



## 20. Security& JWT



# content

---

- ▶ 프로젝트 생성
- ▶ Entity와 Repository 작성
- ▶ Service 작성
- ▶ Security
- ▶ JWT
- ▶ Controller



# 1. 프로젝트 생성

새 프로젝트

제너레이터

Maven 원형

Jakarta EE

Spring Initializr

JavaFX

Quarkus

Micronaut

Ktor

Kotlin Multiplatform

Compose Multiplatform

HTML

React

Express

Angular CLI

IDE 플러그인

Android

Vue.js

Vite

서버 URL: start.spring.io

이름: ShopBackend

위치: D:\project\2024\_springboot  
프로젝트이(가) 다음에 생성됩니다. D:\project\2024\_springboot\ShopBackend

☐ Git 저장소 생성

언어: Java Kotlin Groovy

타입: Gradle - Groovy Gradle - Kotlin Maven

그룹: com.example

아티팩트: ShopBackend

패키지 이름: com.example.shopbackend

JDK: 17 Oracle OpenJDK version 17.0.6

Java: 17

패키지 생성: Jar War

다음(N) 취소

새 프로젝트

Spring Boot: 3.3.0

☒ JDK 및 Maven 라이브러리의 사전 빌드 공유 색인 다운로드

종속성:  
Q+ 검색

> Developer Tools

> Web

> Template Engines

> Security

> SQL

> NoSQL

> Messaging

> I/O

> Ops

> Observability

> Testing

> Spring Cloud

> Spring Cloud Config

> Spring Cloud Discovery

> Spring Cloud Routing

> Spring Cloud Circuit Breaker

SQL

추가된 종속성:

Lombok

Spring Boot DevTools

Spring Web

Spring Security

Spring Data JPA

MySQL Driver

이전(P) 생성(C) 취소

# 1. 프로젝트 생성

---

## ▶ MySQL 워크벤치 새 유저 만들기

- ▶ Sever > Users and Privileges
- ▶ add Account로 새로운 유저 추가
- ▶ Administrative Role 에서 권한 설정
- ▶ 새 connection 추가

## ▶ 새 스키마 작성

- ▶ 스키마 명 : shopdb
- ▶ encoding 설정 : utf8



# 1. 프로젝트 생성

---

## ▶ application 설정

```
server.port=8090
#server.servlet.context-path=/mysite

# MYSQL DB
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/shopdb?useSSL=false&serverTimezone=Asia/Seoul&characterEncoding=UTF-8
spring.datasource.username=pgm
spring.datasource.password=1234

# JPA table ddl auto
spring.jpa.hibernate.ddl-auto=update

# Console sql show
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

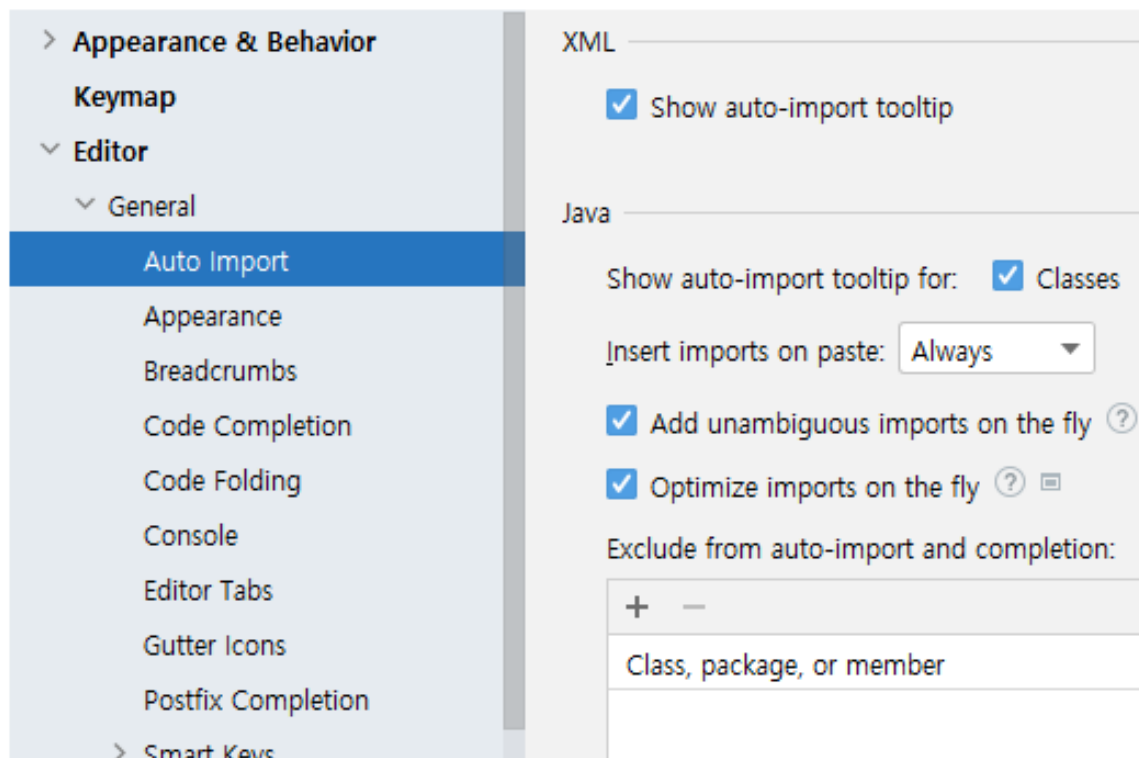


# 1. 프로젝트 생성

---

## ▶ 자동 import 설정

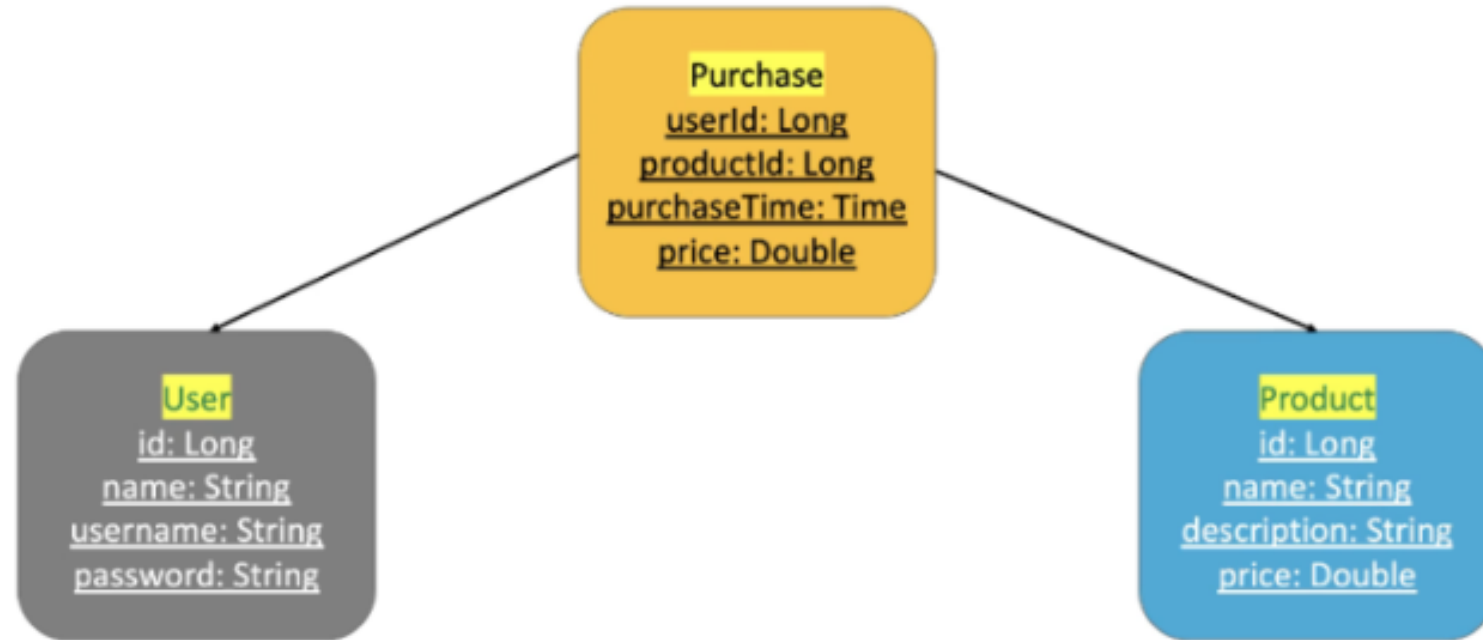
- ▶ 파일 > 세팅( Ctrl + Alt + S)



## 2. Entity와 Repository 작성

---

### ▶ ER-Diagram



## 2. Entity와 Repository 작성

---

### ▶ User Entity

- ▶ model 패키지에 작성

```
@Data
@Entity
@Table(name="users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "username", unique = true, nullable = false, length = 100)
    private String username;

    @Column(name = "password", nullable = false)
    private String password;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "create_time", nullable = false)
    private LocalDateTime createTime;

    @Enumerated(EnumType.STRING)
    @Column(name = "role", nullable = false)
    private Role role;
}
```





## 2. Entity와 Repository 작성

### ▶ Role Enum

- ▶ model 패키지에 작성

```
public enum Role {  
    1개 사용 위치  
    USER, ADMIN;  
}
```

**@Enumerated(EnumType.ORDINAL)**

**@Enumerated(EnumType.STRING)**

ordinal

-----  
ID | role |  
-----

1 | 0

2 | 1

3 | 0

4 | 0

string

-----  
ID | role |  
-----

1 | USER

2 | ADMIN

3 | USER

4 | USER

## 2. Entity와 Repository 작성

### ▶ Product Entity

```
@Data
@Entity
@Table(name="prdduct")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "description", nullable = false)
    private String description;

    @Column(name = "price", nullable = false)
    private Integer price;

    @CreationTimestamp
    @Column(name = "create_time", nullable = false)
    private LocalDateTime createTime;
}
```

## 2. Entity와 Repository 작성

### ▶ Purchase Entity

```
@Data
@Entity
@Table(name="purchase")
public class Purchase {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "user_id", nullable = false)
    private Long userId;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", referencedColumnName = "id", insertable = false, updatable = false)
    //It is only for foreign key.
    private User user;

    @Column(name = "product_id", nullable = false)
    private Long productId;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "product_id", referencedColumnName = "id", insertable = false, updatable = false)
    private Product product;

    @Column(name = "quantity", nullable = false)
    private Integer quantity;

    @Column(name = "purchase_time", nullable = false)
    private LocalDateTime purchaseTime;
}
```

## 2. Entity와 Repository 작성

---

### ▶ 즉시 로딩(**EAGER**)과 지연 로딩(**LAZY**)

- ▶ Purchase를 조회할 때 User와 Product도 함께 조회 해야 할까?
- ▶ 비즈니스 로직에서 단순히 멤버 로직만 사용하는데 함께 조회하면, 트래픽 증가
- ▶ JPA는 이 문제를 지연로딩 LAZY를 사용해서 **필요시 조회**하는 방법으로 해결.

### ▶ insertable = false, updatable = false

- ▶ 입력시 이 열은 제외, 업데이트시 이 열은 제외
- ▶ 이 엔티티에서 외래키 부분(Product 와 User) 를 Purchase 를 입력하거나 업데이트 할때 포함하지 않는다.



## 2. Entity와 Repository 작성

---

### ▶ UserRepository

- ▶ repository 패키지에 작성

2개 사용 위치

```
public interface UserRepository extends JpaRepository<User, Long> {
```

1개 사용 위치

```
    Optional<User> findByUsername(String username);
```

1개 사용 위치

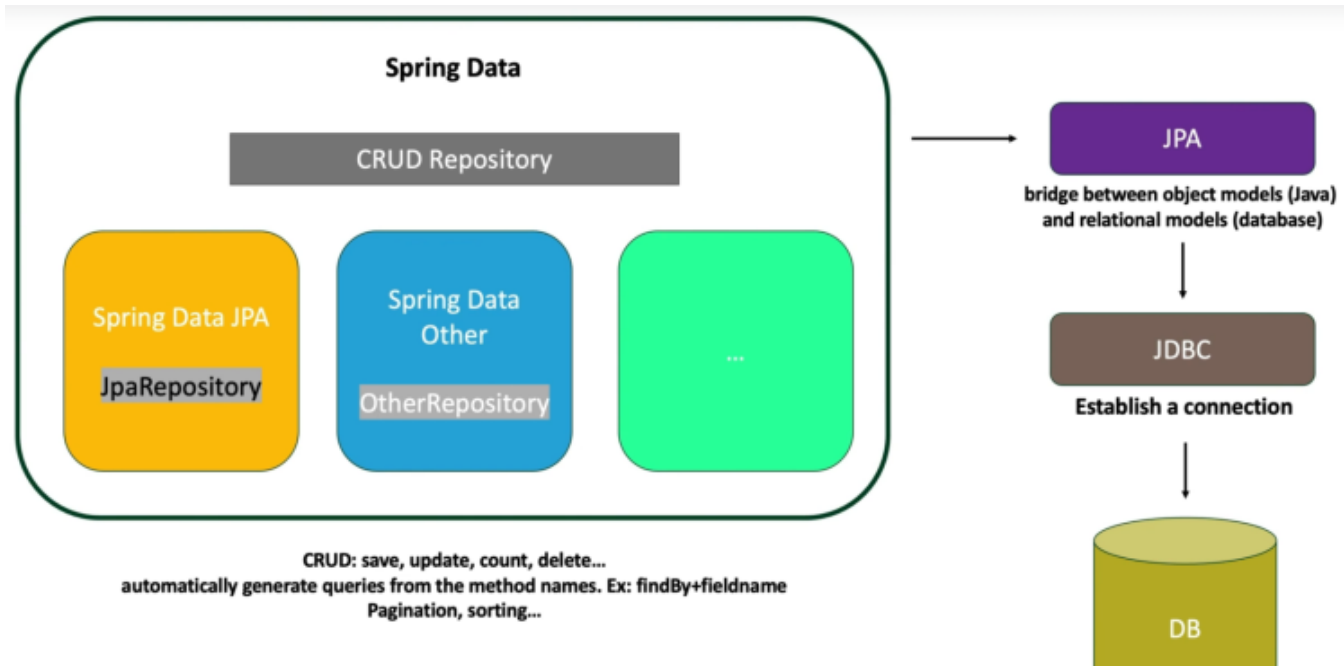
```
    @Modifying  
    @Query("update User set role=:role where username=:username")  
    void updateUserRole(@Param("username") String username, @Param("role") Role role);  
}
```



## 2. Entity와 Repository 작성

### ▶ UserRepository

#### ▶ jpa 데이터 처리 방식



<Class> findBy + fieldName...

User findByName(String name);

User findByCreateTimeAfter(LocalDate threshold)

## 2. Entity와 Repository 작성

---

### ▶ UserRepository

#### ▶ JPQL이란?

- ▶ JPQL은 엔티티 객체를 조회하는 객체지향 쿼리, 즉 테이블을 대상으로 쿼리하는 것이 아니라 엔티티 객체를 대상으로 쿼리
- ▶ 문법은 SQL과 유사하며 간결하다
- ▶ JPQL은 결국 SQL로 변환된다.

#### ▶ JPA 방식이 아닌 JPQL 로 쿼리를 작성 예

##### ▶ Select Query

