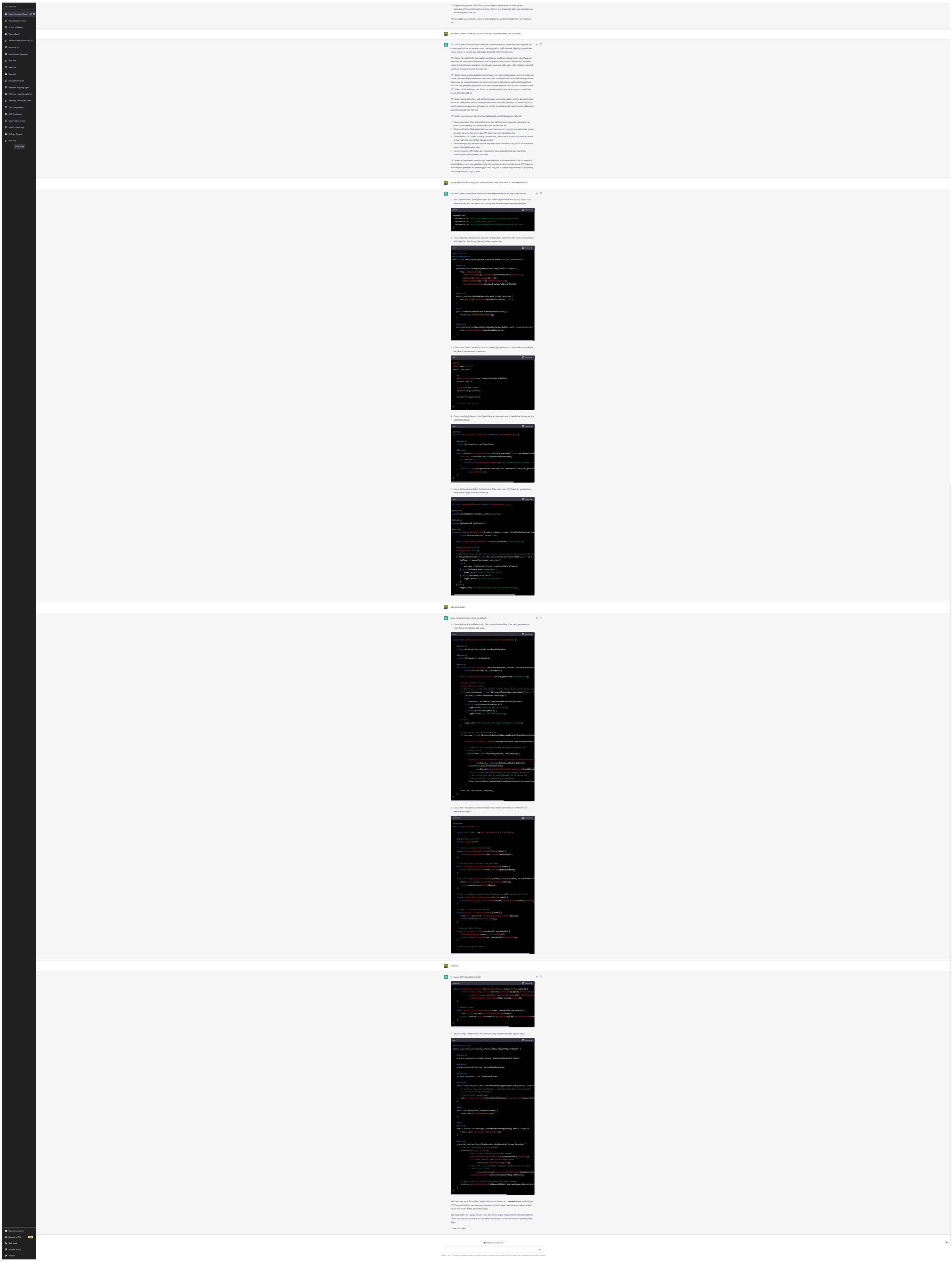

7. Project management skills: Cloud computing ke implementation mein project management ka accha experience hona chahiye, jisse ki aap sahi planning, execution aur monitoring kar sakte hai.

