**Spring Boot Security**

1. Add spring boot starter security dependency in the pom.xml file to provide basic authentication which default username is ‘user’ and password is provided in the console by spring boot.

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

1. If we want our custom username and password for security then we have to use two properties in application.properties or application.yml file. Below are the example of application.properties

*spring.security.user.name=Brajendra*

[*spring.security.user.password=Brajendra@23*](mailto:spring.security.user.password=Brajendra@23)

1. If we want use our custom username and password for security using configuration class then we have configure one class using @Configuration annotation and create one method which return UserDetailsService.

3.1> Create UserDetails object with the help of User class.

3.2>Create InMemoryUserDetailsManager class object with passing UserDetails object and return this object. Below are the example.

**import** org.springframework.security.core.userdetails.User;  
**import** org.springframework.security.core.userdetails.UserDetails;  
**import** org.springframework.security.core.userdetails.UserDetailsService;

@Bean  
**public** UserDetailsService userDetailsService() {

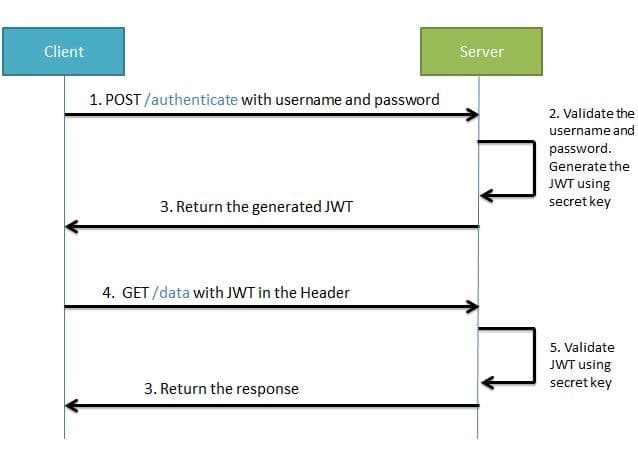
UserDetails userDetails1 = User.*builder*().username(**"Brajendra"**).password(passwordEncoder().encode(**"Brajendra@23"**)).roles(**"ADMIN"**).build();

UserDetails userDetails2 = User.*builder*().username(**"Shikha"**).password(passwordEncoder().encode(**"Shikha@05"**)).roles(**"ADMIN"**).build();

UserDetails userDetails3 = User.*builder*().username(**"Diksha"**).password(passwordEncoder().encode(**"Diksha@07"**)).roles(**"ADMIN"**).build();

**return new** InMemoryUserDetailsManager(userDetails1,userDetails2,userDetails3);  
}

**JWT Authentication Flow**



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

**JWT Encoded Token** **–**

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

**JWT Decoded Value –**

1. HEADER:ALGORITHM & TOKEN TYPE

{

"alg": "HS256",

"typ": "JWT"

}

1. PAYLOAD:DATA

{

"sub": "1234567890",

"name": "John Doe",

"iat": 1516239022

}

1. VERIFY SIGNATURE

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

your-256-bit-secret

)

**Steps to implement JWT Token :-**

1. Make sure spring-boot-starter-security is there in pom.xml
2. Create Class **JWTAthenticationEntryPoint** that

implement **AuthenticationEntryPoint**. Method of this class is called whenever as exception is thrown due to unauthenticated user trying to access the resource that required authentication.

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

@Override

public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException, ServletException {

response.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);

PrintWriter writer = response.getWriter();

writer.println("Access Denied !! " + authException.getMessage());

}

}

1. Create **JWTHelper**  class This class contains method related to perform operations with jwt token like generateToken, validateToken etc.

