

## 1. DI 추가

## Setup new Spring Boot project

Use **Spring web tool** or your development tool (**Spring Tool Suite**, Eclipse, IntelliJ) to create a Spring Boot project.

Then open **pom.xml** and add these dependencies:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

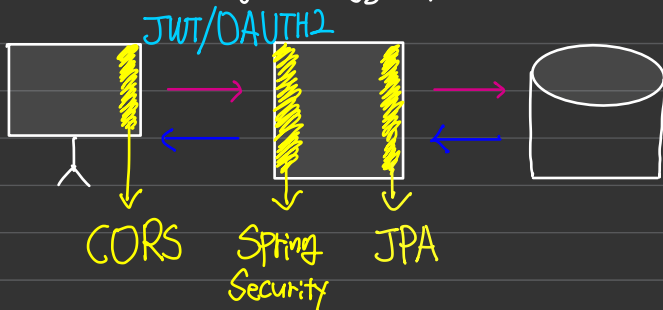
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.9.1</version>
</dependency>
```

We also need to add one more dependency.

- If you want to use **PostgreSQL**:

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
```

보안(로그인, 개인정보 등 민감사항) 위해 Spring Security + JWT 등 DI



기준방식: 시시 + 쿼리 방식 (ex 공인인증서)

⇒ SOAP: Connectionless / Stateless / XML 기반 / HTTP(프린트)

↓ 비터

## RESTful: JWT + Spring Security 방식 (ex 카카오톡방식)

- Stateful / JSON like

↳ **Representational State**

CHP S&E: '71' 25

Key: { [ k { V }, k { V }, ... ] ... }

↓  
Token

value (보안된 상태로 전송)

⇒ JWT 방식

데이터스트리밍 / Value (무엇이 되는 부분) → Payload

### \* Payload vs Parameter

서버  $\rightleftharpoons$  클라이언트

$$\begin{array}{ccc} \text{KOH} & \rightleftharpoons & \text{KOH} \\ \downarrow & & \downarrow \\ \text{J} & & \text{J} \end{array}$$

JAVA JS

J.S

크라이덴트  $\rightleftharpoons$  크라이덴트

TS

TS

이러한 시스템에서 전달

같은기종 내에서 전다

```

1 package com.example.demo.uss.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 import org.springframework.stereotype.Repository;
6
7
8 import com.example.demo.uss.domain.User;
9 import com.example.demo.uss.domain.UserDto;
10
11 interface UserCustomRepository{
12     public UserDto login(String username, String password);
13 }
14 public interface UserRepository extends JpaRepository<User, Long>,
15
16     public UserDto findByUsername(String username);
17 }

```

**Repository**

**Component**

```

1 package com.example.demo.uss.repository;
2
3 import org.springframework.stereotype.Repository;
4
5 import com.example.demo.uss.domain.User;
6 import com.example.demo.uss.domain.UserDto;
7
8 import java.util.List;
9 import java.util.Optional;
10
11 import org.springframework.data.domain.Example;
12 import org.springframework.data.domain.Page;
13 import org.springframework.data.domain.Pageable;
14 import org.springframework.data.domain.Sort;
15 import org.springframework.data.jpa.repository.JpaRepository;
16 @Repository
17 public class UserRepositoryImpl implements UserCustomRepository{
18
19     @Override
20     public UserDto login(String username, String password) {
21         return null;
22     }
23
24
25
26
27 }

```

```
1 package com.example.demo.uss.service;  
2  
3 import org.springframework.security.core.userdetails.UserDetailsService;  
4  
5 import com.example.demo.uss.domain.UserDto;  
6  
7  
8 public interface UserService extends UserDetailsService {  
9  
10     public UserDto login(String username, String password);  
11 }
```

```
1 package com.example.demo.sec.controller;
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 import org.springframework.web.bind.annotation.PostMapping;
7 import org.springframework.web.bind.annotation.RequestBody;
8 import org.springframework.web.bind.annotation.RequestMapping;
9 import org.springframework.web.bind.annotation.RestController;
10
11 import com.example.demo.uss.domain.UserDto;
12
13 @RestController
14 @RequestMapping("/auth")
15 public class AuthController {
16     @PostMapping("/login")
17     public void login(@RequestBody UserDto user) {
18         Map<String, Object> map = new HashMap<>();
19     }
20 }
21
22 }
```

