JWT RSA256 Token Service with Spring Boot (In-Memory Key Pair)

This document contains a complete implementation of a JWT service using RSA256 algorithm (RS256) in Spring Boot.

Features Covered:
1. Generate JWT token using in-memory RSA key pair
2. Validate token and extract username
3. Extract custom claims (e.g., role, email)
4. Check token expiration
5. Optional methods for saving/loading keys from file

Full JwtRsaService.java Implementation:

```
@Service
public class JwtRsaService {

    private PrivateKey privateKey;
    private PublicKey publicKey;
    private final long EXPIRATION_TIME = 1000 * 60 * 60; // 1 hour

    @PostConstruct
    public void init() throws NoSuchAlgorithmException {
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(2048);
        KeyPair keyPair = keyGen.generateKeyPair();
        this.privateKey = keyPair.getPrivate();
        this.publicKey = keyPair.getPublic();
    }

    public String generateToken(String username, Map<String, Object> claims) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
                .signWith(privateKey, SignatureAlgorithm.RS256)
                .compact();
    }

    public String validateTokenAndGetUsername(String token) {
        try {
            Claims claims = Jwts.parserBuilder()
                    .setSigningKey(publicKey)
                    .build()
                    .parseClaimsJws(token)
                    .getBody();
            return claims.getSubject();
        } catch (JwtException e) {
            throw new RuntimeException("Invalid or expired JWT token", e);
        }
    }

    public <T> T extractClaim(String token, String claimKey, Class<T> claimType) {
        Claims claims = Jwts.parserBuilder()
                .setSigningKey(publicKey)
                .build()
                .parseClaimsJws(token)
                .getBody();
        return claims.get(claimKey, claimType);
    }

    public boolean isTokenExpired(String token) {
        try {
            Date expiration = Jwts.parserBuilder()
                    .setSigningKey(publicKey)
                    .build()
                    .parseClaimsJws(token)
                    .getBody()
                    .getExpiration();
            return expiration.before(new Date());
        } catch (JwtException e) {
            return true;
        }
    }
}
```

Controller to Test:

```
@RestController
@RequestMapping("/api/auth")
public class AuthController {

    private final JwtRsaService jwtService;

    public AuthController(JwtRsaService jwtService) {
        this.jwtService = jwtService;
    }

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestParam String username, @RequestParam String role) {
        Map<String, Object> claims = new HashMap<>();
        claims.put("role", role);
        String token = jwtService.generateToken(username, claims);
        return ResponseEntity.ok(token);
    }

    @GetMapping("/verify")
    public ResponseEntity<String> verify(@RequestHeader("Authorization") String authHeader) {
        String token = authHeader.replace("Bearer ", "");
        String username = jwtService.validateTokenAndGetUsername(token);
        String role = jwtService.extractClaim(token, "role", String.class);
        return ResponseEntity.ok("Valid token for user: " + username + " with role: " + role);
    }
}
```

Tip: For production use, load/save keys securely using keystore or file-based PEM keys.