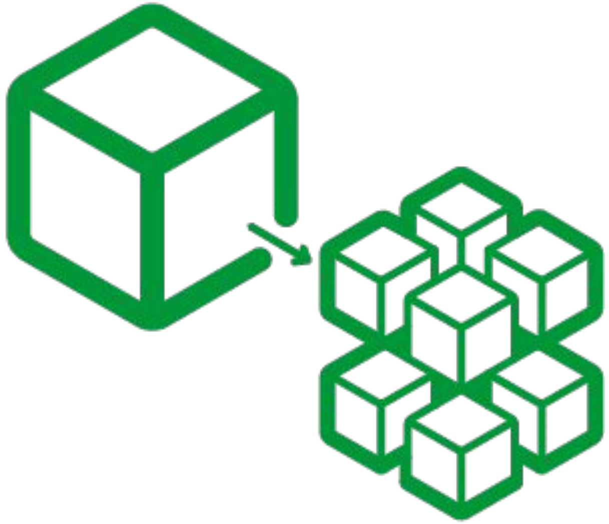
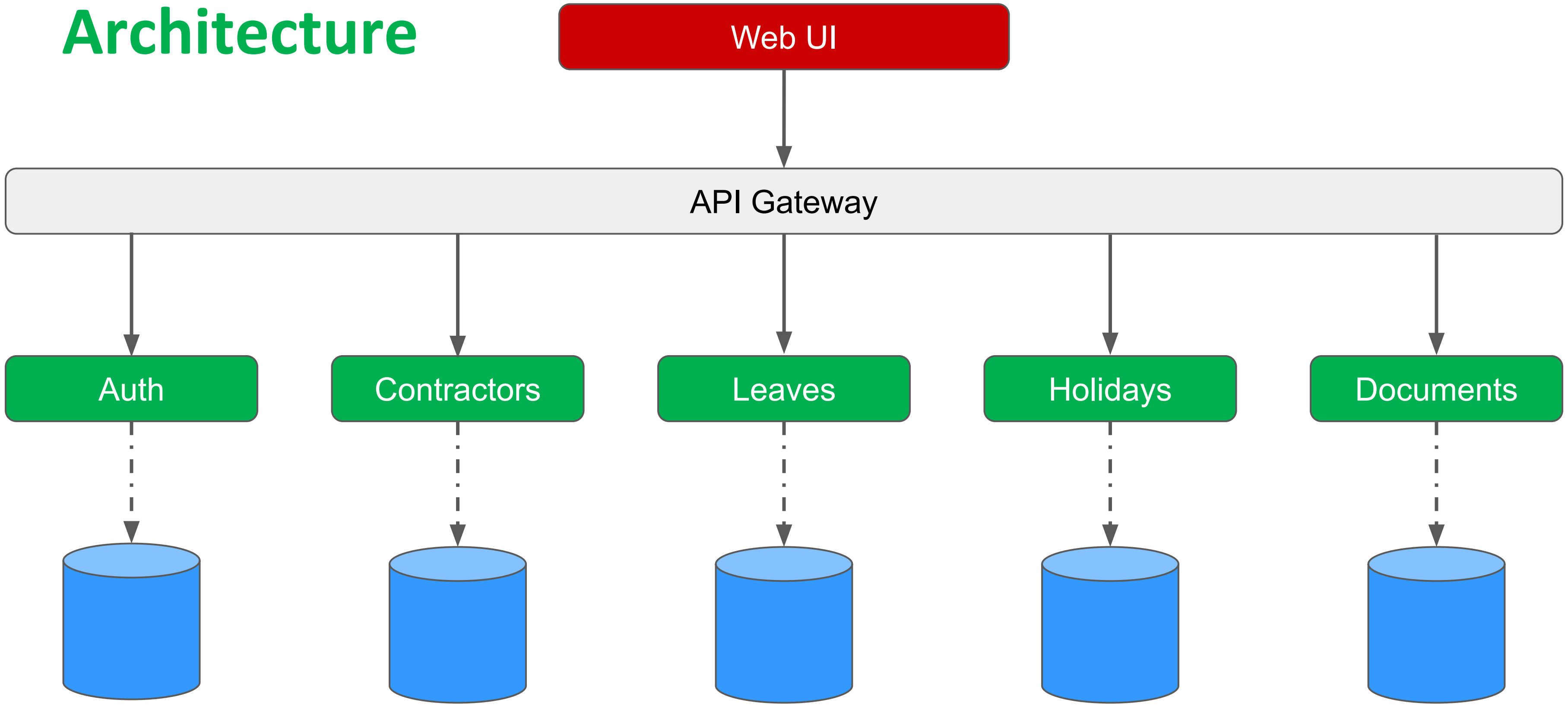
INTERLINK

