```
===========================================
How to secure REST APIs using Spring Boot
============================================


1) Authentication (verifying credentials)

2) Authorization (can this user access specific functionality)


-> 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

                                / contact-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{

                http.authorizeHttpRequests((request) -> request
                                .antMatchers("/","/login","/about", "/swagger-ui.html").permitAll()
                                .anyRequest().authenticated()
                ).formLogin();

                return http.build();
        }
}


==========================================
Spring Security In-Memory Authentication
==========================================

-> In Memory Authentication means storing user credentials in the program for Authentication Purpose.

-> This is not recommended for production.

@Bean
public InMemoryUserDetailsManager configureUsers() {

           UserDetails adminUser = User.withDefaultPasswordEncoder()
                                                                        .username("ashok")
                                                                        .password("ashok@123")
                                                                        .authorities("ADMIN")
                                                                        .build();

                UserDetails normalUser = User.withDefaultPasswordEncoder()
                                                                        .username("raja")
                                                                        .password("raja@123")
                                                                        .authorities("USER")
                                                                        .build();

                return new InMemoryUserDetailsManager(adminUser, normalUser);
}
```

```
==============================================
Spring Boot Security with JDBC Authentication
==============================================

=> JDBC Authentication is used to fetch Db table data for User authentication purpose


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 ('john', '$2a$12$Lj2F05RypfMa2i5rzjGdg.gLTEYfhVUUMRB0QVktv0Q6JQgPS6gHe',
1);

insert into users values ('smith', '$2a$12$68VILR5JATg0zWXYK/l5hu8FIrilVjIV0MvAt9A1tqrtzGgNutTT.',
1);

insert into authorities values ('john', 'ROLE_ADMIN');
insert into authorities values ('john', 'ROLE_USER');
insert into authorities values ('smith', '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:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: AshokIT@123
    url: jdbc:mysql://localhost:3306/sbms33
    username: ashokit
  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.AuthenticationManagerBuilder;
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 SecurityConfiguration {

        private static final String ADMIN = "ADMIN";
        private static final String USER = "USER";

        @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( (req) -> req
                                .antMatchers("/admin").hasRole(ADMIN)
                                .antMatchers("/user").hasAnyRole(ADMIN,USER)
                                .antMatchers("/").permitAll()
                                .anyRequest().authenticated()
                ).formLogin();

                return http.build();
        }

}
```

===
How to work with UserDetailsService in Spring Security
===

=> UserDetailsService is a predefined interface which contains loadUserByUsername(String name) method.

=> This is used to load User record for Authentication purpose in Spring Security.

=> We can implement UserDetailsService interface and we can write the logic to retrieve User record based on given username for Authentication purpose.

=> If we give UserDetailsService object to AuthenticationManagerBuild then AuthManager will call this method for every login request.

```
---------------------------------------------------------------------------------------------------
@Service
public class MyUserDetailsService implements UserDetailsService{

        @Override
        public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

                System.out.println("loadUserByUsername() method called.....");

                // logic to get user data
                // userRepo.findByUsername(name);

                return new User("ashok", "ashok@123", Collections.emptyList());
        }
}
-------------------------------------------------------------------------------
@Configuration
@EnableWebSecurity
public class SecurityConfigurer {

        @Autowired
        private MyUserDetailsService userDtlsService;

        @Autowired
        public void configureUsers(AuthenticationManagerBuilder auth) throws Exception{
                auth.userDetailsService(userDtlsService)
                        .passwordEncoder(NoOpPasswordEncoder.getInstance());
        }
        @Bean
        public SecurityFilterChain securityConfig(HttpSecurity http) throws Exception {
                http.authorizeHttpRequests( (req) ->
                                req.antMatchers("/contact")
                                   .permitAll()
                                   .anyRequest()
                                   .authenticated()
                ).formLogin();
                return http.build();
        }
}
```

```
##############
OAuth 2.0
##############
```