```
@Query("Select c from Class where c.fieldName = :param")  
<Class> findByQueryAnnotation(@Param String params...);
```



## 2. Entity와 Repository 작성

---

### ▶ UserRepository

- ▶ INSERT UPDATE DELETE 쿼리작성시 @Modifying을 붙임

```
@Modifying -> add modifying annotation on update or delete queries  
@Query("Update or Delete ... where c.fieldName = :param")  
<Class> updateOrDelete (@Param String params...);
```

### ▶ JPQL주의점.

- ▶ **대소문자 구분** : 엔티티와 속성은 대소문자를 구분. (단 SELECT, FROM, WHERE 같은 JPQL 키워드는 대소문자를 구분하지 않는다).
- ▶ **엔티티 이름**: JPQL에서 사용한 Member는 테이블명이 아니라 엔티티 클래스 이름이다.
- ▶ **이름 파라미터 (:속성명)** : 파라미터를 이름으로 구분할 때., 이름 기준 파라미터는 앞에 :를 사용.





## 2. Entity와 Repository 작성

---

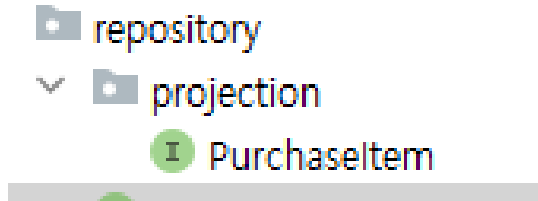
### ▶ ProductRepository

```
public interface ProductRepository extends JpaRepository<Product, Long> {  
  
}
```



## 2. Entity와 Repository 작성

### PurchaseItem 작성



6개 사용 위치

```
public interface PurchaseItem {  
    0개의 사용위치  
    String getName();  
    0개의 사용위치  
    Integer getQuantity();  
    0개의 사용위치  
    LocalDateTime getPurchaseTime();  
}
```

**프로젝션 (Projection)**이란 SELECT 절에서 조회할 대상을 지정하는 것으로  
여기서 제품의 이름, 가격, 구매시간 만을 가져오는 프로젝션을 작성.



## 2. Entity와 Repository 작성

---

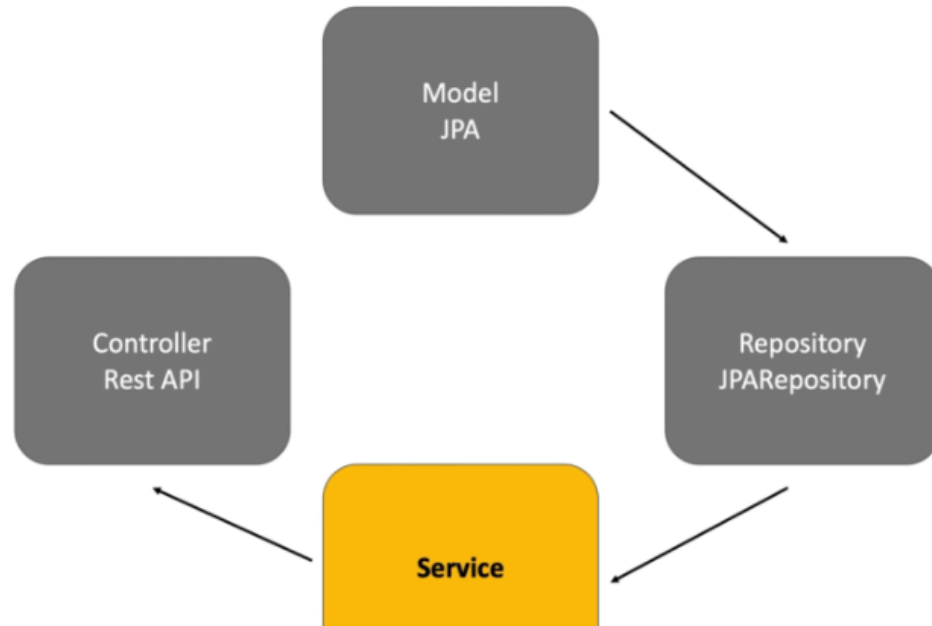
### ▶ PurchaseRepository

```
public interface PurchaseRepository extends JpaRepository<Purchase, Long> {  
    1개 사용 위치  
    @Query("select " +  
        "prd.name as name, pur.quantity as quantity, pur.purchaseTime as purchaseTime " +  
        "from Purchase pur left join Product prd on prd.id = pur.productId " +  
        "where pur.userId = :userId" )  
  
    List<PurchaseItem> findAllPurchasesOfUser(@Param("userId") Long userId);  
}
```



### 3. Service 작성

---



1. 모델 엔티티 설계 ( 테이블 )
2. 리포지토리 작성 ( DAO )
3. 서비스 작성(비즈니스 로직)
4. 컨트롤러에서 서비스 호출

### 3. Service 작성

---

- ▶ User 서비스

- ▶ 유저 서비스에 다음과 같은 로직이 필요

saveUser

findByUsername

changeRole

findAllUsers



### 3. Service 작성

#### ▶ User 서비스



```
public interface UserService {
    0개의 사용위치 1개 구현
    User saveUser(User user);
    1개 사용 위치 1개 구현
    Optional<User> findByUsername(String username);
    0개의 사용위치 1개 구현
    void changeRole(Role newRole, String username);
}
```

패스워드 객체를 빈 등록하기 위해 메인 클래에 빈 등록

```
@Bean
public PasswordEncoder passwordEncoder(){
    return new BCryptPasswordEncoder();
}
```

```
@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService{
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;

    0개의 사용위치
    public User saveUser(User user){
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        user.setRole(Role.USER);
        user.setCreateTime(LocalDateTime.now());
        return userRepository.save(user);
    }

    1개 사용 위치
    @Override
    public Optional<User> findByUsername(String username) {
        return userRepository.findByUsername(username);
    }

    0개의 사용위치
    @Override
    @Transactional
    public void changeRole(Role newRole, String username) {
        userRepository.updateUserRole(username, newRole);
    }
}
```

### 3. Service 작성

#### ▶ ProductService

```
public interface ProductService {  
    0개의 사용위치 1개 구현  
    Product saveProduct(Product product);  
    0개의 사용위치 1개 구현  
    void deleteProduct(Long id);  
    0개의 사용위치 1개 구현  
    List<Product> findAllProducts();  
}
```

```
@Service  
@RequiredArgsConstructor  
public class ProductServiceImpl implements ProductService {  
    private final ProductRepository productRepository;  
    0개의 사용위치  
    @Override  
    public Product saveProduct(Product product) {  
        product.setCreateTime(LocalDateTime.now());  
        return productRepository.save(product);  
    }  
    0개의 사용위치  
    @Override  
    public void deleteProduct(Long id) {  
        productRepository.deleteById(id);  
    }  
    0개의 사용위치  
    @Override  
    public List<Product> findAllProducts() { return productRepository.findAll(); }  
}
```

### 3. Service 작성

#### ▶ PurchaseService

```
public interface PurchaseService {  
    1개 사용 위치 1개 구현  
    Purchase savePurchase(Purchase purchase);  
    1개 사용 위치 1개 구현  
    List<PurchaseItem> findPurchaseItemsOfUser(String username);  
}
```

```
@Service  
@RequiredArgsConstructor  
public class PurchaseServiceImpl implements PurchaseService {  
    private final PurchaseRepository purchaseRepository;  
    private final UserRepository userRepository;  
  
    1개 사용 위치  
    @Override  
    public Purchase savePurchase(Purchase purchase) {  
        purchase.setPurchaseTime(LocalDateTime.now());  
        return purchaseRepository.save(purchase);  
    }  
  
    1개 사용 위치  
    @Override  
    public List<PurchaseItem> findPurchaseItemsOfUser(String username) {  
        System.out.println("service~~~~~"+username);  
        User user=userRepository.findByUsername(username).orElseThrow();  
  
        return purchaseRepository.findAllPurchasesOfUser(user.getId());  
    }  
}
```



## 4. Security와 JWT

---

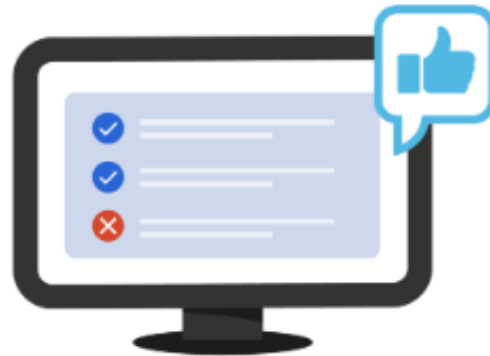
### ▶ 인증(Authentication) vs 인가(Authorization)

#### Authentication



Confirms users  
are who they say they are.

#### Authorization



Gives users permission  
to access a resource.

## 4. Security

---

### ▶ 인증과 인가(허가)의 차이점

- ▶ 인증 : 유저가 자원에 접근할 수 있는지를 확인을 받는 과정(예 : 로그인)
- ▶ 인가 : 유저가 자원에 접근할 때, 어떤 자원에 접근이 가능한지 확인하는 과정(예 : Security에서 사용자에게 따른 접근 권한 설정)



### ▶ 인증과 인가의 필요성

- ▶ Private Api는 물론 Public Api도 기본적인 인증과 인가가 요구된다.
- ▶ 인증(Authentication)은 서비스를 누가 사용했는지 추적이 가능하므로, 제3의 타인에 대한 서비스 정보 노출을 막을 수 있다.
- ▶ 인가(Authorization)는 인증된 사용자에게 대한 권한을 확인함으로써 A가 쓴 글을 B가 수정, 삭제 등을 할 수 없도록 막을 수 있다.



## 4. Security

---

### ▶ 인증방법

#### ▶ HTTP 헤더 요청 방식

- ▶ 인증을 받을 사용자 계정 정보를 http 요청에 담아 보내는 방식
- ▶ 데이터를 요청할 때마다 사용자 정보를 계속해서 보내는 것이므로, 가장 보안이 낮은 인증 방식
- ▶ 해커가 중간에 http 요청을 가로챌다면 사용자의 계정정보를 알아낼 수 있기 때문에 실제 서비스에서 거의 사용되지 않는 방식

#### ▶ Session/ Cookie 방식

- ▶ 사용자의 계정 정보를 매번 모든 요청에 넣어서 보내는 것은 보안에도 취약하고 비효율적이므로 이를 보완하기 위해 나온 인증 방법
- ▶ session/cookie 방식은 세션 저장소를 필요, 사용자가 로그인했을 때 사용자의 정보를 저장하고 열쇠가 되는 session id값을 만들어서 서버에서 관리를 하며, 웹 브라우저가 서버에 접속하여 종료할 때까지 인증 상태를 유지
- ▶ 사용자 정보를 서버에 저장하므로 보안이 비교적 안전하지만, 사용자가 많아질수록 서버의 메모리를 많이 차지하여 서버에 과부하를 주어 성능 저하를 야기

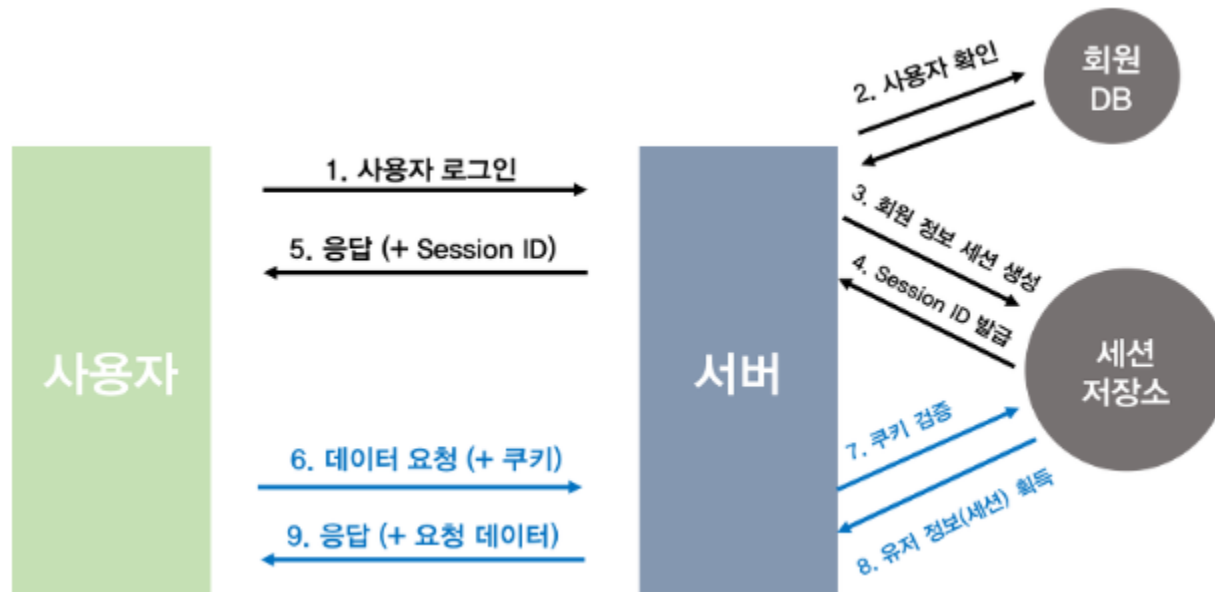
#### ▶ 토큰 인증 방식

- ▶ session이 서버의 메모리를 차지하는 단점 때문에 이를 해결하기 위한 방법
- ▶ token 방식을 사용하면 서버가 정보를 저장하지 않게 되어 사용자가 많아져도 서버의 과부하나 성능 저하가 생기지 않는다.



## 4. Security

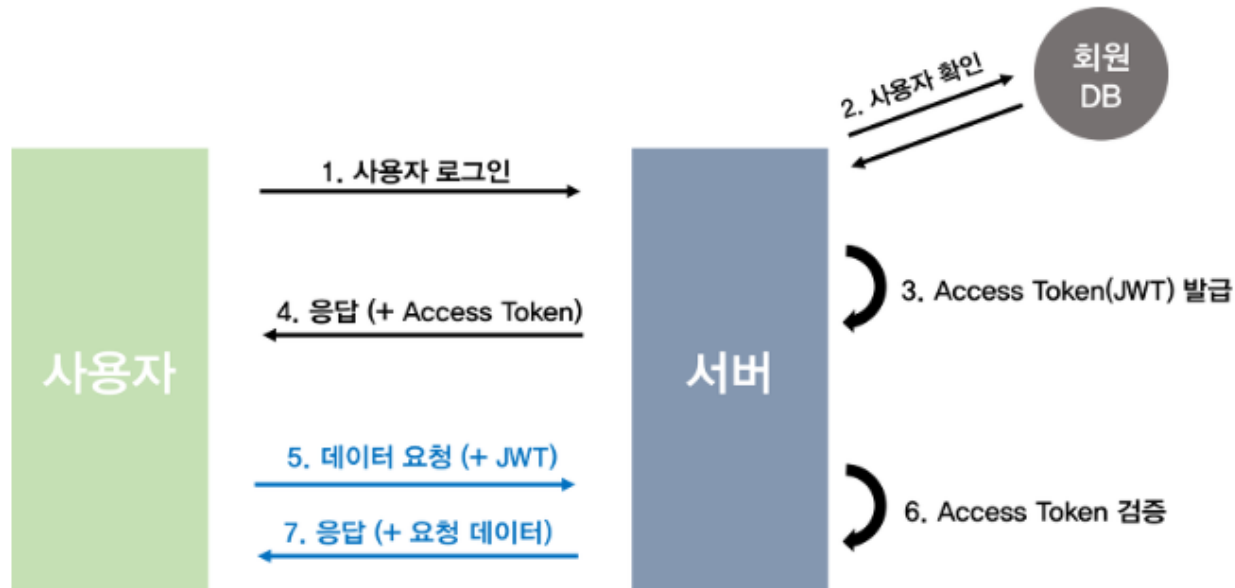
### ▶ Session/ Cookie 방식



- ① 사용자가 로그인을 하면 서버에서 계정 정보를 확인한 후, 사용자에게 고유한 id 값을 부여하여 세션 저장소에 저장한다.
- ② 세션 저장소에 저장한 id 값에 연결되는 세션 id를 발행한다. 사용자는 서버에서 session id를 전달받아 쿠키에 저장한다.
- ③ 이후 인증이 필요한 요청마다 쿠키를 http 헤더에 실어서 보낸다.
- ④ 서버에서는 쿠키를 받아 session id를 세션 저장소에서 대조를 하여 검증한 후에 대응되는 유저 정보를 가져온다.
- ⑤ 인증이 완료되고 서버는 사용자에게 맞는 데이터를 보내준다.

## 4. Security

### ▶ 토큰 인증 방식



- ① 사용자가 로그인을 하면 서버에서 사용자를 확인하고, 사용자의 고유한 ID값을 부여한 후, 기타 정보와 함께 payload에 담는다.
- ② secret key를 이용하여 access token을 발급한다.
- ③ 인증된 사용자는 응답에서 액세스 토큰(Access Token)을 받아 저장한다.
- ④ 이 access token은 이후 http 요청마다 헤더에 담아서 전달하므로 서버는 요청한 사용자를 식별할 수 있다.
- ⑤ 서버에서는 해당 토큰의 Verify Signature를 Secret key로 복호화하고, 검증이 완료되면 payload 디코딩하여 사용자의 id에 맞는 데이터를 가져온다.

토큰은 그 자체로 유저 정보를 가지고 있으므로, 인증 정보를 서버가 저장하지 않아도 되어 애플리케이션을 확장할 수 있다.  
stateless 서버를 만드는데 큰 강점이다  
또한 한번 발급된 토큰은 탈취가 되어도 유효기간이 만료할 때까지 계속 사용이 가능  
토큰이 탈취되어 악의적으로 이용된다면 토큰의 유효기간이 만료될때까지 기다리 것 말고는 할 수 있는 것이 없기 때문에  
보안에 유의해야 한다.

## 4. Security

▼ security

- CustomUserDetailsService
- UserPrincipal

### ▶ UserDetailsService 와 UserDetails 구현

