JSON WEB TOKEN

Step 1) add 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>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.1</version>
</dependency>
```

Step 2) add entity

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AppUser {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "user_id")
    private Long userId;
    @Column(name = "user_name")
    private String username;
    @Column(name = "password")
    private String password;
    @Column(name = "user_role")
    private String userRole;
}
```

Step 3) add Custom user

```
import ezihire.model.AppUser;
import ezihire.model.Authority;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.util.*;

@Component
public class CustomUserDetails implements UserDetails {


    private Optional<AppUser> appUser;

    public CustomUserDetails(Optional<AppUser> appUser) {
        super();
        this.appUser = appUser;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities()
{
        Set<Authority> set = new HashSet<>();
        set.add(new Authority(appUser.get().getUserRole()));
        return null;
    }

    @Override
    public String getPassword() {
        return appUser.get().getPassword();
    }

    @Override
    public String getUsername() {
        return appUser.get().getUsername();
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
```

```
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

Step 4) add authority class

```
import org.springframework.security.core.GrantedAuthority;

public class Authority implements GrantedAuthority {

    private String authority;

    public Authority(String authority){
        this.authority=authority;
    }

    @Override
    public String getAuthority() {
        return this.authority;
    }
}
```

Step 5) add user repository

```
@Repository
public interface UserRepository extends
JpaRepository<AppUser,Long> {
    Optional<AppUser> findByUsername(String username);
}
```

Step 6) add tokenRequestDTO and tokenResponseDTO

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class TokenRequestDTO {

    @ApiModelProperty(position = 0)
    private String username;
    @ApiModelProperty(position = 1)
    private String password;

}
```

```
@Getter
@Setter
@AllArgsConstructor
public class TokenResponseDTO {

    @ApiModelProperty(position = 0)
    private final String token;

}
```

Step 7) add MyUserDetailsService class

```
@Service
public class MyUserDetailsService implements UserDetailsService {


    @Autowired
    private UserRepository userRepository;

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

        Optional<AppUser> appUser =
userRepository.findByUsername(username);
        if (!appUser.isPresent()) {
```

```
            throw new
CustomException(HttpStatus.BAD_REQUEST.value(),
HttpStatus.BAD_REQUEST, "Username doesn't exists !");
        }

        return new CustomUserDetails(appUser);
    }
}
```

Step 8) add websecurityconfig class

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

    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private JwtAuthenticationEntryPoint unauthorizedHandler;
    @Bean
    public AuthenticationManager authenticationManager() throws
Exception {
        return super.authenticationManager();
    }
    @Autowired
    private JwtAuthenticationFilter jwtAuthenticationFilter;
    @Bean
    public BCryptPasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth )
throws Exception {

auth.userDetailsService(this.userDetailsService).passwordEncoder(p
asswordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable().cors().disable()
        .authorizeRequests()
```

```
                .antMatchers("/api/generate/token","/").permitAll()
                .antMatchers(HttpMethod.OPTIONS).permitAll()
                .anyRequest().authenticated()
                .and()

.exceptionHandling().authenticationEntryPoint(unauthorizedHandler)
                .and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.S
TATELESS);
        http.addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);
    }
```

Step 9) add jwtAuthenticationEntryPointClass

```
@Component
public class JwtAuthenticationEntryPoint implements
AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest request,
HttpServletResponse response, AuthenticationException
authException) throws IOException, ServletException {

response.sendError(HttpServletResponse.SC_UNAUTHORIZED,"Unauthoriz
ed");
    }
}
```

Step 10) add JwtAuthenticationFilter

```
@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter
{
    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    private JwtUtils jwtUtils;

    @Override
```

```
   protected void doFilterInternal(HttpServletRequest
httpServletRequest, HttpServletResponse httpServletResponse,
FilterChain filterChain) throws ServletException, IOException {

        final String
requestTokenHeader=httpServletRequest.getHeader("Authorization");
        String username=null;
        String jwtToken=null;

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

            jwtToken = requestTokenHeader.substring(7);
            try {
                username = this.jwtUtils.extractUsername(jwtToken);
            }catch (ExpiredJwtException exception)
            {
                exception.printStackTrace();
                System.out.println("Token has expired ");
            }catch (Exception e){
                e.printStackTrace();
                System.out.println("error");
            }
        }else {
            System.out.println("Invalid Token , did not start with
bearer string ");
        }

        //validated
        if (username != null &&
SecurityContextHolder.getContext().getAuthentication()==null)
        {
            final UserDetails
userDetails=this.userDetailsService.loadUserByUsername(username);
            if (this.jwtUtils.validateToken(jwtToken,userDetails)){
                //token valid
                UsernamePasswordAuthenticationToken
usernamePasswordAuthenticationToken
                        =new
UsernamePasswordAuthenticationToken(userDetails,null,userDetails.g
etAuthorities());
                usernamePasswordAuthenticationToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(httpServletRequest))
;
```

```
SecurityContextHolder.getContext().setAuthentication(usernamePassw
ordAuthenticationToken);
            }
        }else {
            System.out.println("Token is not valid");
        }

filterChain.doFilter(httpServletRequest,httpServletResponse);
    }
}
```

Step 11) add jwtutil class

```
@Service
public class JwtUtils {

    private String SECRET_KEY = "assessment";

    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).getB
ody();
    }
    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(n
ew Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis()
+ 1000 * 60    ))

//                .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));
    }
}
```

Step 12) add tokenController

```
@RestController
@Api(tags = "token")

public class TokenController {

    @Autowired
    private TokenService tokenService;

    @PostMapping("/api/generate/token")
    public CustomResponseEntity generateToken(@RequestBody
TokenRequestDTO tokenRequestDTO) throws Exception{
        return CustomResponseEntity.builder()
                .code(HttpStatus.OK.value())
                .status(CustomResponseStatus.SUCCESS.getStatus())
                .message(CustomResponseStatus.SUCCESS.getMessage())

.data(tokenService.generateToken(tokenRequestDTO)).build();
    }
}
```

chail