Yeh kuch skills aur resources hai, jo cloud computing ke implementation ke liye important hai.

jwt token ky hota hai or iska ky use hai or isko kese implement kare sab batay

JWT (JSON Web Token) ek secure way hai authentication aur information exchange ke liye, jo web applications aur services mein use kiya jata hai. JWT token ek digitally signed token hai, jo ek user ki identity aur authorized access ko represent karta hai.

JWT token ke 3 parts hote hai: header, payload aur signature. Header mein token type aur algorithm ke baare mein information hoti hai, payload mein user ki information aur token expiry time hoti hai aur signature mein header aur payload ka hash code hota hai, jo digital signature ke roop mein use kiya jata hai.

JWT token ka use web applications aur services mein user authentication ke liye kiya jata hai. Jab ek user apne login credentials enter karta hai, tab server user ko ek JWT token generate karke, user ko provide karta hai. Iss token mein user ki identity aur authorized access hoti hai. User jab apne web applications aur services mein request karta hai, tab uss request mein JWT token bhi include hota hai. Server iss token ko verify karta hai aur user ko authorized access provide karta hai.

JWT token ka use karte hue, web applications aur services ko user ki identity aur authorized access ko verify karne ke liye, server par additional load nahi padta hai. Iss token ke use se user ki session management bhi aasan ho jati hai, kyunki server ko user ki session data store karne ki zaroorat nahi hoti hai.

JWT token ko implement karne ke liye, aapko kuch steps follow karne hote hai:

1. Token generation: User authentication ke liye, JWT token ko generate karna hota hai, jisme user ki identity aur authorized access include hoti hai.
2. Token verification: Web applications aur services ko user ki identity aur authorized access ko verify karne ke liye, server par JWT token ko verify karna hota hai.
3. Token refresh: JWT token ki expiry time hoti hai, isliye user ki session ko continue rakhne ke liye, JWT token ko refresh karna hota hai.
4. Token storage: JWT token ko secure way mein store karna hota hai, jisse ki unauthorized access se protect kiya ja sake.
5. Token revocation: JWT token ko revoke karne ka process bhi hota hai, jisse ki kisi unauthorized user ko access nahi mile.

JWT token ko implement karne ke liye, aapko libraries aur frameworks ka use kar sakte hai, jaise ki Node.js mein jsonwebtoken library ka use kiya ja sakta hai. Iske alawa, JWT token ko manually bhi generate aur verify kiya ja sakta hai, jisse ki custom requirements ke according sahi implementation kiya ja sake.

ky aap jwt token ko spring boot me implement karke bata sakte ho with explanation

Yes, main aapko Spring Boot mein JWT token implementation ke steps batata hoon:

1. Add Dependencies: Spring Boot mein JWT token implement karne ke liye, aapko kuch dependencies add karna hota hai. build.gradle file mein dependencies add kijiye.

```python
dependencies {
  implementation 'org.springframework.boot:spring-boot-starter-web'
  implementation 'io.jsonwebtoken:jjwt:0.9.1'
  implementation 'org.springframework.boot:spring-boot-starter-security'
}
```

2. Create Security Configuration: Security configuration class mein JWT token configuration add kijiye. Ye class Spring Security ke liye required hai.

```scss
@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

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

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers(HttpMethod.OPTIONS, "/**");
    }

    @Bean
    public AuthenticationFilter authenticationFilter() {
        return new AuthenticationFilter();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService);
    }
}
```

3. Create User Entity: User entity class ko create kijiye, jisme user ki information store ki jati hai, jaise ki username aur password.

```less
@Entity
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String username;

    private String password;

    // getters and setters
}
```

4. Create UserDetailsService: UserDetailsService class mein user ki details fetch karne ke liye method add kijiye.

```java
@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFound
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("User not found with username: " +
        }
        return new org.springframework.security.core.userdetails.User(user.getUsern
                new ArrayList<>());
    }
}
```