JWT Auth

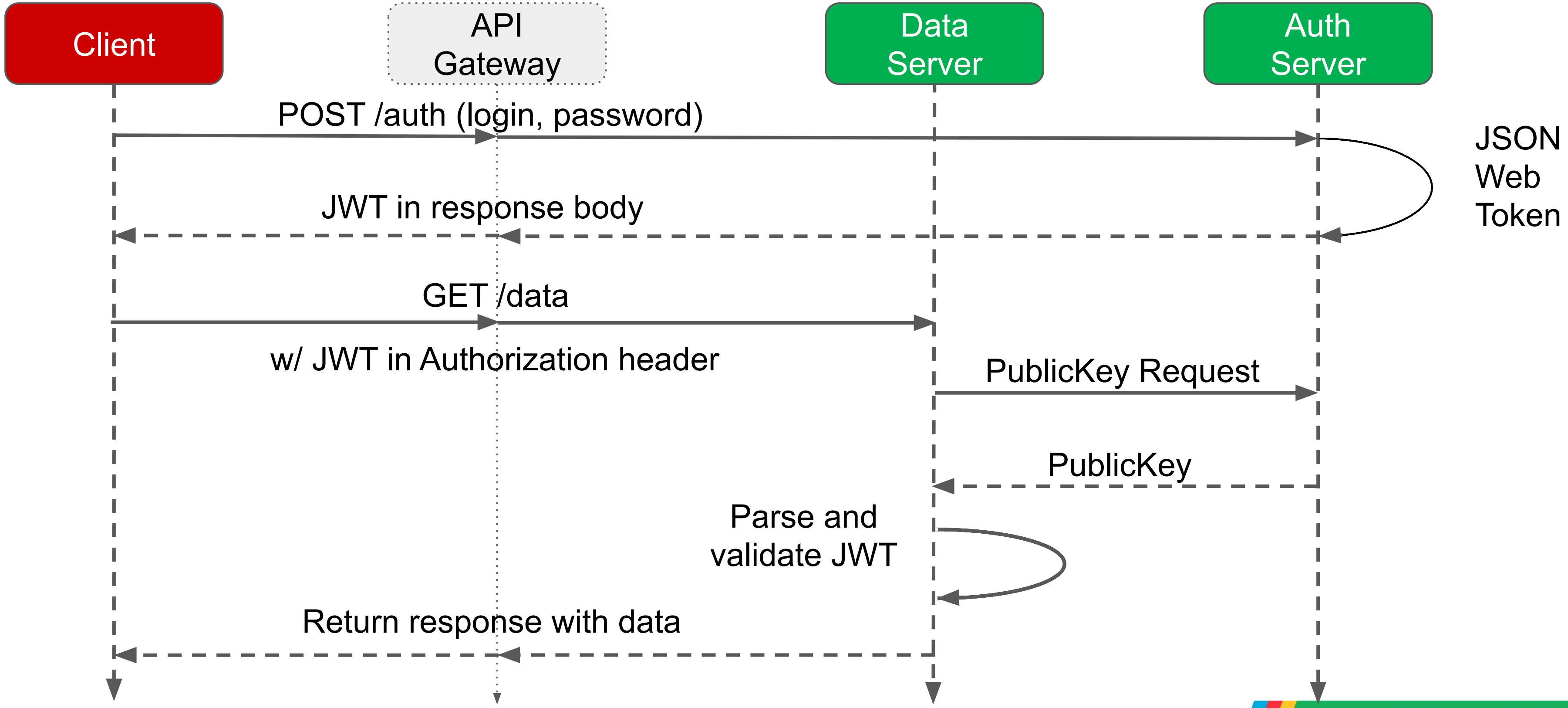
In microservice architecture



Architecture

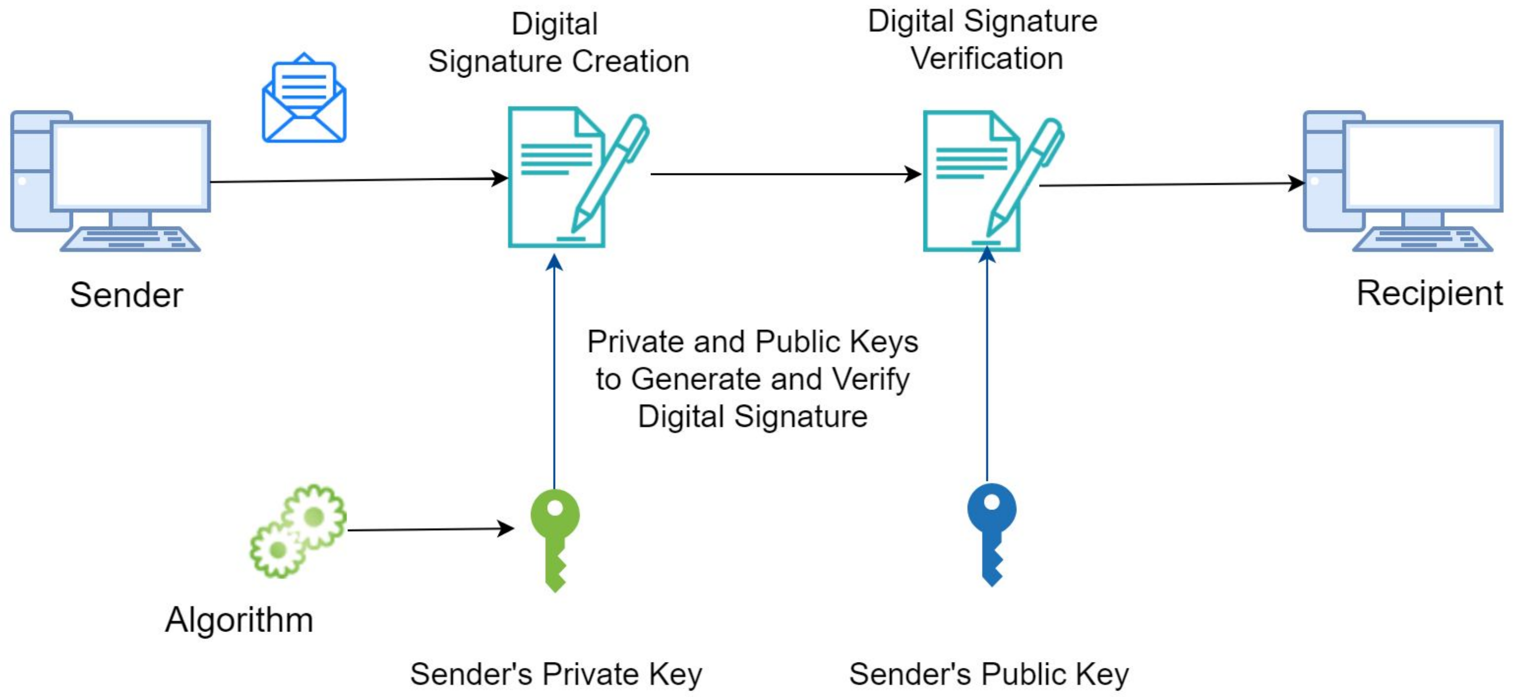


Auth flow

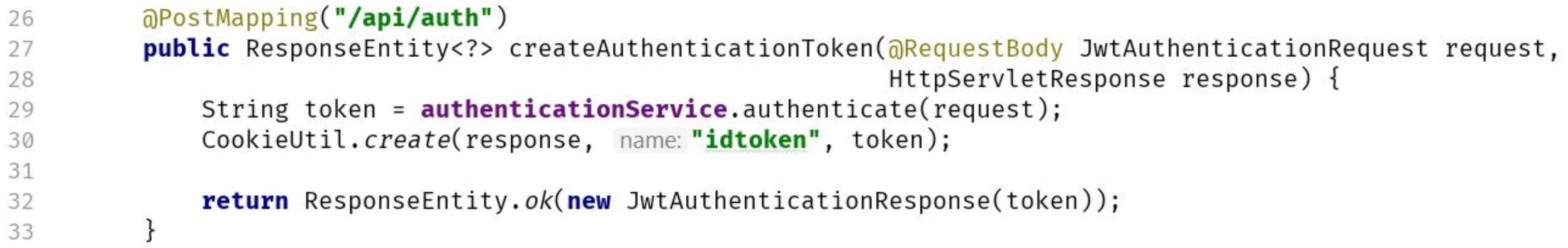


INTERLINK

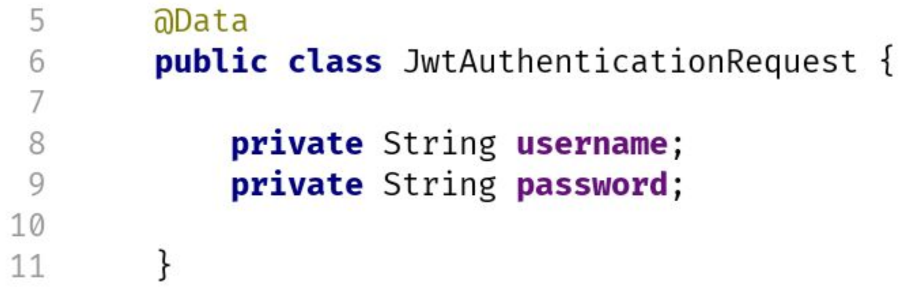
Asymmetric signature


Digital Signature Creation
Digital Signature Verification
Sender
Recipient
Private and Public Keys to Generate and Verify Digital Signature
Algorithm
Sender's Private Key
Sender's Public Key

POST method with username and password

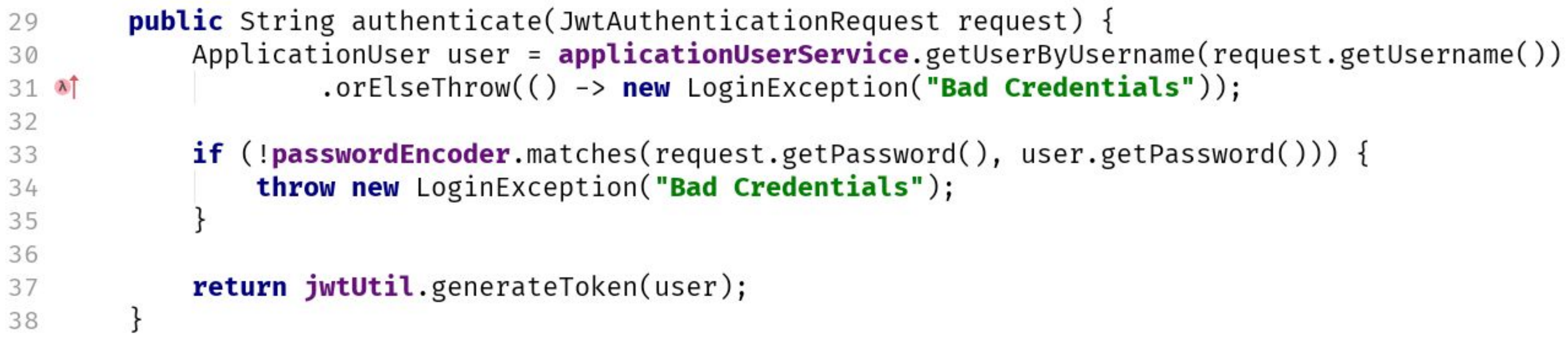


```
@PostMapping("/api/auth")
public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtAuthenticationRequest request,
                                                    HttpServletResponse response) {
    String token = authenticationService.authenticate(request);
    CookieUtil.create(response,  name: "idtoken", token);

    return ResponseEntity.ok(new JwtAuthenticationResponse(token));
}
```



```
@Data
public class JwtAuthenticationRequest {

    private String username;
    private String password;

}
```