Controller

```
1 package com.example.demo.sec.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6 import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
7 import org.springframework.security.config.http.SessionCreationPolicy;
8 import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
9
10 import com.example.demo.sec.domain.JwtTokenProvider;
11 import com.example.demo.sec.filter.CustomAuthenticationEntryPoint;
12 import com.example.demo.sec.filter.JwtAuthenticationFilter;
13
14 @Configuration
15 public class WebSecurityConfigurer extends WebSecurityConfigurerAdapter {
16     @Autowired JwtTokenProvider jwtTokenProvider;
17     @Override
18     protected void configure(HttpSecurity http) throws Exception {
19         http.httpBasic().disable() → SOAP 안쓰겠다
20         .csrf().disable() → URL, 타입 체크
21         .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS) → 스프링 시큐리티 세션 정책 (JWT 세션)
22         .and() → 메소드 명 ↑ Request에 대한 권한 설정 → 스프링 시큐리티에서 생성/가공의 것 사용 X
23         .authorizeRequests().permitAll()
24         .and().antMatchers("/"/login, "/"/signUp").permitAll()
25         .and().anyRequest().hasRole("USER") // 이미 이미지 업로드 모두 인증된 회원만 접근 가능
26     }
27     @ExceptionHandler({})
28     .authenticationEntryPoint(new CustomAuthenticationEntryPoint()) 인증된 도른 으로 큰
29     .addFilterBefore(new JwtAuthenticationFilter(jwtTokenProvider), UsernamePasswordAuthenticationFilter.class);
30 }
31 }
```

Adaptor 패턴 인터페이스  
A와 B가 다른 경우 A 또는 B에 둘면시켜 사용하게 하는 구조  
→ JAVA/J.S 동일 (토글은 생성해 보내줄 것이다)  
Cross-site Request Forgery → SOAP  
시드코딩 용량기 (복합의 문제사항)  
out Notation  
\* 과외  
\* 디어  
사용자가 form으로 입력한 로그인 정보  
게체 전달

```
1 package com.example.demo.sec.domain;
2
3 import java.util.Collection;
4
5 public class AbstractAuthenticationToken implements Authentication {
6     // CredentialsContainer
7     @Override
8     public String getName() {
9         // TODO Auto-generated method stub
10         return null;
11     }
12     /*
13     @Override
14     public Collection<? extends GrantedAuthority> getAuthorities() {
15         return null;
16     }
17     */
18     @Override
19     public Object getCredentials() {
20         // TODO Auto-generated method stub
21         return null;
22     }
23     @Override
24     public Object getDetails() {
25         // TODO Auto-generated method stub
26         return null;
27     }
28     @Override
29     public Object getPrincipal() {
30         // TODO Auto-generated method stub
31         return null;
32     }
33     @Override
34     public boolean isAuthenticated() {
35         // TODO Auto-generated method stub
36         return false;
37     }
38     @Override
39     public void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException {
40         // TODO Auto-generated method stub
41     }
42 }
```

```

1 package com.example.demo.sec.domain;
2
3 import java.io.Serializable;
4 import java.security.Principal;
5 import java.util.Collection;
6
7 public interface Authentication extends Principal, Serializable {
8     // 현재 사용자의 권한 목록을 가져옴
9     // Collection<? extends GrantedAuthority> getAuthorities();
10
11     // credentials(주로 비밀번호)을 가져옴
12     Object getCredentials();
13
14     Object getDetails();
15
16     // Principal 객체를 가져옴.
17     Object getPrincipal();
18
19     // 인증 여부를 가져옴
20     boolean isAuthenticated();
21
22     // 인증 여부를 설정함
23     void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException;
24 }