```
@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {
    private final UserService userService;

    0개의 사용위치
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user=userService.findByUsername(username).orElseThrow(()->
            new UsernameNotFoundException(username)
        );
        Set<GrantedAuthority> authorities= Set.of(SecurityUtils.convertToAuthority(user.getRole().name()));
        return null;
    }
}
```

## 4. Security

---

- ▶ UserDetailsService 와 UserDetails 구현
  - ▶ SecurityUtils



```
public class SecurityUtils {  
    2개 사용 위치  
    public static final String ROLE_PREFIX = "ROLE_";  
  
    1개 사용 위치  
    public static SimpleGrantedAuthority convertToAuthority(String role){  
        String formatRole=role.startsWith(ROLE_PREFIX)? role: ROLE_PREFIX + role;  
        return new SimpleGrantedAuthority(formatRole);  
    }  
}
```



## 4. Security

### ▶ UserDetailsService 와 UserDetails 구현

```
@Getter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class UserPrinciple implements UserDetails {

    private Long id;
    private String username;
    transient private String password;
    transient private User user;
    private Set<GrantedAuthority> authorities;

    1개 사용 위치
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    @Override
    public String getPassword() {
        return password;
    }
}
```

▼ security

- CustomUserDetailsService
- UserPrinciple

```
@Override
public String getUsername() {
    return username;
}

0개의 사용위치
@Override
public boolean isAccountNonExpired() {
    return true;
}

0개의 사용위치
@Override
public boolean isAccountNonLocked() {
    return true;
}

0개의 사용위치
@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}
}
```



## 4. Security

### ▶ UserDetails 란

- ▶ Spring Security에서 사용자의 정보를 담는 인터페이스. 사용자 정보를 불러오기 위해서 구현

| 메소드                       | 리턴 타입                                  | 설명   | 기본값             |
|---------------------------|--|--|-----------------|
| getAuthorities()          | Collection<? extends GrantedAuthority> | 계정의 권한 목록을 리턴                                  |                 |
| getPassword()             | String                                 | 계정의 비밀번호를 리턴                                   |                 |
| getUsername()             | String                                 | 계정의 고유한 값을 리턴<br>( ex : DB PK값, 중복이 없는 이메일 값 ) |                 |
| isAccountNonExpired()     | boolean                                | 계정의 만료 여부 리턴                                   | true ( 만료 안됨 )  |
| isAccountNonLocked()      | boolean                                | 계정의 잠김 여부 리턴                                   | true ( 잠기지 않음 ) |
| isCredentialsNonExpired() | boolean                                | 비밀번호 만료 여부 리턴                                  | true ( 만료 안됨 )  |
| isEnabled()               | boolean                                | 계정의 활성화 여부 리턴                                  | true ( 활성화 됨 )  |

## 4. Security

### ▶ UserDetailsService 란?

- ▶ Spring Security에서 유저의 정보를 가져오는 인터페이스

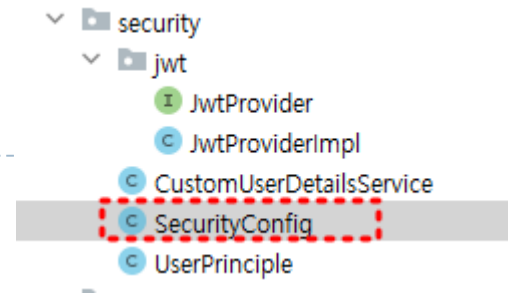
| 메소드                | 리턴 타입       | 설명                           |
|--------------------|-------------|------------------------------|
| loadUserByUsername | UserDetails | 유저의 정보를 불러와서 UserDetails로 리턴 |

loadUserByUsername 메소드의 마지막 부분  
return null 부분을 오른쪽과 같이 수정

```
return UserPrinciple.builder()  
    .user(user)  
    .username(user.getUsername())  
    .password(user.getPassword())  
    .authorities(authorities)  
    .build();
```

## 4. Security

### ▶ @EnableWebSecurity 시큐리티 설정



```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig {
    private final CustomUserDetailsService userDetailsService;

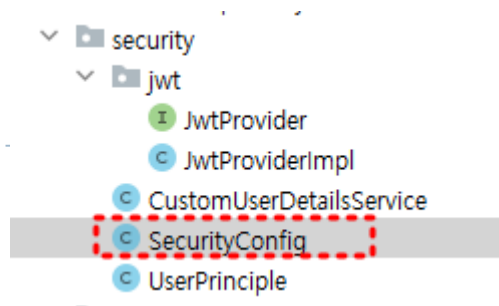
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration)
        throws Exception{
        return authenticationConfiguration.getAuthenticationManager();
    }
}
```

@Configuration은 스프링의 환경설정 파일을 의미하는 애너테이션  
@EnableWebSecurity는 모든 요청 URL이 스프링 시큐리티의 제어를 받도록 만드는 애너테이션

AuthenticationManager : 스프링 시큐리티의 인증을 담당  
AuthenticationManager 빈 생성시 **스프링의 내부 동작**으로 인해 위에서 작성한 customUserDetailsService와 PasswordEncoder가 자동으로 설정되어 사용자가 로그인 하면 유저이름으로 customUserDetailsService를 이용해 유저를 찾은 후 암호화된 패스워드와 유저가 입력한 패스워드를 비교(이때 패스워드 엔코더 객체필요) 하여 인증을 한다.

## 4. Security

- ▶ SecurityFilterChain 빈
  - ▶ 스프링 시큐리티의 세부 설정



0개의 사용위치

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
    http
        .csrf(AbstractHttpConfigurer::disable)
        .cors(httpSecurityCorsConfigurer -> corsConfigurationSource())
        .sessionManagement((session)->session
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests(authz -> authz
            .requestMatchers("/api/authentication/**").permitAll()
            .anyRequest().authenticated()
        );
    return http.build();
}
```

## 4. Security

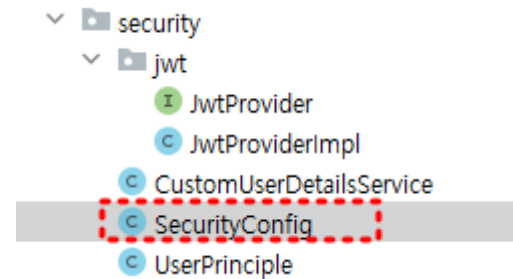
### ▶ CORS 설정

- ▶ 모든 요청시 Cross-Origin 을 허가

@Bean

```
CorsConfigurationSource corsConfigurationSource() {  
    CorsConfiguration configuration = new CorsConfiguration();  
    configuration.setAllowedOrigins(Arrays.asList("*"));  
    configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));  
    configuration.setAllowedHeaders(Arrays.asList("*"));  
    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
    source.registerCorsConfiguration(pattern: "**", configuration);  
    return source;  
}
```

브라우저(화면)을 보여주는 프론트 리액트가 :3000 에서 실행, 백엔드 스프링부트 : 8080에서 요청자료를 보내 주면 동일한 출처가 아니므로 CORS 오류가 발생해서 요청을 거부하게 된다.  
이때 서버사이드 스프링부트에서 시큐리티 설정으로 응답 헤더에 "Access-Control-Allow-Origin" 헤더를 추가 하여 다른 도메인에서 접근할 수 있도록 한다.



## 5. JWT

---

### ▶ JSON Web Token (JWT)

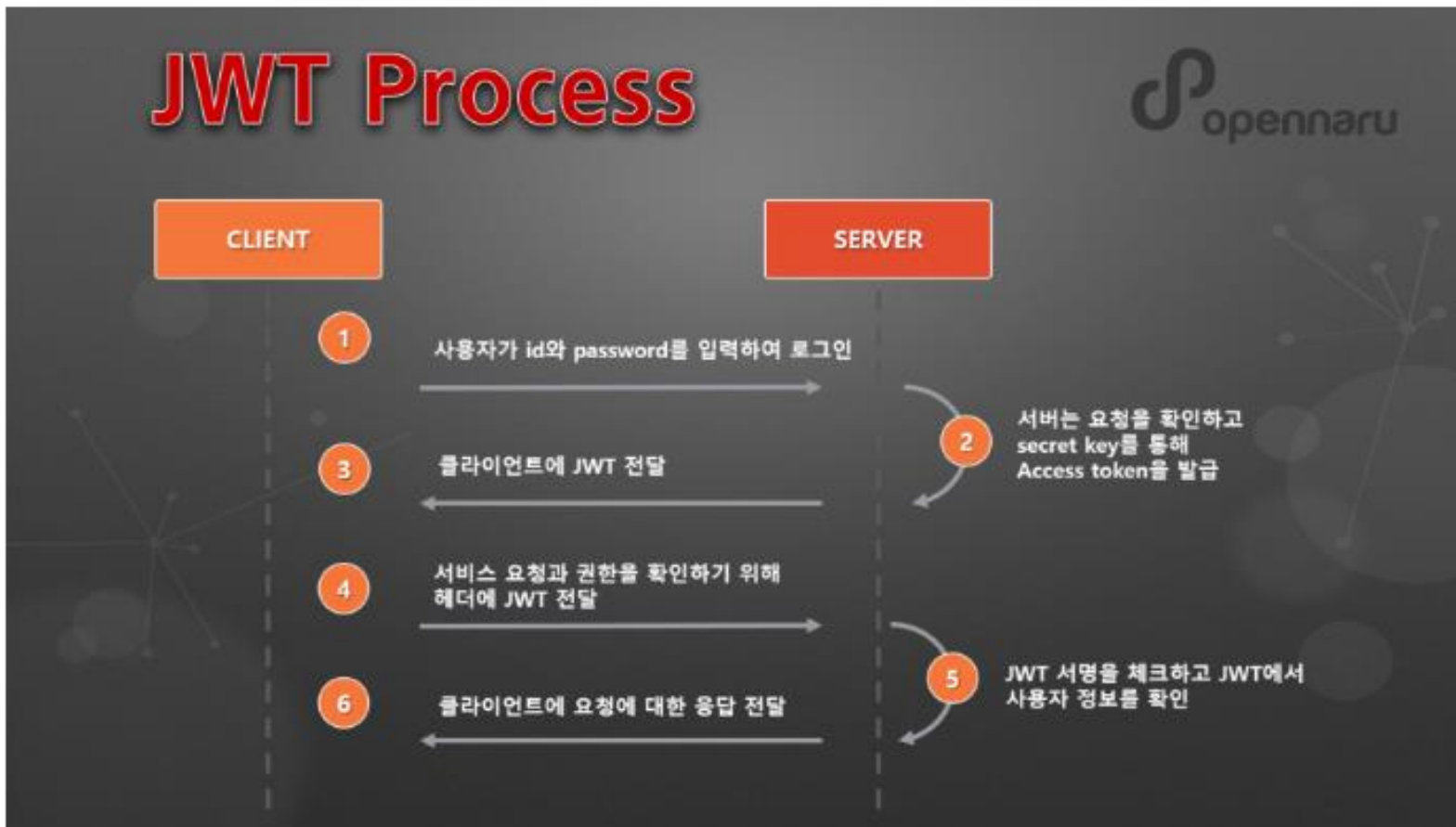
#### ▶ JWT 란?

