JWT - Java com Boas Praticas

O **JWT (JSON Web Token)** é uma tecnologia de autenticação muito utilizada em APIs RESTful para criar autenticação segura sem a necessidade de sessões armazenadas no servidor.
O uso de JWT exige cuidado com as configurações de segurança, como expiração curta, assinatura criptográfica robusta e transporte seguro via HTTPS.

Estrutura de um JWT

Um JWT é dividido em três partes:

- **Header**: Define o algoritmo de criptografia e o tipo do token (JWT).
- **Payload**: Contém as claims, que são as informações sobre o usuário e outros dados.
- **Signature**: Resultado da assinatura do header e do payload com a chave secreta.

Formato de um JWT:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJKb2huRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Vamos criar um exemplo funcional, incluindo geração e validação de tokens, usando as boas práticas recomendadas. Usaremos Java 11+ e a biblioteca **JJWT**.

1. Dependências no Maven

Primeiro, adicione as dependências necessárias no arquivo pom.xml:

xml

```
<dependencies>
    <!-- Dependência para manipulação de JWT -->
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.11.5</version>
    </dependency>

    <!-- Dependência para trabalhar com datas e tempos -->
    <dependency>
        <groupId>org.threeten</groupId>
        <artifactId>threetenbp</artifactId>
        <version>1.5.1</version>
    </dependency>

    <!-- Adicione outras dependências como Spring Boot, se aplicável -->
</dependencies>
```

2. Código Completo para Geração e Validação de JWT

Aqui está o código completo para gerar e validar tokens JWT em Java.

2.1. Utilitário de JWT

Este utilitário encapsula a geração e validação de tokens JWT.
Ele também adiciona boas práticas, como a definição de um tempo de expiração e o uso de uma chave secreta forte.

```
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.SignatureException;
import java.util.Date;

public class JwtUtil {

    // Chave secreta para assinatura (substitua por uma chave forte e segura)
    private static final String SECRET_KEY = "mySecretKey@123456789";  // Exemplo didático

    // Método para gerar o token JWT
    public static String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)     // Define o 'subject' (geralmente o usuário)
                .setIssuedAt(new Date())    // Data de emissão do token
                .setExpiration(new Date(System.currentTimeMillis() + 3600000)) // Expira em 1 hora
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY) // Assina com o algoritmo HS256
                .compact();     // Compacta e retorna o JWT
    }

    // Método para validar o token JWT e retornar as claims
    public static Claims validateToken(String token) throws SignatureException {
        return Jwts.parser()
                .setSigningKey(SECRET_KEY)     // Chave secreta usada para assinar
                .parseClaimsJws(token)   // Verifica a assinatura e obtém as claims
                .getBody();  // Retorna o corpo do JWT (claims)
    }

    // Método para verificar se o token está expirado
    public static boolean isTokenExpired(String token) {
        Date expiration = validateToken(token).getExpiration();
        return expiration.before(new Date()); // Verifica se a data de expiração é anterior à data atual
    }
}
```

2.2. Controlador Simples (Simulação de API REST)

Aqui está um exemplo de como usar o JwtUtil em um controlador REST.
Este exemplo simula um login que gera um token e um endpoint que valida o token recebido.

```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import io.jsonwebtoken.Claims;

@RestController
@RequestMapping("/api")
public class JwtController {

    // Endpoint que simula o login e retorna o JWT
    @GetMapping("/login")
    public String login() {
        String username = "JohnDoe"; // Usuário simulado
        String token = JwtUtil.generateToken(username);
        return "JWT Token: " + token;
    }

    // Endpoint protegido que verifica o token JWT recebido
    @GetMapping("/protected")
    public String protectedResource(@RequestHeader("Authorization") String
authHeader) {
        // Remove o prefixo 'Bearer ' do cabeçalho
        String token = authHeader.replace("Bearer ", "");

        try {
            Claims claims = JwtUtil.validateToken(token); // Valida o token
            return "Acesso autorizado! Bem-vindo, " + claims.getSubject();
        } catch (Exception e) {
            return "Token inválido: " + e.getMessage();
        }
    }
}
```

2.3. Configuração de Segurança (Spring Security)

Se você estiver usando Spring Security, pode configurar o JWT como parte da autenticação adicionando filtros para interceptar o cabeçalho de autorização. Aqui está um exemplo básico:

```
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationM
anagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
```

```
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigur
erAdapter;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
                .authorizeRequests()
                .antMatchers("/api/login").permitAll( // Permite o login sem autenticação
                .antMatchers("/api/protected").authenticated() // Protege o endpoint
                .and()
                .addFilter(new JwtAuthenticationFilter(authenticationManager())); // Adiciona
o filtro JWT
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // Configuração de autenticação (pode ser baseada em banco de dados, etc.)
    }
}
```

3. Boas Práticas de Segurança

- Chave Secreta Segura: Utilize uma chave suficientemente longa e aleatória (pelo menos 256 bits) e armazene-a em um local seguro (por exemplo, em variáveis de ambiente ou serviços de gerenciamento de segredos como AWS Secrets Manager ou Azure Key Vault).
- Tempo de Expiração Curto: Tokens devem ter um tempo de expiração curto. No exemplo acima, o token expira em 1 hora. É uma boa prática usar refresh tokens para garantir que o usuário possa estender sua sessão sem reutilizar tokens expirados.
- Uso de Algoritmos Seguros: Prefira algoritmos de assinatura assimétricos, como RS256 ou ES256, em vez de HS256, especialmente em sistemas distribuídos. Isso separa a responsabilidade de geração e verificação de tokens.
- Armazenamento Seguro no Frontend: No frontend, evite armazenar tokens em localStorage ou sessionStorage, pois isso os expõe a ataques de Cross-Site Scripting (XSS). Prefira armazenar tokens em cookies com as flags HttpOnly e Secure.
- Transmissão de Tokens via HTTPS: Tokens JWT devem ser transmitidos apenas em conexões HTTPS para evitar ataques de interceptação (man-in-the-middle).

4. Testando o Código

- Inicie a aplicação Java que contém o código acima.
- Acesse o endpoint /api/login para gerar um token JWT.
- Envie uma requisição ao endpoint /api/protected com o token no cabeçalho Authorization:
  vbnet
  Copiar código
  GET /api/protected
  Authorization: Bearer <SEU_TOKEN_JWT>

5. Referências Técnicas

- RFC 7519 – JSON Web Token (JWT) - https://datatracker.ietf.org/doc/html/rfc7519
- OAuth 2.0 with JWT – JWT in OAuth: https://oauth.net/2/jwt/
- Spring Security and JWT – Integrating JWT with Spring: https://www.baeldung.com/spring-security-oauth-jwt
- JSON Web Token Best Practices – Auth0: https://auth0.com/docs/secure/tokens/json-web-tokens

Conclusão

Este exemplo demonstra como configurar e usar JWT em uma aplicação Java. Abordamos a geração, validação, e uso de JWT em APIs RESTful com boas práticas de segurança e performance.

Além disso, integramos JWT ao Spring Security, permitindo a proteção de endpoints de maneira escalável e segura. Ao seguir as boas práticas descritas, você estará pronto para implementar autenticação baseada em tokens JWT em um ambiente seguro e robusto.

EducaCiência FastCode para a comunidade.