NoArgsConstructor
@AllArgsConstructor
public class AuthRequest {
	private String username;
	private String password;
}


package in.api.entities;


import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;



@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class UserInfo {

	@Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private int id;
    private String name;
    private String email;
    private String password;
    private String roles;

}
```

==========

3) Create UserInfoRepository interface to established connection to Database(MySQL)

==========

```
package in.api.repositories;

import java.util.Optional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import in.api.entities.UserInfo;

@Repository
public interface UserInfoRepository extends JpaRepository<UserInfo, Integer>{
        Optional<UserInfo> findByName(String username);
}
```

==========

4) Implement UserDetailsService interface to loadUserByUserName (Its Service for Our RestController)

==========

```
package in.api.services;
```

```
import java.util.Optional;

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 in.api.entities.UserInfo;
import in.api.repositories.UserInfoRepository;

@Service
public class UserInfoService implements UserDetailsService{

	private final UserInfoRepository repository;
  private final PasswordEncoder encoder;

  public UserInfoService(UserInfoRepository repository, PasswordEncoder encoder) {
    this.repository = repository;
    this.encoder = encoder;
  }

	@Override
	public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
		Optional<UserInfo> userDetail = repository.findByName(username);
		 // Converting userDetail to UserDetails
```

```
                return userDetail.map(UserInfoDetails::new).orElseThrow(() -> new
UsernameNotFoundException("User not found " + username));
        }

        public String addUser(UserInfo userInfo) {
                userInfo.setPassword(encoder.encode(userInfo.getPassword()));
                repository.save(userInfo);
                return "User Added Successfully";
        }


}
```

==========

5) Implement UserDetails interface to get Core User Info such to get password, Authorities, credentials expired or not (Its also a service)

==========

```
package in.api.services;

import java.util.Arrays;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
```

```
import in.api.entities.UserInfo;

public class UserInfoDetails implements UserDetails{

        private String name;
  private String password;
  private List<GrantedAuthority> authorities;

  public UserInfoDetails(UserInfo userInfo) {
    name = userInfo.getName();
    password = userInfo.getPassword();
    authorities = Arrays.stream(userInfo.getRoles().split(","))
        .map(SimpleGrantedAuthority::new)
        .collect(Collectors.toList());
  }

  @Override
  public Collection<? extends GrantedAuthority> getAuthorities() {
    return authorities;
  }

  @Override
  public String getPassword() {
    return password;
  }

  @Override
  public String getUsername() {
```

```
    return name;
  }
  @Override
  public boolean isAccountNonExpired() {
    return true;
  }
  @Override
  public boolean isAccountNonLocked() {
    return true;
  }
  @Override
  public boolean isCredentialsNonExpired() {
    return true;
  }
  @Override
  public boolean isEnabled() {
    return true;
  }

}
```

==========

6) Create JwtService class for generation of JWT Token, validation of token etc (Its Service for JWT tokens)

==========

```
package in.api.services;

import java.security.Key;
```

```
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

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

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;

@Component
public class JwtService {

	public static final String SECRET = "9D0EB6B1C2E1FAD0F53A248F6C3B5E4E2F6D8G3H1I0J7K4L1M9N2O3P5Q0R7S9T1U4V2W6X0Y3Z";

	public String generateToken(String userName) {
    Map<String, Object> claims = new HashMap<>();
    return createToken(claims, userName);
  }

	private String createToken(Map<String, Object> claims, String userName) {
    return Jwts.builder()
	.setClaims(claims)
```

```
        .setSubject(userName)
        .setIssuedAt(new Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 30))
        .signWith(getSignKey(), SignatureAlgorithm.HS256).compact();
}

        private Key getSignKey() {
   byte[] keyBytes= Decoders.BASE64.decode(SECRET);
   return Keys.hmacShaKeyFor(keyBytes);
}

        public String extractUsername(String token) {
   return extractClaim(token, Claims::getSubject);
}
public Date extractExpiration(String token) {
   return extractClaim(token, Claims::getExpiration);
}
public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
   final Claims claims = extractAllClaims(token);
   return claimsResolver.apply(claims);
}

private Claims extractAllClaims(String token) {
   return Jwts
        .parserBuilder()
        .setSigningKey(getSignKey())
        .build()
        .parseClaimsJws(token)
        .getBody();
```

```
    }

    private Boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }
    public Boolean validateToken(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }

}
```

==========

7) create JWTFilter Class to do internal pre-processing of Http Request if they have valid token & credentials

==========

```
package in.api.filter;

import java.io.IOException;

