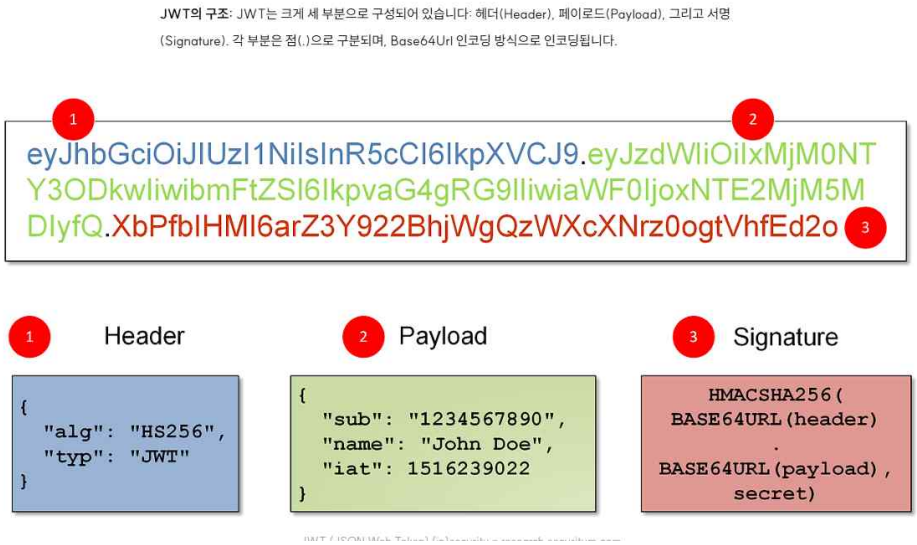
JWT (JSON Web Token)

- JWT는 간결하고 자가 수용적인 방법으로 정보를 안전하게 전달할 수 있게 해줍니다.



- **헤더(Header):** 헤더는 토큰의 타입과 사용된 알고리즘에 대한 정보를 담고 있습니다. 예를 들어, 알고리즘은 HS256이나 RS256과 같은 암호화 방식을 나타내며, 토큰 타입으로는 'JWT'가 일반적으로 사용됩니다.
- **페이로드(Payload):** 페이로드는 토큰에 담길 실제 정보(클레임)를 포함합니다. 여기에는 사용자의 정보, 토큰의 유효 기간, 발행자 등 다양한 클레임이 포함될 수 있습니다. 클레임의 종류에는 등록된 클레임, 공개 클레임, 비공개 클레임이 있으며, 각각의 목적과 사용 방법이 다릅니다.
- **서명(Signature):** 서명은 헤더와 페이로드를 합친 후, 비밀키나 공개키/개인키 쌍을 사용하여 암호화하는 과정을 거칩니다. 이 서명을 통해 JWT의 무결성과 인증이 보장됩니다. 서명 과정은 JWT가 변조되지 않았음을 증명하며, 서버는 이를 검증하여 안전한 정보 교환을 보장합니다.

작동 원리: 웹 애플리케이션에서 사용자가 로그인을 하면, 서버는 사용자의 정보를 기반으로 JWT를 생성하고, 이를 사용자에게 반환합니다. 사용자는 이후의 모든 요청에 이 JWT를 포함시켜 서버에 보냅니다. 서버는 요청을 받을 때마다 JWT의 서명을 검증하고, 유효한 경우 요청을 처리합니다. 이 과정을 통해 사용자의 인증 상태를 유지할 수 있으며, 매번 로그인하지 않아도 되는 편리함을 제공합니다.

결론: JWT는 웹 개발에서 널리 사용되는 인증 방식으로, 그 구조와 작동 원리를 이해하는 것은 안전한 웹 서비스를 제공하기 위해 매우 중요합니다. 본 글을 통해 JWT의 기본적인 개념과 구조에 대한 이해를 돕고자 하였습니다. 안전한 웹 서비스 개발을 위해 JWT를 적극 활용해보세요.