- ▶ 웹에서 사용되는 JSON 형식의 토큰에 대한 표준 규격.
- ▶ 주로 사용자의 인증(authentication) 또는 인가(authorization) 정보를 서버와 클라이언트 간에 안전하게 주고 받기 위해서 사용
- ▶ JWT 토큰 웹에서 보통 Authorization HTTP 헤더를 Bearer <토큰>으로 설정하여 클라이언트에서 서버로 전송
- ▶ 서버에서는 토큰에 포함되어 있는 서명(signature) 정보를 통해서 위변조 여부를 빠르게 검증할 수 있음
- ▶ JWT의 주요한 이점은 사용자 인증에 필요한 모든 정보는 토큰 자체에 포함하기 때문에 별도의 인증 저장소가 필요 없음



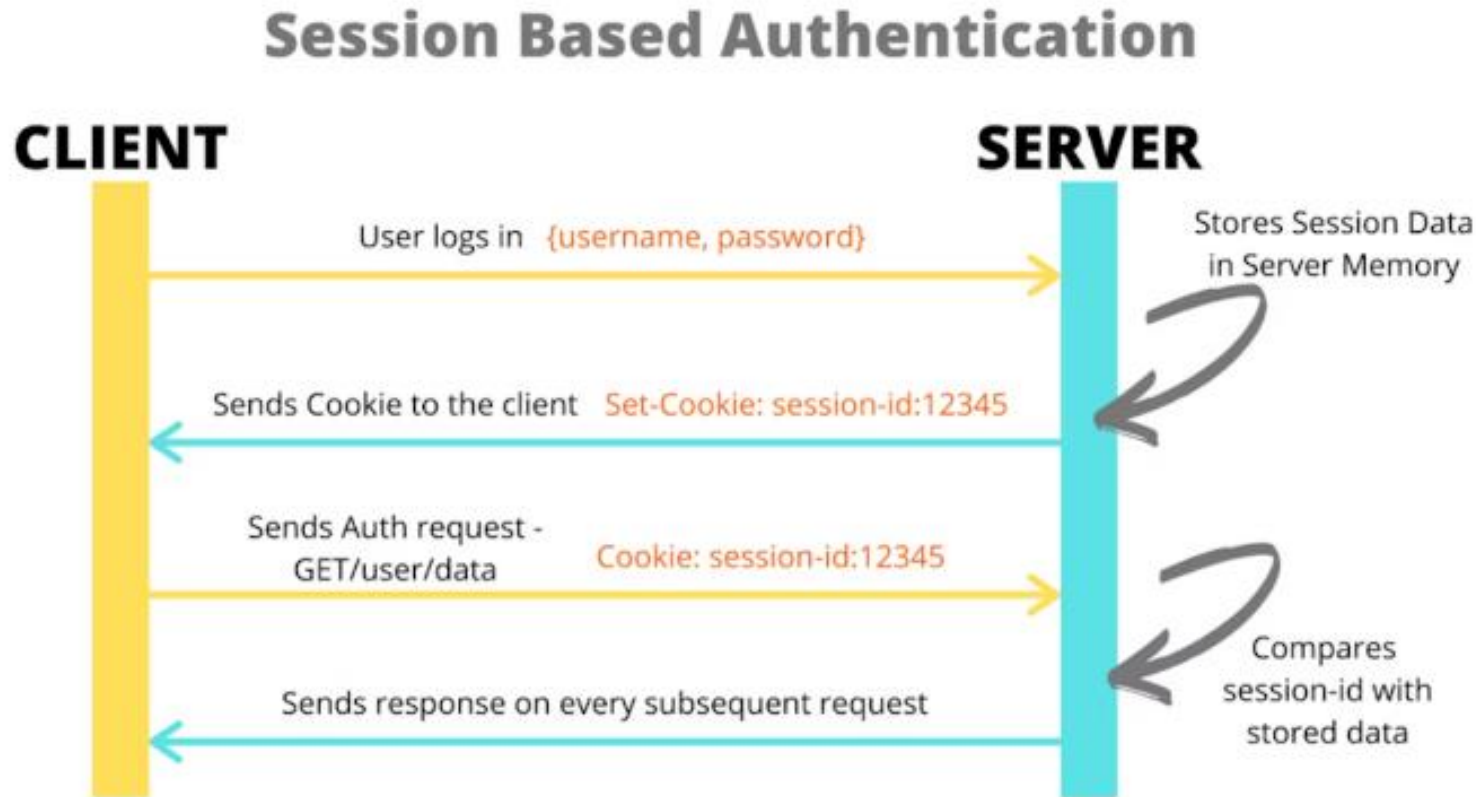
## 4. Security와 JWT

- ▶ JSON Web Token (JWT)
  - ▶ 회원인증 프로세스



## 5. JWT

### ▶ 세션 인증 vs JWT 토큰 인증



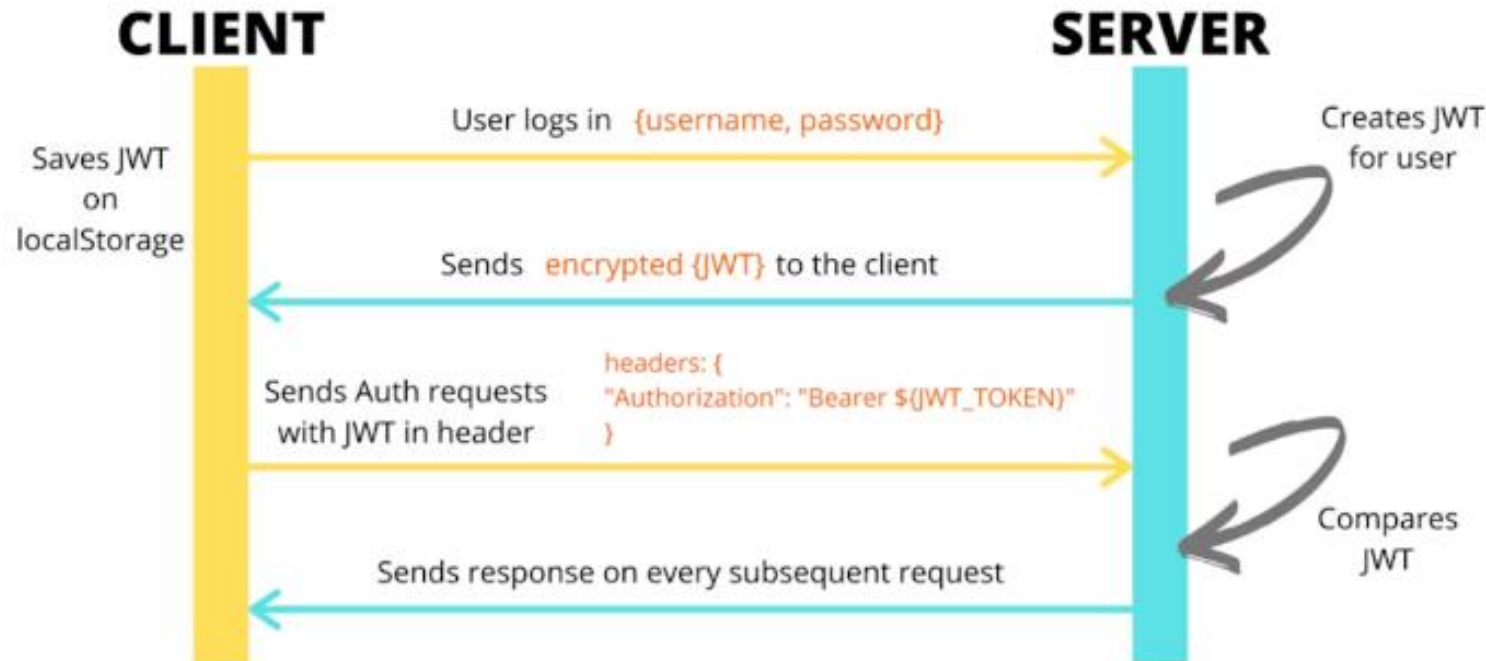
- 사용자가 로그인하면 서버는 사용자를 위한 세션을 만들고 세션 데이터를 서버 메모리에 저장.
- 사용자가 웹 사이트에서 특정 활동을 수행하는 동안 클라이언트 브라우저의 쿠키에 저장된 세션 ID가 생성됩니다. 사용자가 요청할 때마다 쿠키와 함께 전송됩니다.
- 서버는 사용자가 처음 로그인했을 때 서버 메모리에 저장된 세션 데이터로 쿠키의 세션 데이터를 확인할 수 있습니다.
- 사용자가 웹 사이트에서 로그 아웃하면 해당 세션 데이터가 데이터베이스 및 서버 메모리에서 삭제됩니다.



## 5. JWT

### ▶ 세션 인증 vs JWT 토큰 인증

#### Token Based Authentication



- 사용자가 로그인 세부 정보를 사용하여 사용자 인증 요청을 보내면, 서버는 JSON 웹 토큰 (JWT) 형식으로 암호화 된 토큰을 만들어 클라이언트로 다시 보냄
- 클라이언트가 토큰을 수신하면 사용자가 클라이언트를 사용하여 모든 활동을 수행하도록 인증되었음을 의미함.

## 5. JWT

---

### ▶ JWT 토큰 인증

- ▶ JWT는 클라이언트 localStorage에 저장되며 사용자가 서버에서 데이터를 요청하거나 수행할 때 해당 사용자의 고유 키로 사용
- ▶ 서버가 요청을 수신하면 해당 요청에 대해 JWT를 확인한 다음 필요한 응답을 클라이언트로 다시 보냄
- ▶ 요청시 http 헤더에 json 토큰이 추가되어 서버로 전송되는 방식

```
headers: {  
  "Authorization": "Bearer ${JWT_TOKEN}"  
}
```

### ▶ JSON 웹 토큰은 점 (으로 연결된 세 부분으로 구성 )

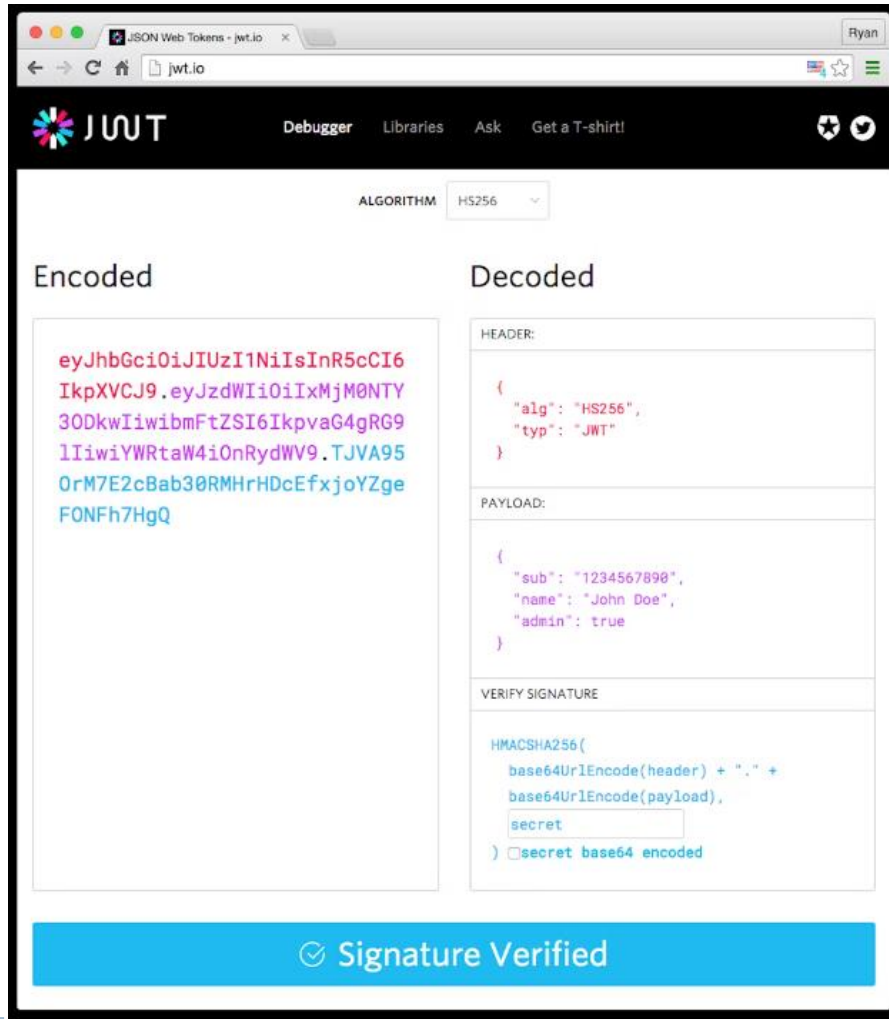
1. 헤더(Header)
2. 페이로드(Payload) : data
3. 서명(Signature)

```
xxxxxx.yyyyyy.zzzzzz
```



# 5. JWT

## ▶ JWT의 구조:



### header

토큰의 타입과 Signature 해싱 알고리즘

### payload(정보, data)

정보의 한 조각을 claim이라고 하며, 클레임은 Json(Key/Value) 형태의 한 쌍으로 이뤄져 있음

### 서명 (Signature)

토큰을 인코딩하거나 유효성 검증을 할 때 사용하는 고유한 암호화 코드

### Jwt의 단점

쿠키/세션과 다르게 토큰의 길이가 길어, 인증 요청이 많아질 수록 네트워크 부하가 심해 짐.  
쿠키/세션보다 복잡함.