@Component

public class JwtHelper {

//requirement :

public static final long JWT\_TOKEN\_VALIDITY = 5 \* 60 \* 60;

// public static final long JWT\_TOKEN\_VALIDITY = 60;

private String secret = "afafasfafafasfasfasfafacasdasfasxASFACASDFACASDFASFASFDAFASFASDAADSCSDFADCVSGCFVADXCcadwavfsfarvf";

//retrieve username from jwt token

public String getUsernameFromToken(String token) {

return getClaimFromToken(token, Claims::getSubject);

}

//retrieve expiration date from jwt token

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);

}

//for retrieveing any information from token we will need the secret key

private Claims getAllClaimsFromToken(String token) {

return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();

}

//check if the token has expired

private Boolean isTokenExpired(String token) {

final Date expiration = getExpirationDateFromToken(token);

return expiration.before(new Date());

}

//generate token for user

public String generateToken(UserDetails userDetails) {

Map<String, Object> claims = new HashMap<>();

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

}

//while creating the token -

//1. Define claims of the token, like Issuer, Expiration, Subject, and the ID

//2. Sign the JWT using the HS512 algorithm and secret key.

//3. According to JWS Compact Serialization(https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41#section-3.1)

// compaction of the JWT to a URL-safe string

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() + JWT\_TOKEN\_VALIDITY \* 1000))

.signWith(SignatureAlgorithm.HS512, secret).compact();

}

//validate token

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = getUsernameFromToken(token);

return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

}

1. Create **JWTAuthenticationFilter** that extends **OncePerRequestFilter** and override method and write the logic to check the token that is comming in header. We have to write 5 important logic
   1. Get Token from request
   2. Validate Token
   3. GetUsername from token
   4. Load user associated with this token
   5. set authentication

@Component

public class JwtAuthenticationFilter extends OncePerRequestFilter {

private Logger logger = LoggerFactory.getLogger(OncePerRequestFilter.class);

@Autowired

private JwtHelper jwtHelper;

@Autowired

private UserDetailsService userDetailsService;

@Override

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

// try {

// Thread.sleep(500);

// } catch (InterruptedException e) {

// throw new RuntimeException(e);

// }

//Authorization

String requestHeader = request.getHeader("Authorization");

//Bearer 2352345235sdfrsfgsdfsdf

logger.info(" Header : {}", requestHeader);

String username = null;

String token = null;

if (requestHeader != null && requestHeader.startsWith("Bearer")) {

//looking good

token = requestHeader.substring(7);

try {

username = this.jwtHelper.getUsernameFromToken(token);

} catch (IllegalArgumentException e) {

logger.info("Illegal Argument while fetching the username !!");

e.printStackTrace();

} catch (ExpiredJwtException e) {

logger.info("Given jwt token is expired !!");

e.printStackTrace();

} catch (MalformedJwtException e) {

logger.info("Some changed has done in token !! Invalid Token");

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

} else {

logger.info("Invalid Header Value !! ");

}

//

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

//fetch user detail from username

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

Boolean validateToken = this.jwtHelper.validateToken(token, userDetails);

if (validateToken) {

//set the authentication

UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);

} else {

logger.info("Validation fails !!");

}

}

filterChain.doFilter(request, response);

}

}

1. Configure spring security in configuration file:
2. package com.jwt.example.JWTSpringBoot3.config;  
     
   import com.jwt.example.JWTSpringBoot3.security.JWTAuthenticationEntryPoint;  
   import com.jwt.example.JWTSpringBoot3.security.JWTAuthenticationFilter;  
   import com.jwt.example.JWTSpringBoot3.service.CustomUserDetailService;  
   import jakarta.servlet.http.HttpServletRequest;  
   import org.springframework.beans.factory.annotation.Autowired;  
   import org.springframework.context.annotation.Bean;  
   import org.springframework.context.annotation.Configuration;  
   import org.springframework.core.annotation.Order;  
   import org.springframework.security.authentication.dao.DaoAuthenticationProvider;  
   import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
   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;  
     
   @Configuration  
   public class SecurityConfig {  
     
     
    @Autowired  
    private PasswordEncoder passwordEncoder;  
     
    @Autowired  
    private UserDetailsService userDetailsService;  
     
    @Autowired  
    private CustomUserDetailService customUserDetailService;  
     