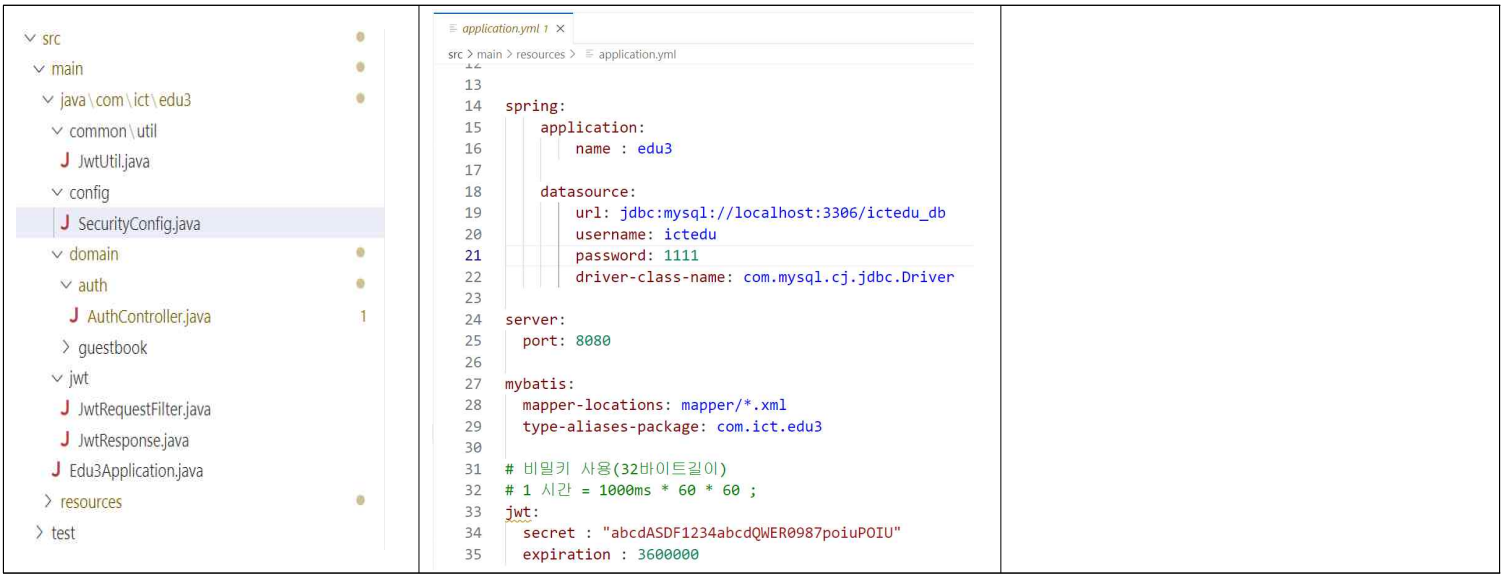
SpringBoot - JWT 생성하기

Spring 프로젝트에서 JWT를 사용하면 의존성 추가 해야 된다.
- build.gradle에서 라이브러리 추가 하자

```
// 0.12.3
// implementation 'io.jsonwebtoken:jjwt-api:0.12.3'
// implementation 'io.jsonwebtoken:jjwt-impl:0.12.3'
// implementation 'io.jsonwebtoken:jjwt-jackson:0.12.3'

// 0.11.5
implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'
implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'
```

build.gradle

```
dependencies {
    // 0.12.3
    // implementation 'io.jsonwebtoken:jjwt-api:0.12.3'
    // implementation 'io.jsonwebtoken:jjwt-impl:0.12.3'
    // implementation 'io.jsonwebtoken:jjwt-jackson:0.12.3'

    // 0.11.5
    implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
    implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'
    implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'
}
```