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 in.api.services.JwtService;
import in.api.services.UserInfoService;
```

```
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@Component
public class JwtAuthFilter extends OncePerRequestFilter{

        private final JwtService jwtService;
  private final UserInfoService userDetailsService;

  JwtAuthFilter(JwtService jwtService, UserInfoService userDetailsService) {
    this.jwtService = jwtService;
    this.userDetailsService = userDetailsService;
  }

  @Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain) throws ServletException, IOException {

        String authHeader = request.getHeader("Authorization");
    String token = null;
    String username = null;
    if (authHeader != null && authHeader.startsWith("Bearer ")) {
      token = authHeader.substring(7);
      username = jwtService.extractUsername(token);
    }
    if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
```

```
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        if (jwtService.validateToken(token, userDetails)) {
          UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
          authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
          SecurityContextHolder.getContext().setAuthentication(authToken);
        }
      }

      filterChain.doFilter(request, response);
    }
}
```

==========

8) Create Config class to configure which endpoints/url will be authenticated and which not ,

To Define AuthenticationProvider, AuthenticationManager and FilterChain

==========

```
package in.api.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.Customizer;
```

```
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfig
uration;

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

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

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

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

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

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

import org.springframework.security.web.SecurityFilterChain;

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


import in.api.filter.JwtAuthFilter;

import in.api.repositories.UserInfoRepository;

import in.api.services.UserInfoService;


@Configuration

@EnableWebSecurity

@EnableMethodSecurity

public class SecurityConfig {


        private final JwtAuthFilter authFilter;


        public SecurityConfig(JwtAuthFilter authFilter) {

            this.authFilter = authFilter;

        }


        @Bean
```

```
    public UserDetailsService userDetailsService(UserInfoRepository repository,
PasswordEncoder passwordEncoder) {
        return new UserInfoService(repository, passwordEncoder);
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http, AuthenticationProvider
authenticationProvider) throws Exception {

        return http
            .authorizeHttpRequests((authz) -> authz
              .requestMatchers("/generateToken", "/register","/h2-console").permitAll()
              .requestMatchers("/hello").authenticated()
            )
            .httpBasic(Customizer.withDefaults())
            .csrf((csrf) -> csrf.disable())
            .sessionManagement((session) -
>session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authenticationProvider(authenticationProvider)
            .addFilterBefore(authFilter, UsernamePasswordAuthenticationFilter.class)
            .build();
    }

    @Bean
    public AuthenticationProvider authenticationProvider(UserDetailsService
userDetailsService, PasswordEncoder passwordEncoder) {
        DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDetailsService); // Gives service
object to retrive data of user
```

```
                authenticationProvider.setPasswordEncoder(passwordEncoder);      // Gives object to find password encoding details
                return authenticationProvider;
        }

        @Bean
        public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {
           return config.getAuthenticationManager();
        }

}
```

==========

9) Create PasswordEncoderConfig class to define which Password Configuration (Here is BCryptPasswordEncoder)

==========

```
package in.api.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class PasswordEncoderConfig {

        @Bean
        public PasswordEncoder passwordEncoder() {
```

```
    return new BCryptPasswordEncoder();
  }

}
```

==========

10) Create UserController which defines REST API's endpoints

==========

```
package in.api.controllers;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import in.api.entities.AuthRequest;
import in.api.entities.UserInfo;
import in.api.services.JwtService;
import in.api.services.UserInfoService;

@RestController
public class UserController {
```

```
        private final UserInfoService service;

    private final JwtService jwtService;

    private final AuthenticationManager authenticationManager;


    UserController(UserInfoService service, JwtService jwtService, AuthenticationManager
authenticationManager) {

        this.service = service;

        this.jwtService = jwtService;

        this.authenticationManager = authenticationManager;

    }


    @PostMapping("/register")

    public ResponseEntity<String> addNewUser(@RequestBody UserInfo userInfo) {

        String response = service.addUser(userInfo);

        return ResponseEntity.status(HttpStatus.CREATED).body(response);

    }


    @PostMapping("/generateToken")

    public ResponseEntity<String> authenticateAndGetToken(@RequestBody AuthRequest
authRequest) {

        Authentication authentication = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(authRequest.getUsername(),
authRequest.getPassword()));

        if (authentication.isAuthenticated()) {

            String token = jwtService.generateToken(authRequest.getUsername());

            return ResponseEntity.ok(token);

        } else {

            throw new UsernameNotFoundException("Invalid user request!");

        }

    }
```

```
    @GetMapping("/hello")
    public String hello() {
        return "Hello World!";
    }

}
```

==========
==========