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:
            clientSecret:
```

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.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ij
oxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

-> JWT contains below 3 parts

                1) Header
                2) Payload
                3) Signature

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



=========================================================
1) Create Spring Boot appliation with below dependencies
=========================================================
        <dependencies>
                <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>

                <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-tomcat</artifactId>
                        <scope>provided</scope>
                </dependency>

                <dependency>
                        <groupId>io.jsonwebtoken</groupId>
                        <artifactId>jjwt</artifactId>
                        <version>0.9.1</version>
```

```
		</dependency>
		<dependency>
			<groupId>javax.xml.bind</groupId>
			<artifactId>jaxb-api</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>
		<dependency>
			<groupId>org.springframework.security</groupId>
			<artifactId>spring-security-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>
```

```
===================================================
2) Create Request and Response Binding Classes
===================================================
@Data

public class AuthenticationRequest implements Serializable {

	private String username;
	private String password;
}

public class AuthenticationResponse implements Serializable {

    private final String jwt;

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

    public String getJwt() {
        return jwt;
    }
}

=============================================================
3) Create UserDetailsService for credentials configuration
=============================================================

package com.ashokit.security;

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;

import java.util.ArrayList;

@Service
public class MyUserDetailsService implements UserDetailsService {
        @Override
        public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException {
                return new User("admin",
"$2a$12$e9oIZjBeSJDryJ/P5p1Ep.WPzJ3f4.C2vHC/as1E22R25XXGpPYyG", new ArrayList<>());
        }
}

===================================================
4) Create JwtUtils class
===================================================
@Service
public class JwtUtil {

    private String SECRET_KEY = "secret";

    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.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).getBody();
    }

    private Boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

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

    private String createToken(Map<String, Object> claims, String subject) {

        return Jwts.builder()
                          .setClaims(claims)
                          .setSubject(subject)
                          .setIssuedAt(new Date(System.currentTimeMillis()))
                          .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 10))
                          .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                          .compact();
    }

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


===================================================
5) Create Filter class
===================================================


@Component
public class JwtRequestFilter extends OncePerRequestFilter {
    @Autowired
    private MyUserDetailsService userDetailsService;

    @Autowired
    private JwtUtil jwtUtil;

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

        final String authorizationHeader = request.getHeader("Authorization");

        String username = null;
        String jwt = null;

        if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
            jwt = authorizationHeader.substring(7);
            username = jwtUtil.extractUsername(jwt);
        }


        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

            UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

            if (jwtUtil.validateToken(jwt, userDetails)) {

                UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new
UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());
                usernamePasswordAuthenticationToken
                        .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
            }
        }
        chain.doFilter(request, response);
    }

}

====================================
6) Create WebSecurity Config class
====================================

package com.ashokit.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import com.ashokit.filters.JwtRequestFilter;
```

```
@Configuaration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

        @Autowired
        private UserDetailsService myUserDetailsService;

        @Autowired
        private JwtRequestFilter jwtRequestFilter;

        @Autowired
        public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
                auth.userDetailsService(myUserDetailsService);
        }

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

        @Override
        @Bean
        public AuthenticationManager authenticationManagerBean() throws Exception {
                return super.authenticationManagerBean();
        }

        @Override
        protected void configure(HttpSecurity httpSecurity) throws Exception {
                httpSecurity.csrf()
                                        .disable()
                                        .authorizeRequests()
                                        .antMatchers("/authenticate")
                                        .permitAll()
                                        .anyRequest()
                                        .authenticated()
                                        .and()
                                        .exceptionHandling()
                                        .and()
                                        .sessionManagement()
                                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

                httpSecurity.addFilterBefore(jwtRequestFilter,
UsernamePasswordAuthenticationFilter.class);
        }
}

==================================
7) create Rest Controller class
===================================

package com.ashokit.rest;

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.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
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.ashokit.models.AuthenticationRequest;
import com.ashokit.models.AuthenticationResponse;
import com.ashokit.security.MyUserDetailsService;
import com.ashokit.util.JwtUtil;

@RestController
public class HelloRestController {

        @Autowired
        private AuthenticationManager authenticationManager;

        @Autowired
        private JwtUtil jwtTokenUtil;

        @Autowired
        private MyUserDetailsService userDetailsService;

        @RequestMapping({ "/hello" })
        public String firstPage() {
                return "Hello World";
        }

        @RequestMapping(value = "/authenticate", method = RequestMethod.POST)
        public ResponseEntity<?> createAuthenticationToken(@RequestBody AuthenticationRequest 
authenticationRequest)
                        throws Exception {

                try {
                        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(
                                        authenticationRequest.getUsername(), 
authenticationRequest.getPassword()));
                } catch (BadCredentialsException e) {
                        throw new Exception("Incorrect username or password", e);
                }

                final UserDetails userDetails = 
userDetailsService.loadUserByUsername(authenticationRequest.getUsername());

                final String jwt = jwtTokenUtil.generateToken(userDetails);

                return ResponseEntity.ok(new AuthenticationResponse(jwt));
        }
}



==================================
8) Run the application and Test it
===================================
```