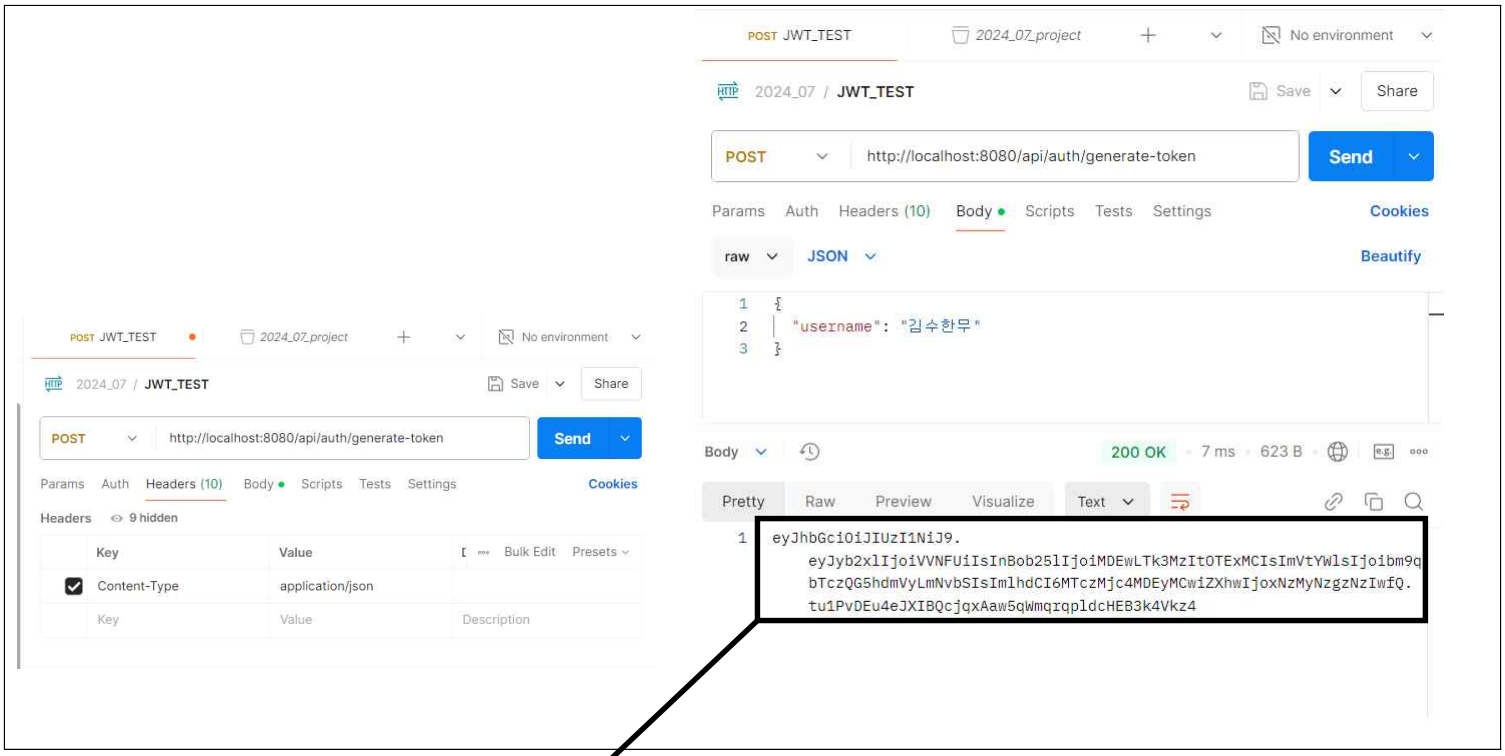



```
@RestController
@RequestMapping("/api/auth")
public class AuthAPIController {
    @Autowired
    private JwtUtil jwtUtil;

    @PostMapping("/generate-token")
    public String postMethodName(@RequestBody Map<String, String> request) {

        // 클라이어트가 uesrname라는 key에 정보를 보냈다고 가정
        String username = request.get("uesrname");

        // jwt를 생성할때 더 많은 정보를 추가 할 수있다.
        Map<String, Object> claims = new HashMap<>();
        claims.put("role", "USER");

        return jwtUtil.generateToken(username, claims);
    }

}
```

```
@Component
public class JwtUtil {
    @Value("${jwt.secret}")
    private String secret; // 비밀키

    @Value("${jwt.expiration}")
    private long expiration; // 만료시간

    // String secret키 => SecretKey
    private SecretKey getKey() {
        byte[] keyBytes = secret.getBytes(StandardCharsets.UTF_8);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    // 토큰 생성
    public String generateToken(String username, Map<String, Object> claimes) {
        // 내용을 더 추가 하고 싶으면
        // 보안 때문에 중요한 정보를 넣으면 안됨
        claimes.put("email", "nojm73@naver.com");
        claimes.put("phone", "010-9732-9110");
        return Jwts.builder()
                .setClaims(claimes)
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + expiration))
                .signWith(getKey(), SignatureAlgorithm.HS256)
                .compact();
    }
    // 클레임 이름 추출
    // 클레임에서 특정 데이터 추출
    // 모든 클레임 추출
    // 만료 여부 확인
    // 만료 시간 추출
    // 토큰 유효성 검사
}
```

```
@Configuration
public class SecurityConfig {
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .cors(cors -> cors.configurationSource(corsConfigurationSource()))
                .csrf(csrf -> csrf.disable())
                // 요청별 권한 설정
                .authorizeHttpRequests(authorize -> authorize
                        // 특정 URL에 인증없이 허용
                        .requestMatchers("/api/auth/**").permitAll()
                        // 나머지는 인증 필요
                        .anyRequest().authenticated());

        return http.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration corsConfig = new CorsConfiguration();
        // 허용할 Origin 설정
        corsConfig.setAllowedOrigins(Arrays.asList("http://localhost:8080"));
        // 허용할 http 메서드 설정
        corsConfig.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));
        // 허용할 헤더 설정
        corsConfig.setAllowedHeaders(Arrays.asList("Authorization", "Content-Type"));
        // 인증정보 허용
        corsConfig.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", corsConfig);
        return source;
    }
}
```

PostMan을 이용한 JWT 확인하기



Web에서 JWT 검증하기