Authenticate

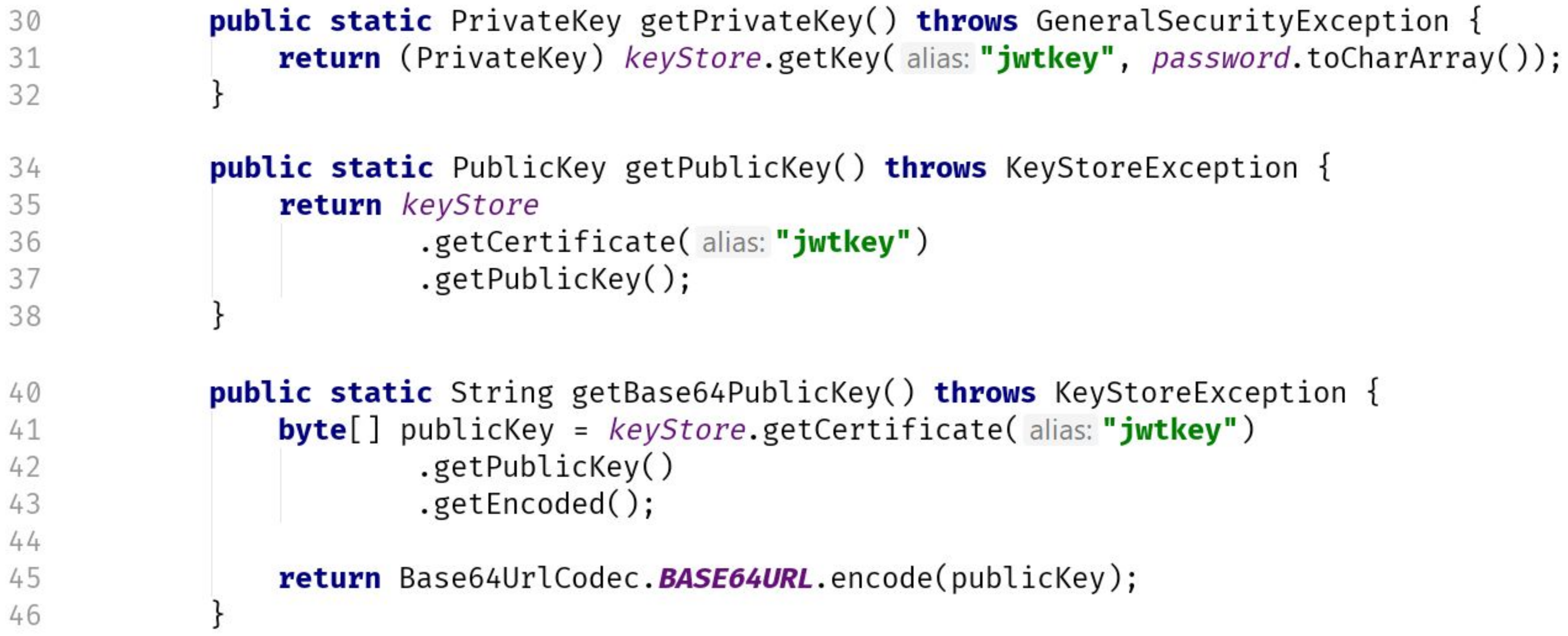


```java
    public String authenticate(JwtAuthenticationRequest request) {
        ApplicationUser user = applicationUserService.getUserByUsername(request.getUsername())
                .orElseThrow(() -> new LoginException("Bad Credentials"));

        if (!passwordEncoder.matches(request.getPassword(), user.getPassword())) {
            throw new LoginException("Bad Credentials");
        }

        return jwtUtil.generateToken(user);
    }
```