## 5. JWT

### ▶ JWT 설정 과 properties 설정

- ▶ dependency 추가
- ▶ -> maven예

```
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.5</version>
</dependency>

<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.11.5</version>
  <scope>runtime</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-jackson -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.11.5</version>
  <scope>runtime</scope>
</dependency>
```

## 5. JWT

---

- ▶ JWT 설정 과 properties 설정
  - ▶ **application.properties**

```
#Key depends on JSW alogrithm HMAC ( >= 64bytes )  
app.jwt.secret=RandomSecretKey1234567890!RandomSecretKey1234567890!RandomSecretKey1234567890!  
#1 day  
app.jwt.expiration-in-ms=86400000
```

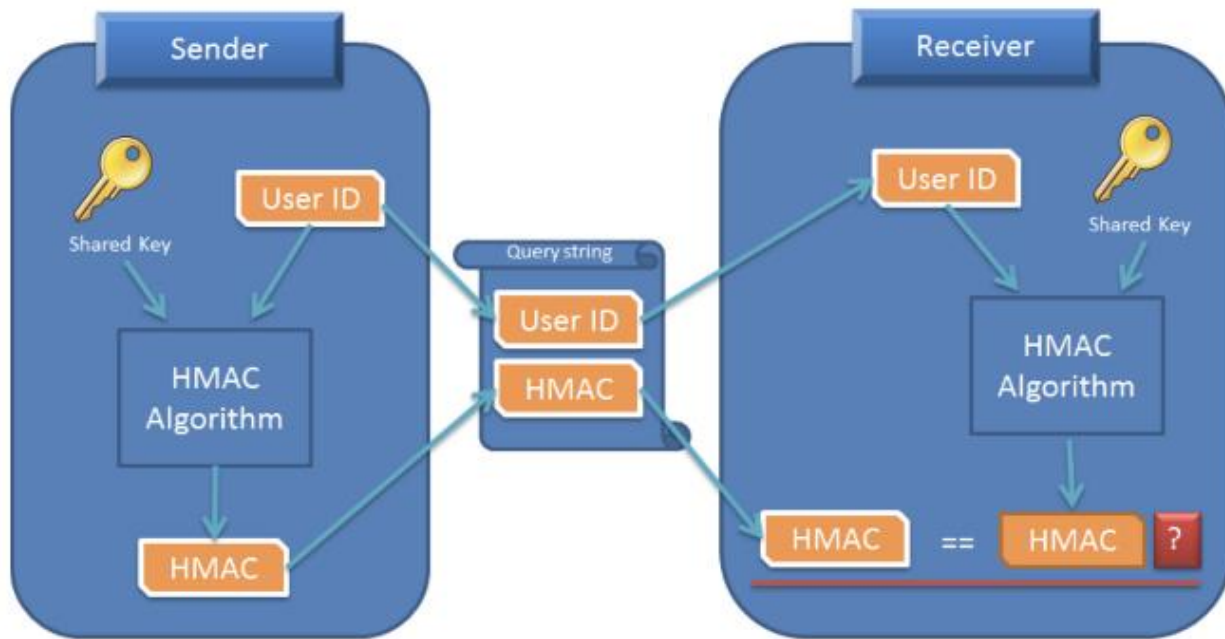
- HMAC(Hash-based Message Authentication Code)
- 메시지 인증 코드(Message Authentication Code, MAC)는 메시지의 인증에 쓰이는 정보(코드)이다.
- 메시지의 무결성 및 신뢰성을 보장하는 용도로 MAC을 사용한다



## 5. JWT

### ▶ HMAC(Hash-based Message Authentication Code)

- ▶ 메시지 인증 코드(Message Authentication Code, MAC)는 메시지의 인증에 쓰이는 정보(코드)
- ▶ 메시지의 무결성 및 신뢰성을 보장하는 용도로 MAC을 사용

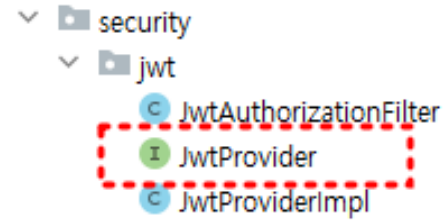


- ① 사전에 Sender와 Receiver는 별도 채널로 해시에 사용할 키(Share key)를 공유, 양쪽에서 사용할 해시 알고리즘을 정함.
- ② Sender는 공유키를 사용해서 UserId를 해싱.
- ③ Sender는 원본 UserId와 그 해시결과(HMAC)을 쿼리스트링 값으로 Receiver에게 전달.
- ④ Receiver는 받은 UserId를 공유키를 사용하여 같은 알고리즘으로 해시한 결과(Receiver's HMAC)를 만생
- ⑤ Receiver가 만든 HMAC과 쿼리스트링으로 받은 HMAC이 같다면 UserId는 변경되지 않았다고 신뢰할 수 있다.

## 5. JWT

### ▶ 토큰 생성 메서드

#### ▶ JwtProvider 인터페이스



```
package org.pgm.productseller.security.jwt;

import jakarta.servlet.http.HttpServletRequest;
import org.pgm.productseller.security.UserPrinciple;
import org.springframework.security.core.Authentication;

4개 사용 위치 1개 구현
public interface JwtProvider {
    1개 사용 위치 1개 구현
    String generateToken(UserPrinciple auth);

    1개 사용 위치 1개 구현
    Authentication getAuthentication(HttpServletRequest request);

    1개 사용 위치 1개 구현
    boolean isValidToken(HttpServletRequest request);
}
```

## 5. JWT

### ▶ JwtProviderImpl 토큰 생성 메서드

@Component

```
public class JwtProviderImpl implements JwtProvider {
```

```
    @Value("${app.jwt.secret}")
```

```
    private String JWT_SECRET;
```

```
    @Value("864000000")
```

```
    private Long JWT_EXPIRATION_IN_MS;
```

1개 사용 위치

@Override

```
public String generateToken(UserPrinciple auth) {
```

```
    String authorities = auth.getAuthorities().stream() Stream<capture of extends GrantedAuthority>
        .map(GrantedAuthority::getAuthority) Stream<String>
        .collect(Collectors.joining( delimiter: ","));
```

```
    Key key = Keys.hmacShaKeyFor(JWT_SECRET.getBytes(StandardCharsets.UTF_8));
```

```
    return Jwts.builder()
```

```
        .setSubject(auth.getUsername())
```

```
        .claim( s: "roles", authorities)
```

```
        .claim( s: "userId", auth.getId())
```

```
        .setExpiration(new Date(System.currentTimeMillis() + JWT_EXPIRATION_IN_MS))
```

```
        .signWith(key, SignatureAlgorithm.HS512)
```

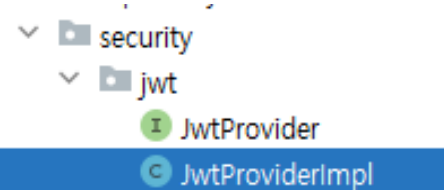
```
        .compact();
```

```
}
```

#### 토큰 생성 함수

UserPrinciple : 인증 후 유저정보가 저장된 객체

authorities : 문자열로 [ "ROLE\_ADMIN", "ROLE\_USER" ] 을 생성



```
"authorities": [
  "ROLE_ADMIN",
  "ROLE_USER"
]
```



## 5. JWT

### ▶ JwtProviderImpl 토큰 생성 메서드

#### JWT String

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0Ij0eXAIjYXNjaWkiOiJkbG9jaWkiLCJpdiI6ImVudCJ9.WRtaW4iOnRydWUsImp0aSI6ImUxOWM1ZDM2LTZhMjctNGM3Ny1hNzFiLTU0NjBjOWM3YzEzNSIsImhhdCI6MTU0MDgzNTg4MCwiZXhwIjoxNTQwODM5NDgwZlQzIj1JjGhhU5ns_GO8KsikiYehBtU4iLqHgpDAmMII
```

#### Header

```
{  
  "typ": "JWT",  
  "alg": "HS256"  
}
```

#### Payload

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true,  
  "jti": "e19c5d36-1a27-4c77-a71b-6460c9c7c135",  
  "iat": 1540835880,  
  "exp": 1540839480  
}
```

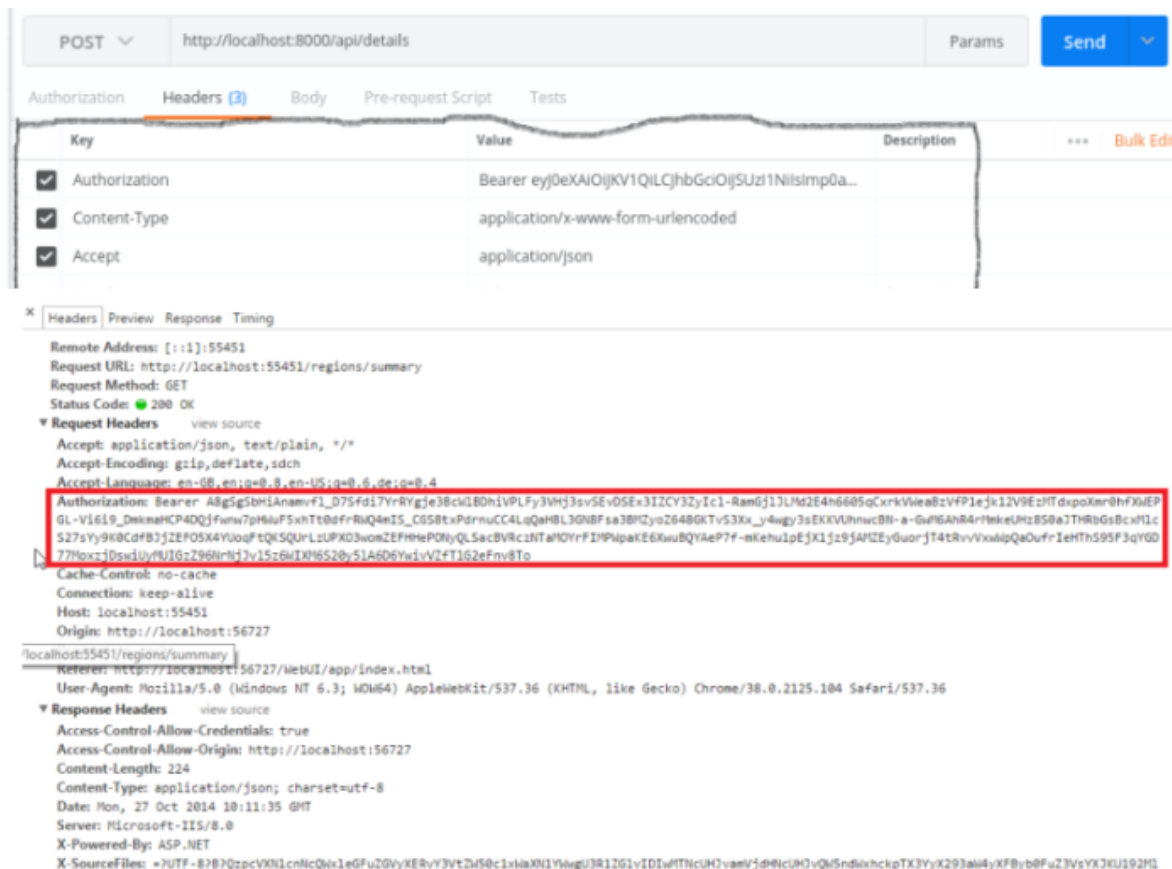
## 5. JWT

### ▶ 요청시 static token 가져오는 메서드

```
public class SecurityUtils {  
    2개 사용 위치  
    public static final String ROLE_PREFIX = "ROLE_";  
    1개 사용 위치  
    public static final String AUTH_HEADER = "authorization";  
    1개 사용 위치  
    public static final String AUTH_TOKEN_TYPE = "Bearer";  
    1개 사용 위치  
    public static final String AUTH_TOKEN_PREFIX = AUTH_TOKEN_TYPE + " ";  
    ...  
  
    public static String extractAuthTokenFromRequest(HttpServletRequest request){  
        String bearerToken = request.getHeader(AUTH_HEADER);  
  
        if(StringUtils.hasLength(bearerToken) && bearerToken.startsWith(AUTH_TOKEN_PREFIX)){  
            return bearerToken.substring(beginIndex: 7); //토큰 부분만 잘라서 리턴  
        }  
        return null; //없을경우  
    }  
}
```