```

→ 인증

```

19 @RequiredArgsConstructor
20 @Component
21 public class JwtTokenProvider {
22     @Value("${spring.jwt.secret}")
23     private String SECRET_KEY;
24     private long tokenValidMilisecond = 1000L * 60 * 60; // 1시간만 토큰 유효
25     private final UserDetailsServiceImpl userDetailsService;
26     @PostConstruct
27     protected void init() {
28         SECRET_KEY = Base64.getEncoder().encodeToString(SECRET_KEY.getBytes());
29     }
30     public String createToken(String userPk, Collection<? extends GrantedAuthority> roles)
31     {
32         Claims claims = Jwts.claims().setSubject(userPk);
33         claims.put("roles", roles);
34         Date now = new Date();
35         return Jwts.builder()
36             .setClaims(claims)
37             .setIssuedAt(now)
38             .setExpiration(new Date(now.getTime() + tokenValidMilisecond))
39             .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
40             .compact();
41     }
42 }

```

▶ 토큰 생성자

② Value는 application.properties에서 조회

원래는 주입이 이루어진 후 초기화 수행

64진수의 키 값

Wild Card

Payload의 정보 한정작

빌더패턴: 방향각서 생성과도, 포함방법 불리

다른 포함 결과를 만드는 패턴

이제 연산자로

다음에서 사용 가능

JWT J = new Jwt()

대신 사용 가능

```

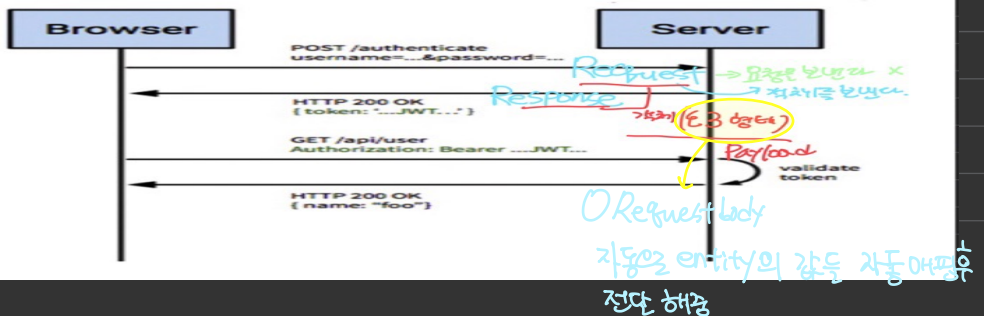
1 methods {
2     @RequestMapping(value = "/api/auth/login", method = RequestMethod.POST)
3     public ResponseEntity<String> login(@RequestBody LoginRequest request) {
4         String username = request.getUsername();
5         String password = request.getPassword();
6         UserDetails user = userDetailsService.loadUserByUsername(username);
7         if (user != null && !user.isAccountNonExpired() && !user.isAccountNonLocked() && !user.isCredentialsExpired() && user.isAccountNonDeactivated()) {
8             String token = createToken(user.getUsername(), user.getAuthorities());
9             return ResponseEntity.ok(token);
10        }
11        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid username or password");
12    }
13
14    @RequestMapping(value = "/api/auth/logout", method = RequestMethod.POST)
15    public ResponseEntity<String> logout(@RequestBody LogoutRequest request) {
16        String token = request.getToken();
17        // TODO: Implement token invalidation logic
18        return ResponseEntity.ok("Logout successful");
19    }
20 }