Generate Token

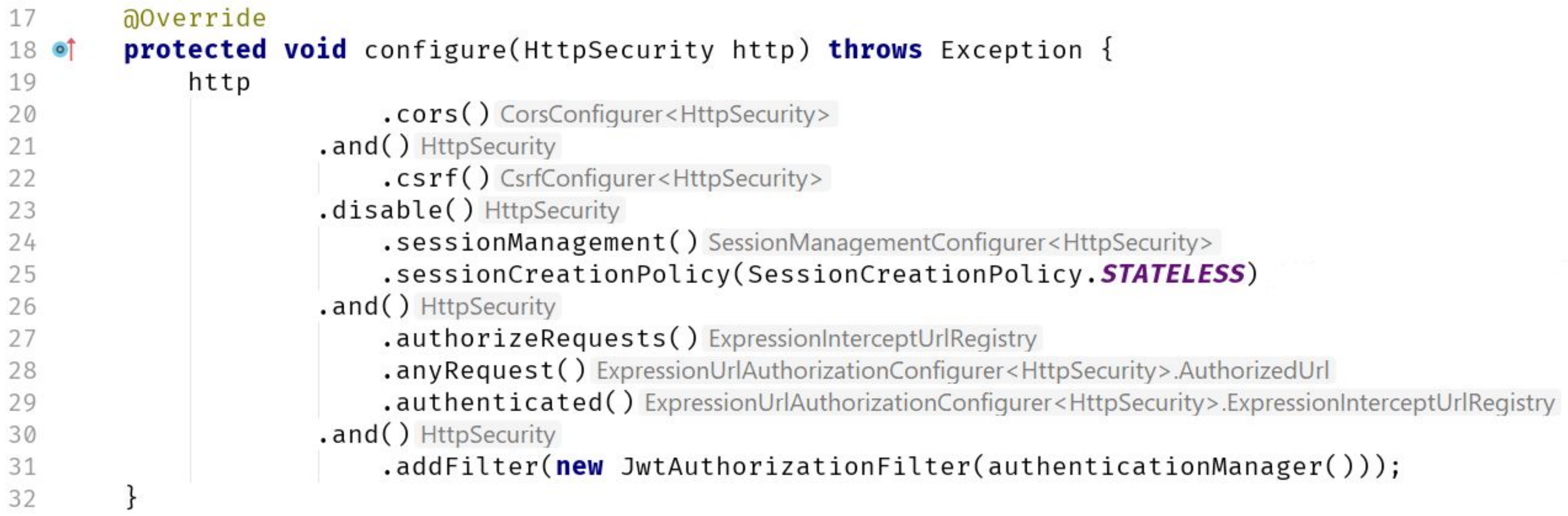
```java
    public String generateToken(ApplicationUser user) {
        Map<String, Object> claims = new HashMap<>();
        claims.put("roles", mapToCommaSeparatedRole(user.getRoles()));
        claims.put("authorities", mapToCommaSeparatedAuthority(user.getAuthorities()));

        return doGenerateToken(claims, String.valueOf(user.getContractorId()));
    }

    private String doGenerateToken(Map<String, Object> claims, String subject) {
        final Date createdDate = clock.now();
        final Date expirationDate = calculateExpirationDate(createdDate);

        return Jwts.builder()
                .setClaims(claims)
                .setSubject(subject)
                .setIssuedAt(createdDate)
                .setExpiration(expirationDate)
                .signWith(SignatureAlgorithm.RS256, SigningKeyUtil.getPrivateKey())
                .compact();
    }
```

Asymmetric Signature



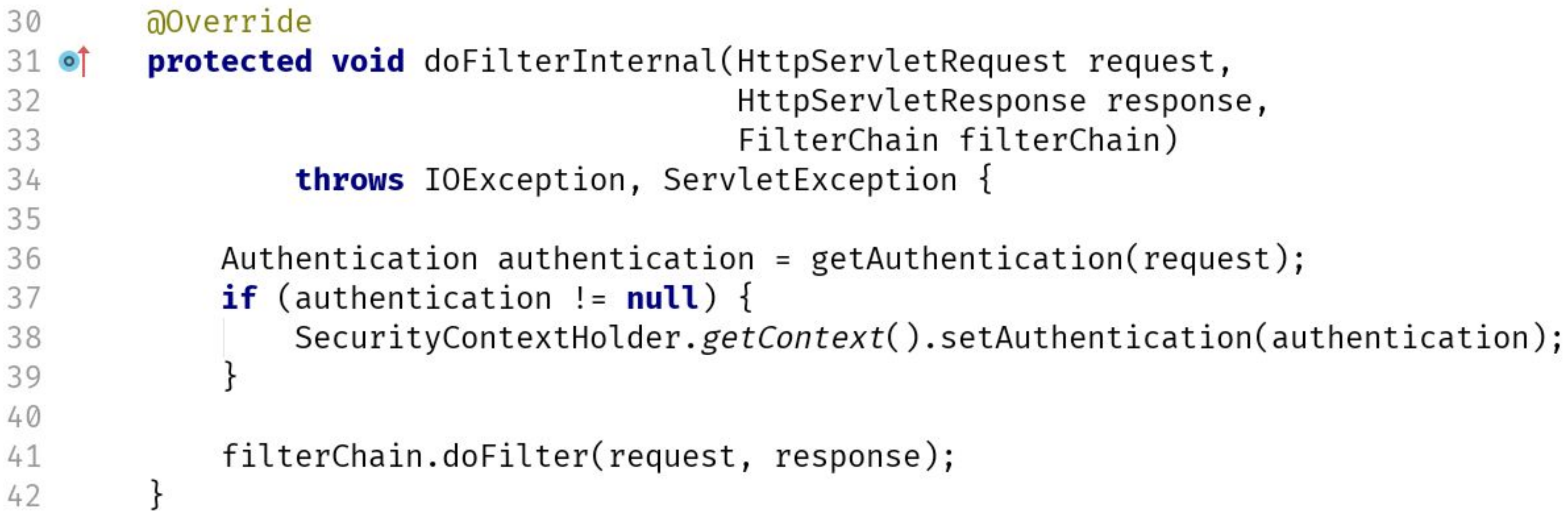
```java
    public static PrivateKey getPrivateKey() throws GeneralSecurityException {
        return (PrivateKey) keyStore.getKey( alias: "jwtkey", password.toCharArray());
    }

    public static PublicKey getPublicKey() throws KeyStoreException {
        return keyStore
                .getCertificate( alias: "jwtkey")
                .getPublicKey();
    }

    public static String getBase64PublicKey() throws KeyStoreException {
        byte[] publicKey = keyStore.getCertificate( alias: "jwtkey")
                .getPublicKey()
                .getEncoded();

        return Base64UrlCodec.BASE64URL.encode(publicKey);
    }
```