## ▶ 요청시 static token 가져오는 메서드

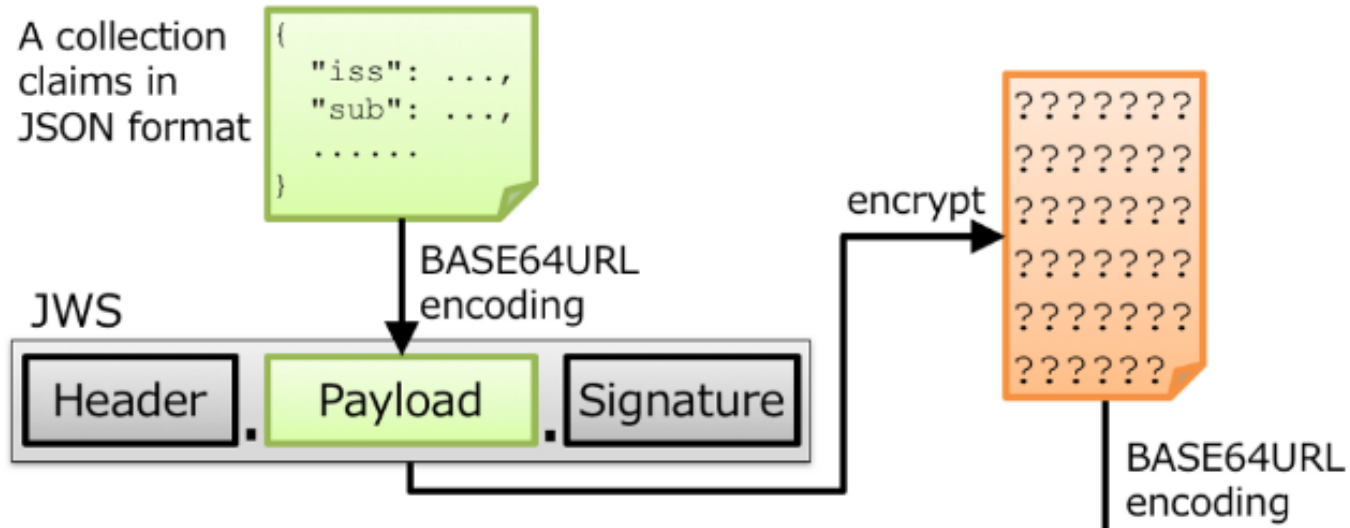


```
Authorization: Bearer <token>
```

## 5. JWT

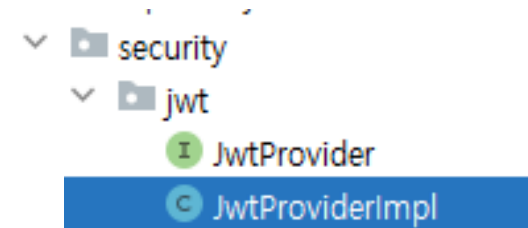
- ▶ 리퀘스트의 토큰에서 암호풀어 Claims 가져오기

### Nested JWT (JWS in JWE pattern)



## 5. JWT

### ▶ request의 토큰에서 암호풀어 Claims 가져오기



```
private Claims extractClaims(HttpServletRequest request){  
    String token = SecurityUtils.extractAuthTokenFromRequest(request);
```

```
    if(token == null) {  
        System.out.println("token null error");  
        return null;  
    }
```

request에서 토큰을 가져오고 토큰이 null이 아닐 경우에  
시크릿키를 사용해 다시 암호를 풀어  
Claims ( name/value 쌍으로 이루어진 값들 ) 형식으로 가져 오

```
    Key key = Keys.hmacShaKeyFor(JWT_SECRET.getBytes(StandardCharsets.UTF_8));
```

```
    return Jwts.parserBuilder() JwtParserBuilder  
        .setSigningKey(key)  
        .build() JwtParser  
        .parseClaimsJws(token) Jws<Claims>  
        .getBody();
```

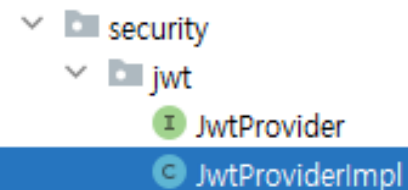
```
}
```

## 5. JWT

- ▶ 이름,비번,권한으로 인증토큰 생성 ,토큰이 사용가능한지 체크 하는 메서드

@Override

```
public Authentication getAuthentication(HttpServletRequest request) {  
    Claims claims = extractClaims(request);  
    if(claims == null) return null;  
    String username = claims.getSubject();  
    Long userId = claims.get("userId", Long.class);  
  
    Set<GrantedAuthority> authorities = Arrays.stream(claims.get("roles").toString().split(", "))  
        .map(SecurityUtils::convertToAuthority)  
        .collect(Collectors.toSet());  
    UserDetails userDetails = UserPrincipal.builder()  
        .username((username))  
        .authorities(authorities)  
        .id(userId)  
        .build();  
  
    if(username == null) {  
        System.out.println("username null error");  
        return null;  
    }  
  
    return new UsernamePasswordAuthenticationToken(userDetails, null, authorities);  
}
```



## 5. JWT

---

### ▶ 토큰 유효성 체크

```
@Override
public boolean isValid(HttpServletRequest request) {
    Claims claims = extractClaims(request);

    if(claims == null) {
        System.out.println("claims null error");
        return false;
    }

    if(claims.getExpiration().before(new Date())) return false;

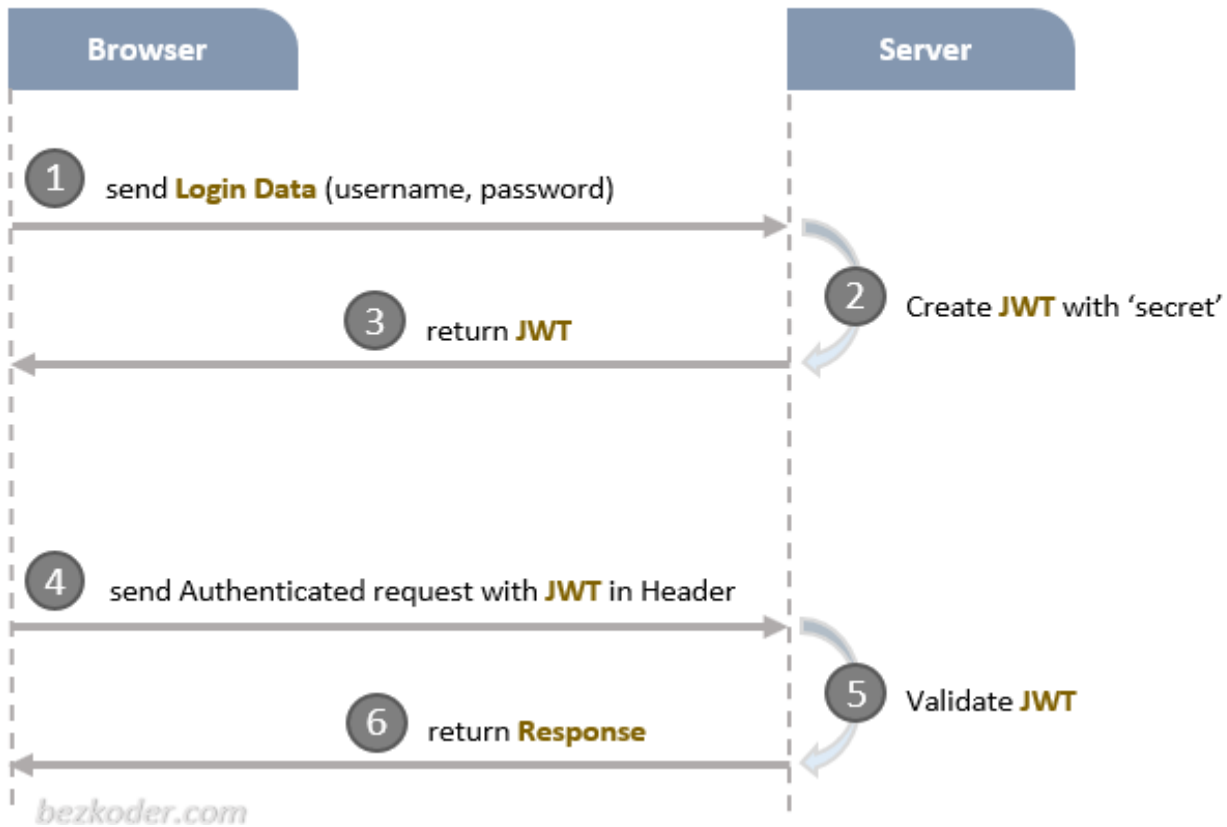
    return true;
}
```

토큰에 claims 데이터가 없으면 사용불가 false 리턴  
토큰사용기간 만료시에도 사용불가 false 리턴  
아니면 true 리턴



## 5. JWT

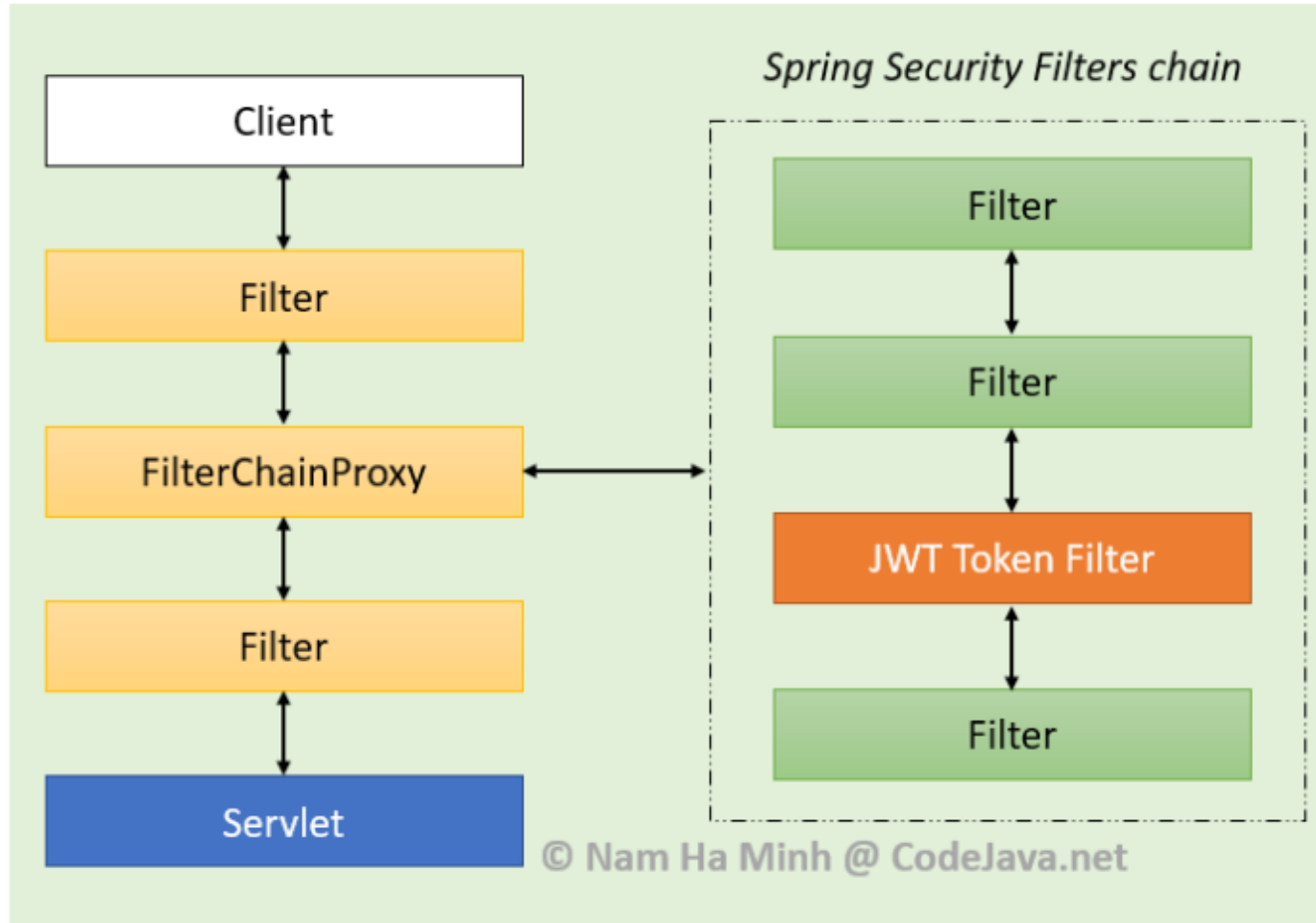
- ▶ 시큐리티 JWT 필터 만들기
  - ▶ JWT 인증 과정





## 5. JWT

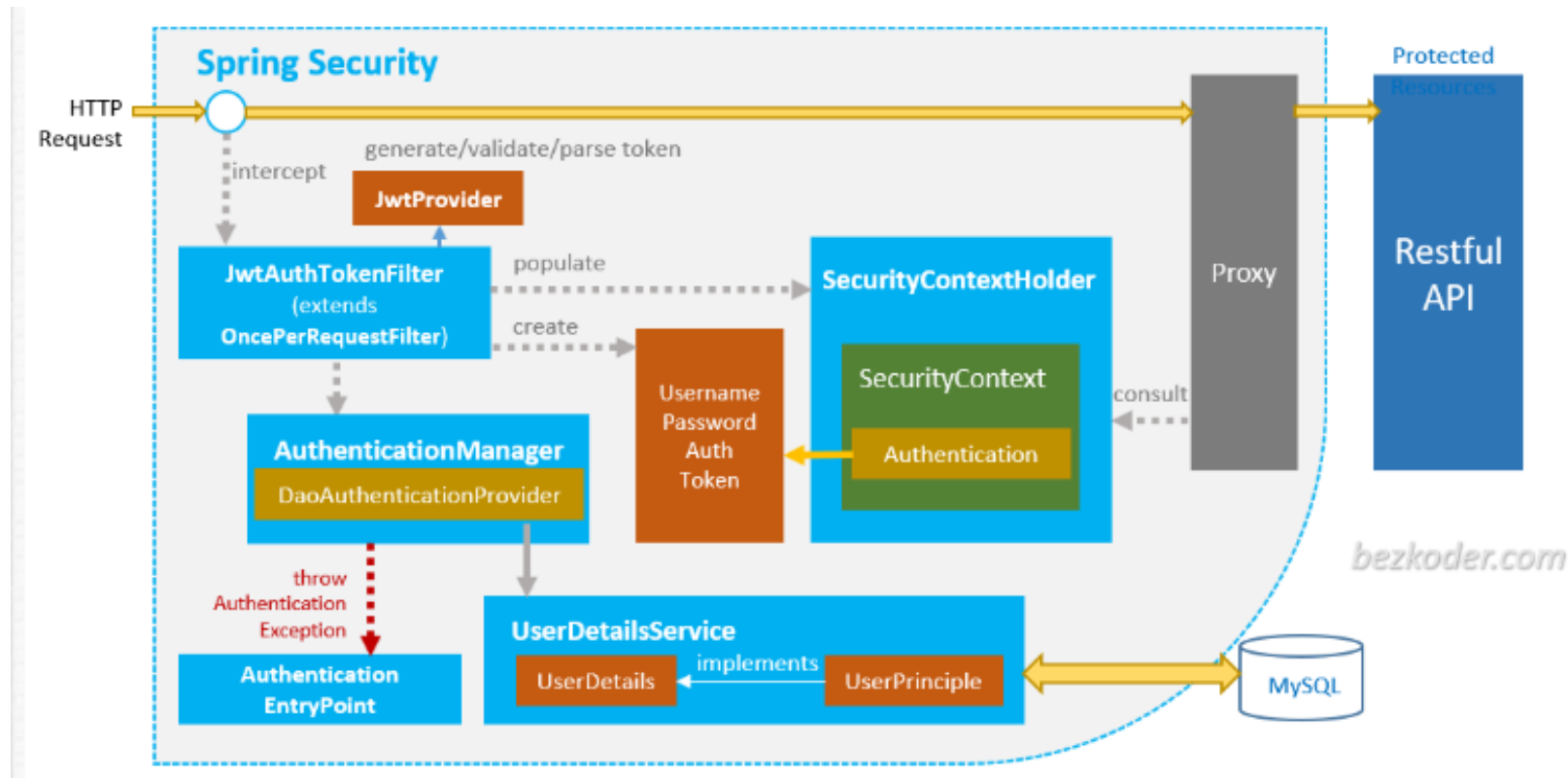
### ▶ 스프링 시큐리티 필터 체인



## 5. JWT

### ▶ 스프링 시큐리티 필터 체인

- ▶ 여러 개의 필터체인 중 JWT 토큰을 인증하기 위한 필터를 만들
- ▶ 단 한번에 인증으로 다시 인증 체크하는 일이 없도록 보장해야 한다!