```

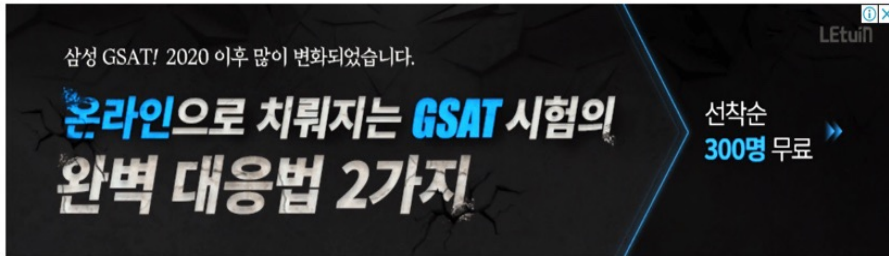
구조 분해 할당

타이머 일기

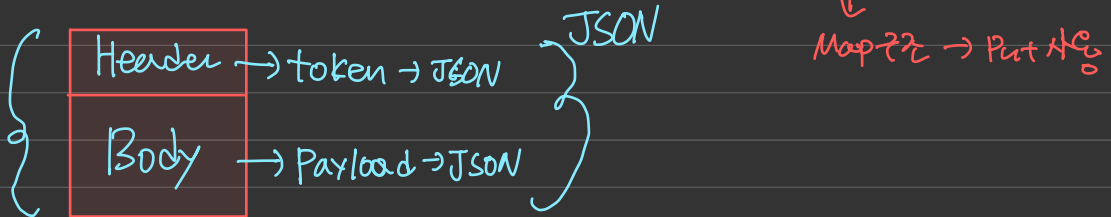
## Modern Token-Based Auth



이렇게 기존의 세션-쿠키 기반의 로그인인 아니라 JWT같은 토큰 기반의 로그인을 하게 되면 세션이 유지되지 않는 다중 서버 환경에서도 로그인을 유지할 수 있게 되고 한 번의 로그인으로 유저정보를 공유하는 여러 도메인에서 사용할 수 있다는 장점이 있습니다.



이 때 회원을 구분할 수 있는 정보가 담기는 곳이 바로 JWT의 payload 부분이고 이곳에 담기는 정보의 한 '조각'을 Claim 이라고 합니다. Claim은 name / value 한 쌍으로 이루어져 있으며 당연히 여러개의 Claim들을 넣을 수 있습니다.



[Header].[Payload].(Verify Signature)

3가지 정보로 '로' 인코딩하여 전달한다.

JWT 공식 사이트를 통해 JWT를 생성 및 검증할 수 있다.

### Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmVziIjoiImlkZW50LnBkaWwifQ.eyJpcyIjoiZm9udCJ9
```

*(Handwritten: 지저분)*

### Decoded

HEADER	PAYLOAD
{ "alg": "HS256", "typ": "JWT" }	
CLAIMS	{ "iss": "1234567890", "name": "Admin User", "iat": 1516239822 }
SIGNATURE	

*(Handwritten: @Req..Body)*

**Header:** JWT 토큰에 유형이나 사용된 해시 알고리즘의 정보가 들어간다.

**Payload:** 클레임(claims)은 로컬화 된 데이터가 들어가는 곳이다. → DTO, Entity, List  
*(Handwritten: 이진)*

토큰 여기에는 iss, sub, aud, exp, nbf, iat, jti 와 같은 기본 정보가 들어간다.

- 1. 등록된 클레임 (Registered claims)  
이미 정의된 클레임들로 무조건 따라야 하는 것은 아니지만 권장한다.
- iss (issuer, 토론표 명칭), exp (expiration time, 토론표 만료시간), sub (subject, 토론표 제목), aud (audience, 토론표 대상자)와 같은 클레임들이 포함된다.
- 클레임의 이름은 compact를 취해 3글자로 사용한다.
- 2. 공개된 클레임 (Public claims)  
JWT를 사용하는 사람들에 의해 정의되는 클레임이다.
- 클레임 충돌을 피하기 위해서 IANA JSON Web Token Registry 에 정의하거나 URI 형식으로 정의해야 한다.
- 3. 비공개 클레임 (Private claims)