WebSecurityConfig



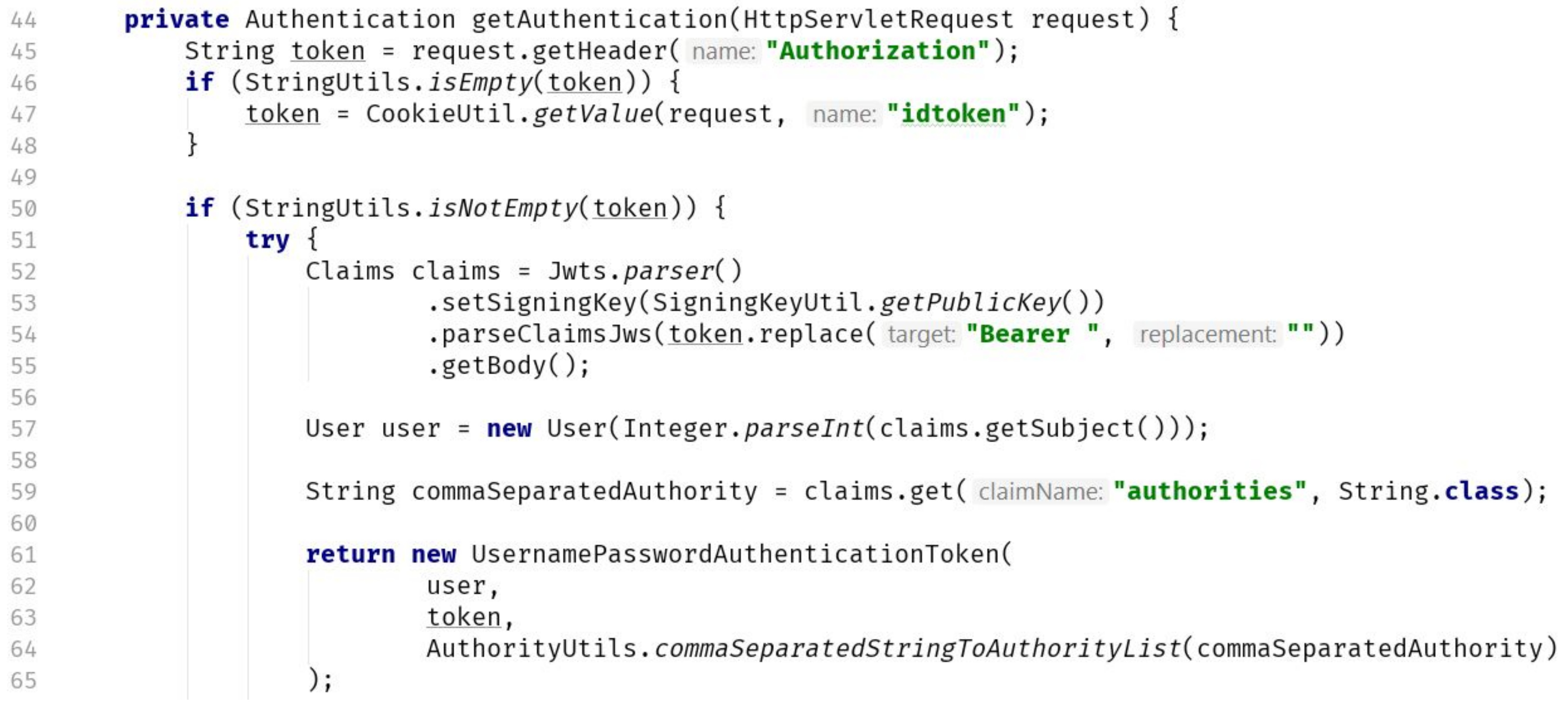
```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .cors() CorsConfigurer<HttpSecurity>
            .and() HttpSecurity
                .csrf() CsrfConfigurer<HttpSecurity>
            .disable() HttpSecurity
                .sessionManagement() SessionManagementConfigurer<HttpSecurity>
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and() HttpSecurity
                .authorizeRequests() ExpressionInterceptUrlRegistry
                .anyRequest() ExpressionUrlAuthorizationConfigurer<HttpSecurity>.AuthorizedUrl
                .authenticated() ExpressionUrlAuthorizationConfigurer<HttpSecurity>.ExpressionInterceptUrlRegistry
            .and() HttpSecurity
                .addFilter(new JwtAuthorizationFilter(authenticationManager()));
    }
```