## 5. JWT

### ▶ JwtAuthorizationFilter

@Component

```
public class JwtAuthorizationFilter extends OncePerRequestFilter {
```

@Autowired

```
private JwtProvider jwtProvider;
```

0개의 사용위치

@Override

```
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
```

```
throws ServletException, IOException {
```

```
Authentication authentication = jwtProvider.getAuthentication(request);
```

```
if(authentication != null && jwtProvider.isTokenValid(request)){
```

```
    SecurityContextHolder.getContext().setAuthentication(authentication);
```

```
    filterChain.doFilter(request, response);
```

```
}
```

```
}
```

jwt

JwtAuthorizationFilter

JwtProvider

JwtProviderImpl

OncePerRequestFilter Class : Http Request의 한 번의 요청에 대해 한 번만 실행하는 Filter  
예: 컨트롤러에서 Forwarding이 발생하면 Filter Chain이 다시 동작하여 인증처럼 딱 한번만 수행하도록 하여 Logic이 불필요하게 여러 번 수행될 수 있는 것을 막아 줌.

## 5. JWT

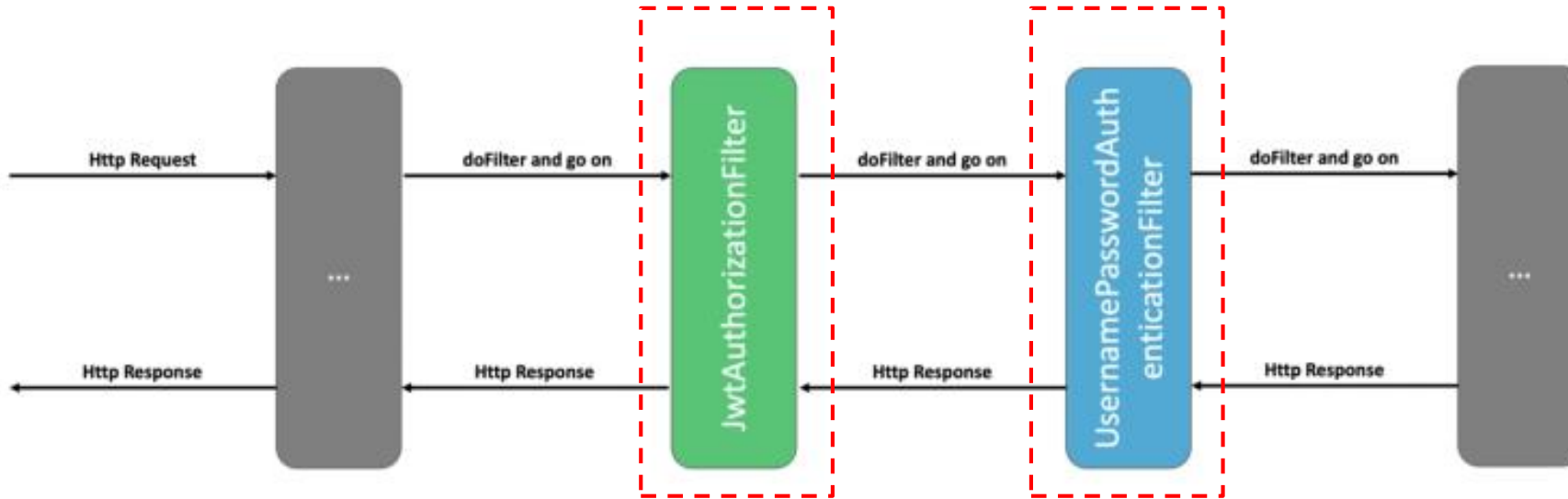
### ▶ SecurityConfig에 빈등록 및 필터 추가

```
@Bean
public JwtAuthorizationFilter jwtAuthorizationFilter(){
    return new JwtAuthorizationFilter();
}
```

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
    http
        .csrf(AbstractHttpConfigurer::disable)
        .cors(httpSecurityCorsConfigurer -> corsConfigurationSource())
        .sessionManagement((session)->session
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests((authz)-> authz
            .requestMatchers(🛡️"/api/authentication/**").permitAll()
            .anyRequest().authenticated()
        )
        .addFilterBefore(jwtAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

## 5. JWT

### ▶ 시큐리티 설정에 JWT필터 추가하기



JwtAuthorizationFilter : 직접 만든 커스텀 필터

UsernameAndPasswordAuthenticationFilter : 스프링 시큐리티의 기본 로그인 필터

반드시 커스텀 JWT 필터는 시큐리티의 기본 필터보다 먼저 처리되어야 함.

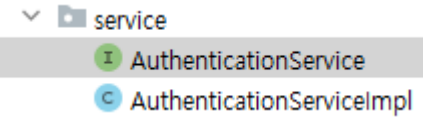
## 5. JWT

### ▶ AuthenticationServiceImpl 인증 메소드 구현

```
public interface AuthenticationService {
```

1개 사용 위치 1개 구현

```
    public User signInAndReturnJWT(User signInRequest);  
}
```



## 5. JWT

### ▶ AuthenticationServiceImpl 인증메소드 구현

```
@Service
@RequiredArgsConstructor
public class AuthenticationServiceImpl implements AuthenticationService{
    private final AuthenticationManager authenticationManager;
    private final JwtProvider jwtProvider;
    1개 사용 위치
    @Override
    public User signInAndReturnJWT(User signInRequest) {
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(signInRequest.getUsername(), signInRequest.getPassword())
        );

        UserPrincipal userPrincipal = (UserPrincipal) authentication.getPrincipal();
        String jwt = jwtProvider.generateToken(userPrincipal);

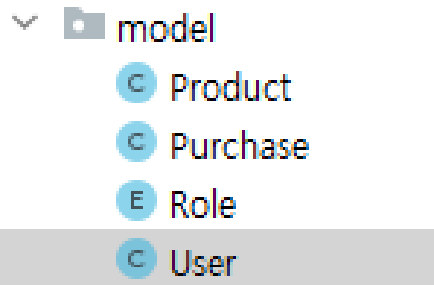
        User signInUser = userPrincipal.getUser();
        signInUser.setToken(jwt);

        return signInUser;
    }
}
```

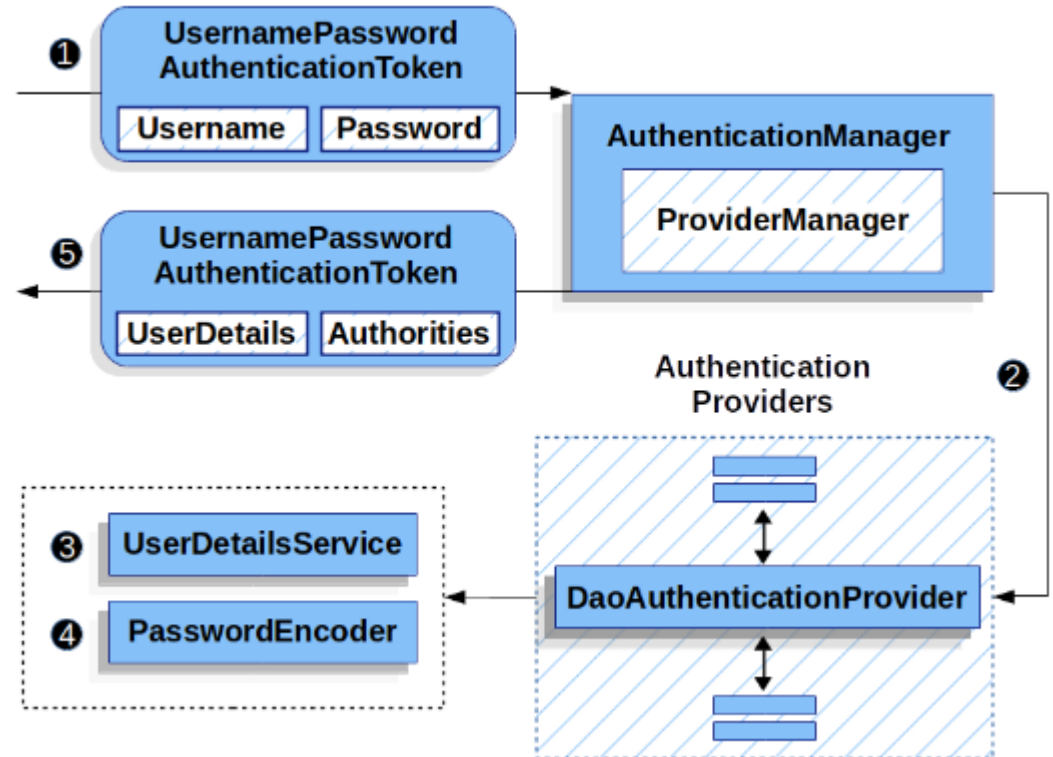
## 5. JWT

### ▶ AuthenticationServiceImpl 인증메소드 구현

- 유저의 로그인 요청시 AuthenticationManager 로 인증하고 인증되면 JwtProvider 에서 인증된 유저정보로 토큰을 발행
- 발행된 토큰은 유저의 필드변수에 저장장후 리턴됨( DB에는 저장되지 않도록 @)



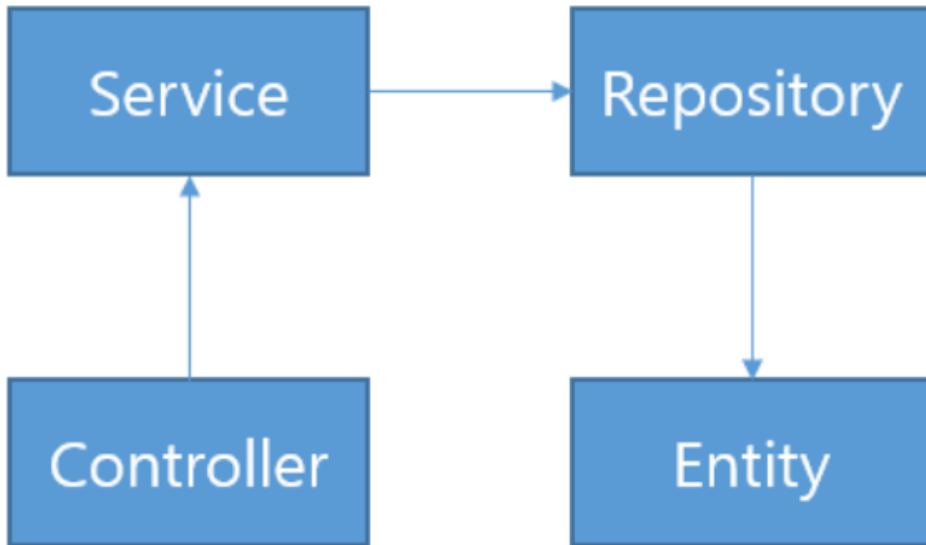
@Transient  
private String token; //DB에 저장되지 않음





## 5. JWT

### ▶ 로그인과 가입하기



Entity => Repository => Service => 컨트롤러 작성

인증관련 엔드포인트 주소  
sign-in 로그인, sign-up 회원가입

Sign-In => **POST** /api/authentication/sign-in

Sign-Up => **POST** /api/authentication/sign-up

```
.authorizeHttpRequests((authz)-> authz
    .requestMatchers(🛡️"/api/authentication/**").permitAll()
    .anyRequest().authenticated()
)
```

시큐리티 설정에서 모든 사용자에게 허용되는 엔드포인트 주소는 /api/authentication/모든주소 이므로 인증 로그인, 회원가입 요청은 인증 되지 않은 사용자(모든 사용자)에게 허용됨

# 5. JWT

## ► AuthenticationController

```
@RestController
@RequestMapping("/api/authentication")
@RequiredArgsConstructor
public class AuthenticationController {
    private final AuthenticationService authenticationService;
    private final UserService userService;

    @PostMapping("/sign-up")
    public ResponseEntity<Object> signUp(@RequestBody User user) {
        if(userService.findByUsername(user.getUsername()).isPresent()){
            return new ResponseEntity<>(HttpStatus.CONFLICT);
        }
        return new ResponseEntity<>(userService.saveUser(user), HttpStatus.CREATED);
    }

    @PostMapping("/sign-in")
    public ResponseEntity<Object> signIn(@RequestBody User user){
        return new ResponseEntity<>(authenticationService.signInAndReturnJWT(user), HttpStatus.OK);
    }
}
```

## 5. JWT

### ▶ 포스트맨 테스트 ( sign-up 가입하기)

POST http://localhost:8080/api/authentication/sign-up

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▾

```
1 {
2   ... "username": "user",
3   ... "password": "user",
4   ... "name": "user"
5 }
```

Body Cookies Headers (11) Test Results 201 Created 299 ms 523 B Save

Pretty Raw Preview Visualize **JSON** ▾ ↻

```
1 {
2   "id": 1,
3   "username": "user",
4   "password": "$2a$10$6KRLAZiyDwSps5wmXUGFd0JAnu0Xpgp4hea/VNjYY5fLXfAiq4m.0",
5   "name": "user",
6   "createTime": "2023-07-01T19:15:40.6700714",
7   "role": "USER",
8   "token": null
9 }
```

## ▶ 포스트맨 테스트 ( sign-in 로그인 인증)



## 5. JWT

---

### ▶ 로그인(인증)된 사용자 정보 가져오는 방법(Tip)

#### ▶ Bean 에서 사용자 정보 얻기

```
Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();  
UserDetails userDetails = (UserDetails)principal;  
String username = principal.getUsername();
```

#### ▶ Controller 에서 사용자 정보 얻기(Principal 객체로 얻기)

```
@Controller  
public class SecurityController {  
    @GetMapping("/username")  
    @ResponseBody  
    public String currentUserUsername(Principal principal) {  
        return principal.getName();  
    }  
}
```