    @Autowired  
    private JWTAuthenticationEntryPoint jwtAuthenticationEntryPoint;  
     
    @Autowired  
    private JWTAuthenticationFilter jwtAuthenticationFilter;  
     
      
      
    @Bean  
    public DaoAuthenticationProvider daoAuthenticationProvider(){  
    DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();  
    authenticationProvider.setUserDetailsService(customUserDetailService);  
    authenticationProvider.setPasswordEncoder(passwordEncoder);  
    return authenticationProvider;  
    }  
     
     
      
      
    @Bean  
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
     
    http.csrf(csrf -> csrf.disable())  
    .cors(cors -> cors.disable())  
    .authorizeHttpRequests(auth -> auth.requestMatchers("/home/\*\*").authenticated()  
    .requestMatchers("/auth/login").permitAll()  
    .requestMatchers("/auth/create-user").permitAll().anyRequest().authenticated())  
    .exceptionHandling(ex -> ex.authenticationEntryPoint(jwtAuthenticationEntryPoint))  
    .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.*STATELESS*));  
    http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);  
     
    return http.build();  
    }  
     
   }
3. Create JWTRequest and JWTResponse to receive request data and send Login success response.

public class JWTRequest {  
  
 private String email;  
  
 private String password;  
  
}

public class JWTResponse {  
  
 private String jwtToken;  
  
 private String username;  
  
}

1. Create login api to accept username and password and return token if username and password is correct.

@PostMapping("/login")  
public ResponseEntity<JWTResponse> login(@RequestBody JWTRequest request) {  
  
 this.doAuthentication(request.getEmail(),request.getPassword());  
 UserDetails userDetails = customUserDetailService.loadUserByUsername(request.getEmail());  
 String token = this.helper.generateToken(userDetails);  
 JWTResponse response = JWTResponse.*builder*()  
 .jwtToken(token)  
 .username(userDetails.getUsername())  
 .build();  
 return new ResponseEntity<>(response, HttpStatus.*OK*);  
}  
  
private void doAuthentication(String email, String password) {  
  
 UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(email,password);  
 try {  
 manager.authenticate(authentication);  
 }catch (BadCredentialsException be) {  
 throw new BadCredentialsException("Invalid username or password !!");  
 }  
}

* **Difference between InMemoryUserDetailManager and DataBase user for login.**

If we want to user username and password from database then we have to change some configuration.

1. Adding spring-boot-starter-data-jpa and mysql dependencies in the pom.xml file.

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
  
<dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
 <scope>runtime</scope>  
</dependency>

1. Adding database configuration in the application.properties file or application.yaml file.
2. Create User entity for storing user information and it can be use as User class to implement UserDetails interface.

public class User implements UserDetails {  
 @Id  
 private String userId;  
 private String email;  
 private String name;  
 private String password;  
 private String about;  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return null;  
 }  
  
 @Override  
 public String getUsername() {  
 return this.email;  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
}

1. Create Custom UserDetailsService class named CustomUserDetailService to load username from database.

@Service  
public class CustomUserDetailService implements UserDetailsService {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
  
 //load from database  
  
 User user = userRepository.findByEmail(username).orElseThrow(() -> new RuntimeException("User not found !!!"));  
  
 return user;  
 }  
}

1. Use this CustomUserDetailService class to all places where UserDetailService bean has Autowired like AuthController.java and JWTAuthenticationFilter.java.
2. Adding DAOAuthenticationProvider in the SecurityConfig.java file.

@Bean  
public DaoAuthenticationProvider daoAuthenticationProvider(){  
 DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();  
 authenticationProvider.setUserDetailsService(customUserDetailService);  
 authenticationProvider.setPasswordEncoder(passwordEncoder);  
 return authenticationProvider;  
}

**Note :-** After done all of the above configurations you test spring security with JWT. If user detail not there in the data base then create api for create-user and put this api url in the request matcher and permitAll in the SecurityConfig file.

.requestMatchers("/auth/create-user").permitAll()