JwtAuthorizationFilter



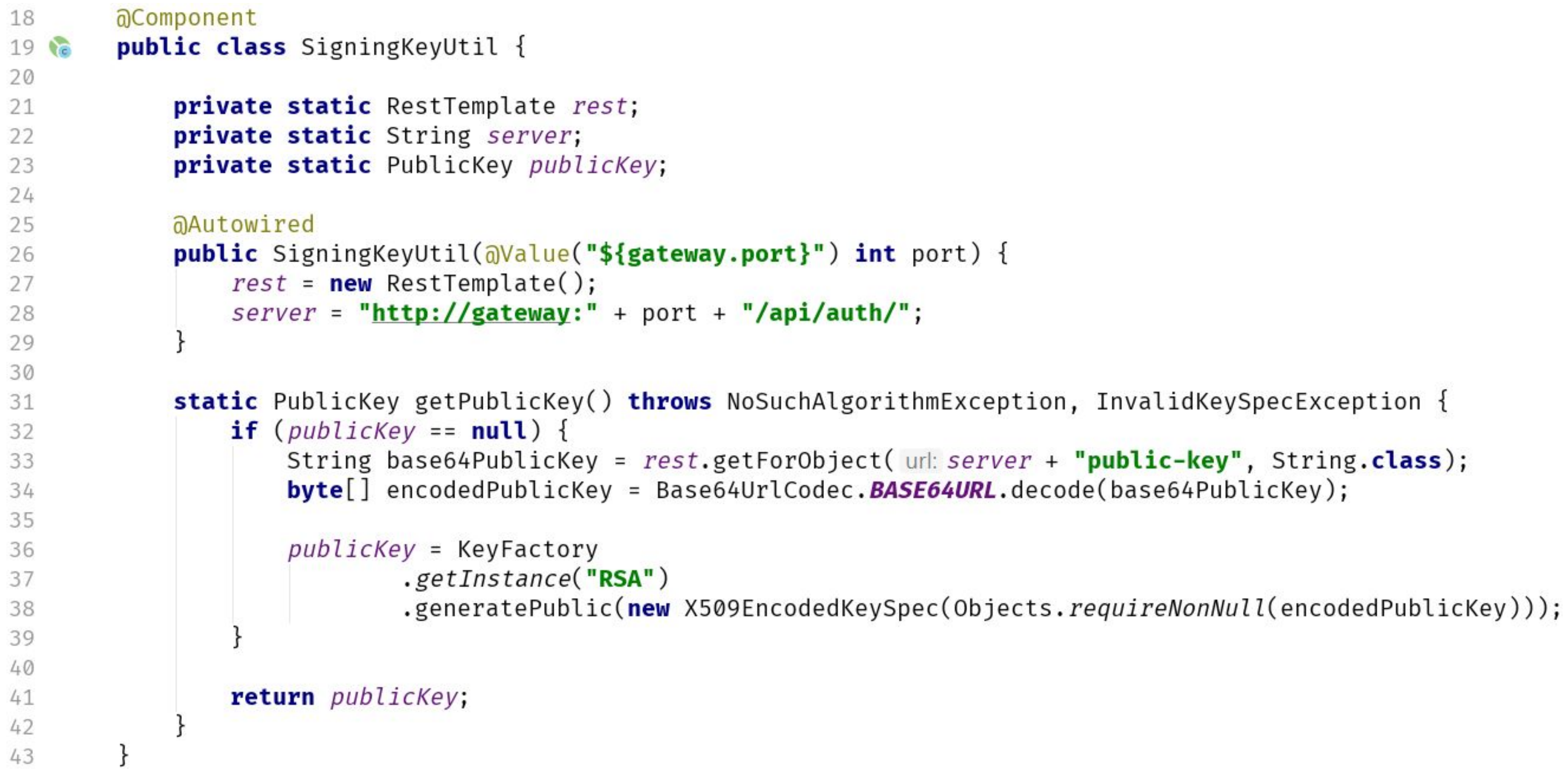
```java
@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain)
        throws IOException, ServletException {

    Authentication authentication = getAuthentication(request);
    if (authentication != null) {
        SecurityContextHolder.getContext().setAuthentication(authentication);
    }

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

JwtAuthorizationFilter getAuthentication() method



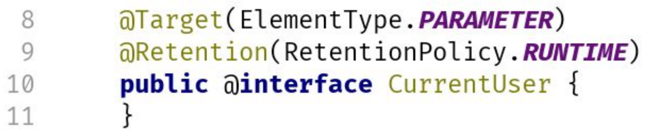
```java
private Authentication getAuthentication(HttpServletRequest request) {
    String token = request.getHeader( name: "Authorization");
    if (StringUtils.isEmpty(token)) {
        token = CookieUtil.getValue(request,  name: "idtoken");
    }

    if (StringUtils.isNotEmpty(token)) {
        try {
            Claims claims = Jwts.parser()
                    .setSigningKey(SigningKeyUtil.getPublicKey())
                    .parseClaimsJws(token.replace( target: "Bearer ",  replacement: ""))
                    .getBody();

            User user = new User(Integer.parseInt(claims.getSubject()));

            String commaSeparatedAuthority = claims.get( claimName: "authorities", String.class);