5. Create AuthenticationFilter: AuthenticationFilter class mein JWT token ko generate aur verify karne ke liye methods add kijiye.

```java
lic class AuthenticationFilter extends OncePerRequestFilter {

@Autowired
private UserDetailsServiceImpl userDetailsService;

@Autowired
private JwtTokenUtil jwtTokenUtil;

@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse resp
        throws ServletException, IOException {

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

    String username = null;
    String jwtToken = null;
    // JWT Token is in the form "Bearer token". Remove Bearer word and get only the
    if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer ")) {
        jwtToken = requestTokenHeader.substring(7);
        try {
            username = jwtTokenUtil.getUsernameFromToken(jwtToken);
        } catch (IllegalArgumentException e) {
            logger.error("Unable to get JWT Token");
        } catch (ExpiredJwtException e) {
            logger.error("JWT Token has expired");
        }
    } else {
        logger.warn("JWT Token does not begin with Bearer String");
```

isko pura batao

Sure, continuing from where we left off:

5. Create AuthenticationFilter (contd.): Ab, authentication filter class mein generate aur verify karne ke methods add kijiye.

```java
public class AuthenticationFilter extends OncePerRequestFilter {

    @Autowired
    private UserDetailsServiceImpl userDetailsService;

    @Autowired
    private JwtTokenUtil jwtTokenUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
            throws ServletException, IOException {

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

        String username = null;
        String jwtToken = null;
        // JWT Token is in the form "Bearer token". Remove Bearer word and get only
        if (requestTokenHeader != null && requestTokenHeader.startsWith("Bearer "))
            jwtToken = requestTokenHeader.substring(7);
            try {
                username = jwtTokenUtil.getUsernameFromToken(jwtToken);
            } catch (IllegalArgumentException e) {
                logger.error("Unable to get JWT Token");
            } catch (ExpiredJwtException e) {
                logger.error("JWT Token has expired");
            }
        } else {
            logger.warn("JWT Token does not begin with Bearer String");
        }

        // Once we get the token validate it.
        if (username != null && SecurityContextHolder.getContext().getAuthenticatio

            UserDetails userDetails = this.userDetailsService.loadUserByUsername(us

            // if token is valid configure Spring Security to manually set
            // authentication
            if (jwtTokenUtil.validateToken(jwtToken, userDetails)) {

                UsernamePasswordAuthenticationToken usernamePasswordAuthenticationT
                        userDetails, null, userDetails.getAuthorities());
                usernamePasswordAuthenticationToken
                        .setDetails(new WebAuthenticationDetailsSource().buildDetai
                // 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(usernamePasswo
            }
        }
        chain.doFilter(request, response);
    }
}
```

6. Create JWT Token Util: JwtTokenUtil class mein token generate aur verify karne ke methods add kijiye.

```typescript
@Component
public class JwtTokenUtil {

    public static final long JWT_TOKEN_VALIDITY = 5 * 60 * 60;

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

    // 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 retrieving 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
```

continue

6. Create JWT Token Util: (contd.)

```typescript
private String doGenerateToken(Map<String, Object> claims, String subject) {
        return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new
                .setExpiration(new Date(System.currentTimeMillis() + JWT_TOKEN_VALI
                .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
    }
}
```

7. Add Security Configurations: Ab spring security configurations ko update kijiye.

```scss
@EnableWebSecurity
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
        // configure AuthenticationManager so that it knows from where to load
        // user for matching credentials
        // Use BCryptPasswordEncoder
        auth.userDetailsService(jwtUserDetailsService).passwordEncoder(passwordEnco
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return 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").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(jwtAuthenticat
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        // Add a filter to validate the tokens with every request
        httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticati
    }
}
```

Iske baad aap apne Spring Boot application ko run karein. Ab, "`/authenticate`" endpoint pe POST request bhejkar username aur password ke saath. Agar username aur password sahi hai, to server JWT token generate karega.

Iske baad, token ko request header mein add karke secure endpoints tak pahunch sakte hai. Token ko verify karke server user ko authenticate karega aur secure resource access karene dega.

I hope this helps!

ChatGPT Feb 13 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.