## 5. JWT

---

- ▶ 로그인(인증)된 사용자 정보 가져오는 방법(Tip)
  - ▶ **Controller** 에서 사용자 정보 얻기(**Authentication** 객체로 얻기)

```
@Controller
public class SecurityController{
    @GetMapping("/username")
    @ResponseBody
    public String currentUser_name(Authentication authentication) {
        UserDetails userDetails = (UserDetails) authentication.getPrincipal();
        return userDetails.getUsername();
    }
}
```



## 5.JWT

---

- ▶ 로그인(인증)된 사용자 정보 가져오는 방법(Tip)
  - ▶ **request 객체에서 얻기**

```
@RequestMapping("/username")  
@ResponseBody  
public String currentUserSimpleName(HttpServletRequest request) {  
    Principal principal = request.getUserPrincipal();  
    return principal.getName();  
}
```



## 5. JWT

---

### ▶ 로그인(인증)된 사용자 정보 가져오는 방법(Tip)

#### ▶ **@AuthenticationPrincipal**

- ▶ Spring Security 3.2 부터는 annotation을 이용하여 현재 로그인한 사용자 객체를 인자에 주입할 수 있음.
- ▶ @AuthenticationPrincipal를 이용하여 CustomUser 객체에 사용자 객체를 주입

```
@Controller
public class SecurityController {
    @GetMapping("/messages/inbox")
    public ModelAndView currentUserName(@AuthenticationPrincipal CustomUser customUser) {
        String username = customUser.getUsername();
        // .. find messages for this user and return them ...
    }
}
```





## 6. Controller

### ▶ 유저 컨트롤러

- ▶ 새유저 생성(sign-up), 로그인(sign-in) 등은 AuthController에서 처리
- ▶ role 업데이트 처리

```
@RestController
@RequestMapping("/api/user")
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;

    @PutMapping("change/{role}")
    public ResponseEntity<Object> changeRole(@AuthenticationPrincipal UserPrincipal userPrincipal,
                                              @PathVariable Role role) {
        userService.changeRole(role, userPrincipal.getUsername());
        return ResponseEntity.ok( body: true);
    }
}
```

## 6. Controller

### ▶ 유저 컨트롤러- 테스트

#### ▶ 테스트 포스트맨에서 admin 유저 가입하기

POST http://localhost:8090/api/authentication/sign-up

Params Authorization Headers (9) **Body** Scripts Tests Settings Cookies Beautify

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

```
1 {
2   "username": "admin",
3   "password": "admin",
4   "name": "admin"
5 }
```

**Result Grid** Filter Rows: Edit: Export/Import: Wrap Cell Content

|   | id   | create_time                | name  | password  | role | username |
|---|------|----------------------------|-------|---|------|----------|
| ▶ | 1    | 2023-07-01 19:32:07.655572 | user  | \$2a\$10\$LL1dJhlj7a.rhMWJEDsHB.nfHpbrkr44EI... | USER | user     |
|   | 2    | 2023-07-02 08:09:12.719145 | admin | \$2a\$10\$gYEmg2z20ZL5B52ztQaGK.SoCoXTnED...    | USER | admin    |
| * | NULL | NULL                       | NULL  | NULL  | NULL | NULL     |

body Cookies Headers (14) Test Results Status: 201 Created Time: 276 ms Size: 613 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 7,
3   "username": "admin",
4   "password": "$2a$10$76RsCA279engAxTGXWcag.0jXrKJ5KT0Gb4k8z7eMUx3SKalPGEVS",
5   "name": "admin",
6   "createTime": "2024-06-18T15:33:01.351854",
7   "role": "USER",
8   "token": null
9 }
```

## 6. Controller

- ▶ 유저 컨트롤러- 테스트
  - ▶ 로그인 하기

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://localhost:8090/api/authentication/sign-in
- Body:**

```
{
  "username": "admin",
  "password": "admin"
}
```
- Status:** 200 OK
- Time:** 391 ms
- Size:** 784 B
- Response Body:**

```
{
  "id": 7,
  "username": "admin",
  "password": "$2a$10$76RsCA279engAxTGXWcag.0jXrKJ5KT0Gb4k8z7eMUx3SKalPGEVS",
  "name": "admin",
  "createTime": "2024-06-18T15:33:01.351854",
  "role": "USER",
  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbGlzInJvbGVzIjoiaUk9MRV9VU0VSIiwiaXhwIjoxNzc4NDc3fQ.9bNS0s7RGHxxKdzWaq-3vj9Dm6n-_62uHRa0qkpOLhmZQv63GCxEjbedDXjUKXirYE7tW4CHtmUfj6_BVHgN6w"
}
```

## 6. Controller

- ▶ 유저 컨트롤러- 테스트
  - ▶ 유저 롤 업데이트

The screenshot shows a REST client interface. The request method is PUT, and the URL is http://localhost:8090/api/user/change/ADMIN. The request body is a JSON object: { "username": "admin", "password": "admin" }. The response status is 403 Forbidden, which is highlighted with a red dashed box. The response time is 16 ms and the size is 389 B. The response body is empty.

```
1 {
2   ... "username": "admin",
3   ... "password": "admin"
4 }
```

Status: 403 Forbidden Time: 16 ms Size: 389 B Save as example

결과 : 403 권한없음, 유저 업데이트 요청시에는 인증된 유저의 요청만 받음.  
이유는 ?  
기존의 시큐리티 로그인시 인증된 정보가 세션에 남아있기 때문에 유지되지만  
JWT 토큰 방식은 요청시 헤더에 토큰을 가지고 있어야 한다.

## 6. Controller

- ▶ 유저 컨트롤러- 테스트
  - ▶ 유저 롤 업데이트

PUT ▼ http://localhost:8090/api/user/change/ADMIN Send ▼

Params Authorization Headers (10) Body Scripts Tests Settings Cookies

**Auth Type**  
Bearer Token ▼

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about [variables](#).

Token  
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pt...

Body Cookies Headers (14) Test Results

Pretty Raw Preview Visualize JSON ▼ 🔗

1 true

**Result Grid** 📄 ↺ Filter Rows: Edit: Export/Import: Wrap Cell Content

|   | id   | create_time                | name  | password                                      | role  | username |
|---|------|----------------------------|-------|---|-------|----------|
| ▶ | 1    | 2023-07-01 19:32:07.655572 | user  | \$2a\$10\$LL1dJhlj7a.rhMWJEDsHB.nfHpbrkr44EI. | USER  | user     |
|   | 2    | 2023-07-02 08:09:12.719145 | admin | \$2a\$10\$gYEmg2z20ZL5B52ztQaGK.SoCoXTnED.    | ADMIN | admin    |
| * | NULL | NULL                       | NULL  | NULL  | NULL  | NULL     |

## 6. Controller

### ▶ ProductController

- ▶ 제품 추가
- ▶ 제품 리스트 가져오기
- ▶ 제품삭제

```
@RestController
@RequiredArgsConstructor
@RequestMapping("api/product")
public class ProductController {
    private final ProductService productService;

    @PostMapping
    public ResponseEntity<Object> saveProduct(@RequestBody Product product){
        return new ResponseEntity<>(productService.saveProduct(product) , HttpStatus.CREATED);
    }

    @GetMapping
    public ResponseEntity<Object> getAllProducts(){
        return new ResponseEntity<>(productService.findAllProducts(), HttpStatus.OK);
    }

    @DeleteMapping("{productId}")
    public ResponseEntity<Object> deleteProduct(@PathVariable Long productId){
        productService.deleteProduct(productId);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

## 6. Controller

### ▶ ProductController

- ▶ 시큐리티의 필터체인 수정 제품리스트는 누구든지 가능
- ▶ 그 외 제품 수정, 삭제, 입력은 ADMIN 롤만 가능

@Bean

```
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
    http
        .csrf(AbstractHttpConfigurer::disable)
        .cors(httpSecurityCorsConfigurer -> corsConfigurationSource())
        .sessionManagement((session)->session
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests((authz)-> authz
            .requestMatchers(🛡️ "/api/authentication/**").permitAll()
            .requestMatchers(HttpMethod.GET, 🛡️ "/api/product/**").permitAll()
            .requestMatchers(🛡️ "/api/product/**").hasRole(Role.ADMIN.name())
            .anyRequest().authenticated()
        )
        .addFilterBefore(jwtAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

## 6. Controller

### ▶ ProductController 테스트- 제품추가 :ADMIN Role만 접근 가능

POST http://localhost:8090/api/product

Params Authorization Headers (10) Body Scripts Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "name": "자장면",
3   "description": "보통",
4   "price": 7000
5 }
```

Body Cookies Headers (14) Test Results Status: 201 Created Time: 231 ms Size: 533 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "name": "자장면",
4   "description": "보통",
5   "price": 7000,
6   "createTime": "2024-06-18T16:01:04.872335"
7 }
```



# 6.Controller

## ▶ ProductController 테스트- 제품 리스트 얻기

The screenshot shows a REST client interface with a GET request to `http://localhost:8090/api/product`. The response status is 200 OK, and the body is a JSON array of two product objects.

Request details:

- Method: GET
- URL: `http://localhost:8090/api/product`
- Body: none (selected)

Response details:

- Status: 200 OK
- Time: 12 ms
- Size: 633 B

Response body (JSON):

```
[
  {
    "id": 1,
    "name": "자장면",
    "description": "보통",
    "price": 7000,
    "createTime": "2024-06-18T16:01:04.872335"
  },
  {
    "id": 2,
    "name": "짬뽕",
    "description": "보통",
    "price": 8000,
    "createTime": "2024-06-18T16:13:47.142108"
  }
]
```

## 6. Controller

### ▶ ProductController 테스트- 제품 삭제

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** http://localhost:8090/api/product/1
- Buttons:** Send, Cookies
- Tabs:** Params, Authorization, Headers (8), Body, Scripts, Tests, Settings
- Body Type:** none (selected), form-data, x-www-form-urlencoded, raw, binary, GraphQL
- Message:** This request does not have a body
- Response Tabs:** Body, Cookies, Headers (13), Test Results
- Response Status:** Status: 200 OK, Time: 34 ms, Size: 382 B
- Response Actions:** Save as example, Copy, Search
- Response Format:** Pretty (selected), Raw, Preview, Visualize
- Response Content Type:** Text
- Response Body:** 1

## 6. Controller

### ▶ PurchaseController

- ▶ 구매추가, 사용자별 구매리스트

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/api/purchase")
public class PurchaseController {
    private final PurchaseService purchaseService;

    @PostMapping
    public ResponseEntity<Object> savePurchase(@RequestBody Purchase purchase){
        return new ResponseEntity<>(purchaseService.savePurchase(purchase), HttpStatus.CREATED);
    }

    @GetMapping("/getList")
    public ResponseEntity<Object> getAllPurchasesOfUser(@AuthenticationPrincipal UserPrincipal userPrincipal){
        System.out.println("~~~~~"+userPrincipal.getUser());
        return ResponseEntity.ok(purchaseService.findPurchaseItemsOfUser(userPrincipal.getUsername()));
    }
}
```

## 6. Controller

---

- ▶ PurchaseServiceImpl 의 findPurchaseItemofUser() 수정

```
...  
@Override  
public List<PurchaseItem> findPurchaseItemsOfUser(String username) {  
    System.out.println("service~~~~~"+username);  
    User user=userRepository.findByUsername(username).orElseThrow();  
  
    return purchaseRepository.findAllPurchasesOfUser(user.getId());  
}
```



## 6. Controller

### ▶ user 사용자로 로그인

The screenshot shows a REST client interface with a POST request to `http://localhost:8090/api/authentication/sign-in`. The request body is a JSON object with `username: "user"` and `password: "user"`. The response status is 200 OK, and the response body is a JSON object containing user details and a token.

**Request:**

```
POST http://localhost:8090/api/authentication/sign-in
```

**Body:**

```
{
  "username": "user",
  "password": "user"
}
```

**Response:**

```
{
  "id": 1,
  "username": "user",
  "password": "$2a$10$zcc.vYgGmuA66oRd77FoX0kUhspT3plRdTtn1yNj7ZiQp/7TTArcu",
  "name": "user",
  "createTime": "2024-06-16T17:16:10.953417",
  "role": "USER",
  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ1c2VyIiwicm9sZSI6ImF1dG8iLCJleHAiOiE3MTg3ODE5MjF9.ncNmKsm-tCT2jmeLPW4ZEz_BmlxpIRc5m6UscPqUM6Y1b8VFyxXxBPYi4kvgcvsG7pnioLWP1iSnA55yydQxMg"
}
```

## 6. Controller

### ▶ 구매추가 : body에 추가 데이터, Authorization에 Token 설정

The screenshot displays the Postman interface for a POST request to `http://localhost:8090/api/purchase`. The request body is a JSON object: `{"userId":1,"productId":1,"quantity":5}`. The Authorization tab is selected, showing the Auth Type as "Bearer Token". A warning message states: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about [variables](#)." The Token field contains the value `eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ1c2Vyli...`. The bottom status bar indicates a 201 Created status, 245 ms time, and 546 B size.

POST `http://localhost:8090/api/purchase` Send

Params • Authorization • Headers (10) Body • Scripts Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON

1 `{"userId":1,"productId":1,"quantity":5}`

POST `http://localhost:8090/api/purchase` Send

Params Authorization • Headers (10) Body • Scripts Tests Settings Cookies

Auth Type  
Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Token  
`eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ1c2Vyli...`

Body Cookies Headers (14) Test Results

Status: 201 Created Time: 245 ms Size: 546 B Save as example

## 6. Controller

### ▶ 구매리스트 확인

The screenshot shows a web browser's developer tools interface. The address bar displays a local URL: `http://localhost:8080/api/buyList`. The 'Body' tab is selected, showing a JSON response. The response is a list of three items, each with a name, quantity, and purchase time.

Request URL: `http://localhost:8080/api/buyList`

Params: Authorization Headers (8) Body Scripts Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (14) Test Results Status: 200 OK Time: 24 ms Size: 653 B Save as example

Pretty Raw Preview Visualize JSON

```
[
  {
    "name": "짜장면",
    "quantity": 10,
    "purchaseTime": "2024-06-18T18:39:33.44699"
  },
  {
    "name": "짬뽕",
    "quantity": 5,
    "purchaseTime": "2024-06-18T18:40:09.450081"
  },
  {
    "name": "짬뽕",
    "quantity": 10,
    "purchaseTime": "2024-06-19T10:31:58.474666"
  }
]
```