            return new UsernamePasswordAuthenticationToken(
                    user,
                    token,
                    AuthorityUtils.commaSeparatedStringToAuthorityList(commaSeparatedAuthority)
            );
```

How other microservices get a public key?

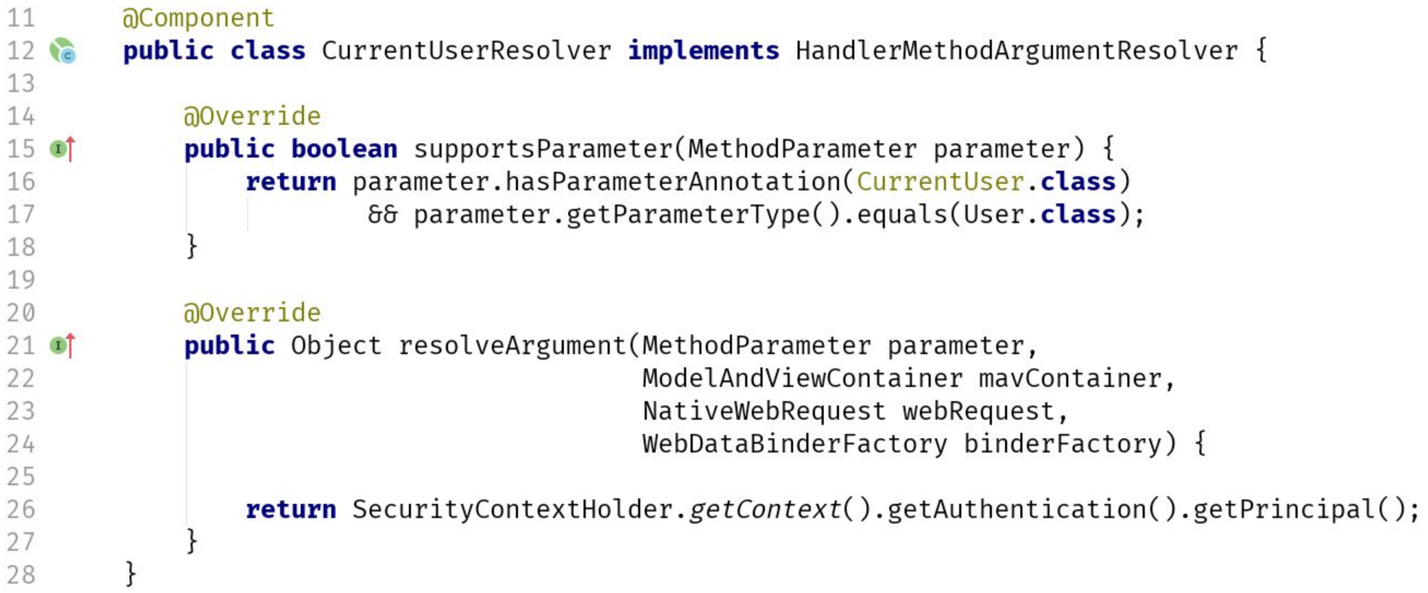


```java
@Component
public class SigningKeyUtil {

    private static RestTemplate rest;
    private static String server;
    private static PublicKey publicKey;

    @Autowired
    public SigningKeyUtil(@Value("${gateway.port}") int port) {
        rest = new RestTemplate();
        server = "http://gateway:" + port + "/api/auth/";
    }

    static PublicKey getPublicKey() throws NoSuchAlgorithmException, InvalidKeySpecException {
        if (publicKey == null) {
            String base64PublicKey = rest.getForObject( url: server + "public-key", String.class);
            byte[] encodedPublicKey = Base64UrlCodec.BASE64URL.decode(base64PublicKey);

            publicKey = KeyFactory
                    .getInstance("RSA")
                    .generatePublic(new X509EncodedKeySpec(Objects.requireNonNull(encodedPublicKey)));
        }

        return publicKey;
    }
}
```

Bonus: CurrentUser annotation

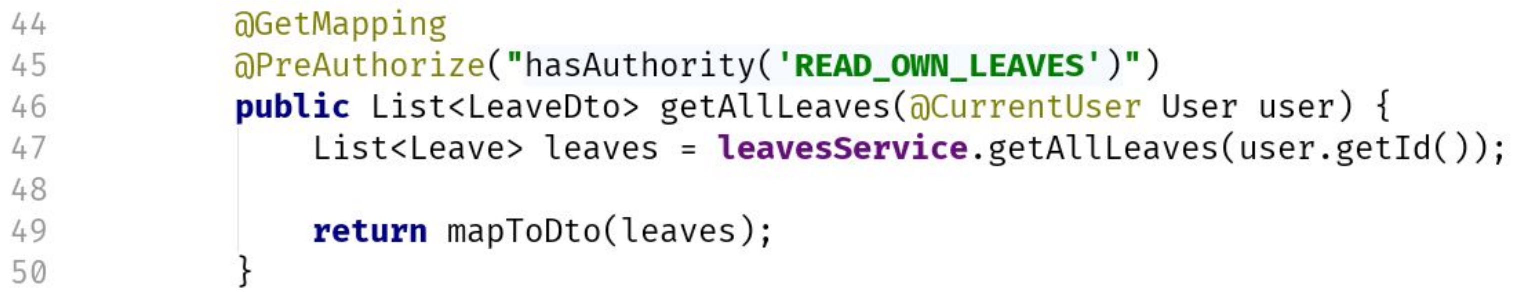


```java
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
public @interface CurrentUser {
}
```



```java
@Component
public class CurrentUserResolver implements HandlerMethodArgumentResolver {

    @Override
    public boolean supportsParameter(MethodParameter parameter) {
        return parameter.hasParameterAnnotation(CurrentUser.class)
                && parameter.getParameterType().equals(User.class);
    }

    @Override
    public Object resolveArgument(MethodParameter parameter,
                                  ModelAndViewContainer mavContainer,
                                  NativeWebRequest webRequest,
                                  WebDataBinderFactory binderFactory) {

        return SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    }
}
```

CurrentUser annotation usage



```java
    @GetMapping
    @PreAuthorize("hasAuthority('READ_OWN_LEAVES')")
    public List<LeaveDto> getAllLeaves(@CurrentUser User user) {
        List<Leave> leaves = leavesService.getAllLeaves(user.getId());

        return mapToDto(leaves);
    }
```

Future plan

JSON Web Key Set (JWKS) - is a set of keys containing the public keys that should be used to verify any JSON Web Token issued by the authorization server and signed using the RS256 signing algorithm.

INTERLINK

Thank you for your attention :)

blog.interlink-ua.com

friends@interlink-ua.com

facebook.com/interlinkua