⑂ aa146db611 ⌄   •••

datahub / **metadata-service** / **auth-impl** / **src** / **main** / **java** / **com** / **datahub** / **authentication** /
token / **StatelessTokenService.java** / <> Jump to ⌄

ksrinath feat(metadata-service-auth): add support for eternal personal access ... ... ✕   🕑 History

⚇ 2 contributors

163 lines (147 sloc) | 5.91 KB   •••

```java
1    package com.datahub.authentication.token;
2
3    import com.datahub.authentication.Actor;
4    import com.datahub.authentication.ActorType;
5    import io.jsonwebtoken.Claims;
6    import io.jsonwebtoken.JwtBuilder;
7    import io.jsonwebtoken.Jwts;
8    import io.jsonwebtoken.SignatureAlgorithm;
9    import java.nio.charset.StandardCharsets;
10   import java.security.Key;
11   import java.util.ArrayList;
12   import java.util.Base64;
13   import java.util.Date;
14   import java.util.HashMap;
15   import java.util.List;
16   import java.util.Map;
17   import java.util.Objects;
18   import java.util.UUID;
19   import javax.annotation.Nonnull;
20   import javax.annotation.Nullable;
21   import javax.crypto.spec.SecretKeySpec;
22
23   import static com.datahub.authentication.token.TokenClaims.*;
24
25
26   /**
27    * Service responsible for generating JWT tokens for use within DataHub in stateless way.
```

```java
 * This service is responsible only for generating tokens, it will not do anything else with them.
 */
public class StatelessTokenService {

  protected static final long DEFAULT_EXPIRES_IN_MS = 86400000L; // One day by default
  private static final List<String> SUPPORTED_ALGORITHMS = new ArrayList<>();

  static {
    SUPPORTED_ALGORITHMS.add("HS256"); // Only support HS256 today.
  }

  private final String signingKey;
  private final SignatureAlgorithm signingAlgorithm;
  private final String iss;

  public StatelessTokenService(
      @Nonnull final String signingKey,
      @Nonnull final String signingAlgorithm
  ) {
    this(signingKey, signingAlgorithm, null);
  }

  public StatelessTokenService(
      @Nonnull final String signingKey,
      @Nonnull final String signingAlgorithm,
      @Nullable final String iss
  ) {
    this.signingKey = Objects.requireNonNull(signingKey);
    this.signingAlgorithm = validateAlgorithm(Objects.requireNonNull(signingAlgorithm));
    this.iss = iss;
  }

  /**
   * Generates a JWT for an actor with a default expiration time.
   *
   * Note that the caller of this method is expected to authorize the action of generating a token
   *
   */
  public String generateAccessToken(@Nonnull final TokenType type, @Nonnull final Actor actor) {
    return generateAccessToken(type, actor, DEFAULT_EXPIRES_IN_MS);
  }

  /**
   * Generates a JWT for an actor with a specific duration in milliseconds.
   *
   * Note that the caller of this method is expected to authorize the action of generating a token
   *
   */
  @Nonnull
```

```java
 77      public String generateAccessToken(
 78          @Nonnull final TokenType type,
 79          @Nonnull final Actor actor,
 80          @Nullable final Long expiresInMs) {
 81        Objects.requireNonNull(type);
 82        Objects.requireNonNull(actor);
 83
 84        Map<String, Object> claims = new HashMap<>();
 85        claims.put(TOKEN_VERSION_CLAIM_NAME, String.valueOf(TokenVersion.ONE.numericValue)); // Hardco
 86        claims.put(TOKEN_TYPE_CLAIM_NAME, type.toString());
 87        claims.put(ACTOR_TYPE_CLAIM_NAME, actor.getType());
 88        claims.put(ACTOR_ID_CLAIM_NAME, actor.getId());
 89        return generateAccessToken(actor.getId(), claims, expiresInMs);
 90      }
 91
 92      /**
 93       * Generates a JWT for a custom set of claims.
 94       *
 95       * Note that the caller of this method is expected to authorize the action of generating a token
 96       */
 97      @Nonnull
 98      public String generateAccessToken(
 99          @Nonnull final String sub,
100          @Nonnull final Map<String, Object> claims,
101          @Nullable final Long expiresInMs) {
102        Objects.requireNonNull(sub);
103        Objects.requireNonNull(claims);
104        final JwtBuilder builder = Jwts.builder()
105          .addClaims(claims)
106          .setId(UUID.randomUUID().toString())
107          .setSubject(sub);
108
109        if (expiresInMs != null) {
110          builder.setExpiration(new Date(System.currentTimeMillis() + expiresInMs));
111        }
112        if (this.iss != null) {
113          builder.setIssuer(this.iss);
114        }
115        byte [] apiKeySecretBytes = this.signingKey.getBytes(StandardCharsets.UTF_8);
116        final Key signingKey = new SecretKeySpec(apiKeySecretBytes, this.signingAlgorithm.getJcaName()
117        return builder.signWith(signingKey, this.signingAlgorithm).compact();
118      }
119
120      /**
121       * Validates a JWT issued by this service.
122       *
123       * Throws an {@link TokenException} in the case that the token cannot be verified.
124       */
125      @Nonnull
```

```java
126    public TokenClaims validateAccessToken(@Nonnull final String accessToken) throws TokenException
127      Objects.requireNonNull(accessToken);
128      try {
129        byte [] apiKeySecretBytes = this.signingKey.getBytes(StandardCharsets.UTF_8);
130        final String base64Key = Base64.getEncoder().encodeToString(apiKeySecretBytes);
131        final Claims claims = (Claims) Jwts.parserBuilder()
132              .setSigningKey(base64Key)
133              .build()
134              .parse(accessToken)
135              .getBody();
136        final String tokenVersion = claims.get(TOKEN_VERSION_CLAIM_NAME, String.class);
137        final String tokenType = claims.get(TOKEN_TYPE_CLAIM_NAME, String.class);
138        final String actorId = claims.get(ACTOR_ID_CLAIM_NAME, String.class);
139        final String actorType = claims.get(ACTOR_TYPE_CLAIM_NAME, String.class);
140        if (tokenType != null && actorId != null && actorType != null) {
141          return new TokenClaims(
142                TokenVersion.fromNumericStringValue(tokenVersion),
143                TokenType.valueOf(tokenType),
144                ActorType.valueOf(actorType),
145                actorId,
146                claims.getExpiration() == null ? null : claims.getExpiration().getTime());
147        }
148      } catch (io.jsonwebtoken.ExpiredJwtException e) {
149        throw new TokenExpiredException("Failed to validate DataHub token. Token has expired.", e);
150      } catch (Exception e) {
151        throw new TokenException("Failed to validate DataHub token", e);
152      }
153      throw new TokenException("Failed to validate DataHub token: Found malformed or missing 'actor'
154    }
155
156    private SignatureAlgorithm validateAlgorithm(final String algorithm) {
157      if (!SUPPORTED_ALGORITHMS.contains(algorithm)) {
158        throw new UnsupportedOperationException(
159            String.format("Failed to create Token Service. Unsupported algorithm %s provided", algor
160      }
161      return SignatureAlgorithm.valueOf(algorithm);
162    }
163  }
```