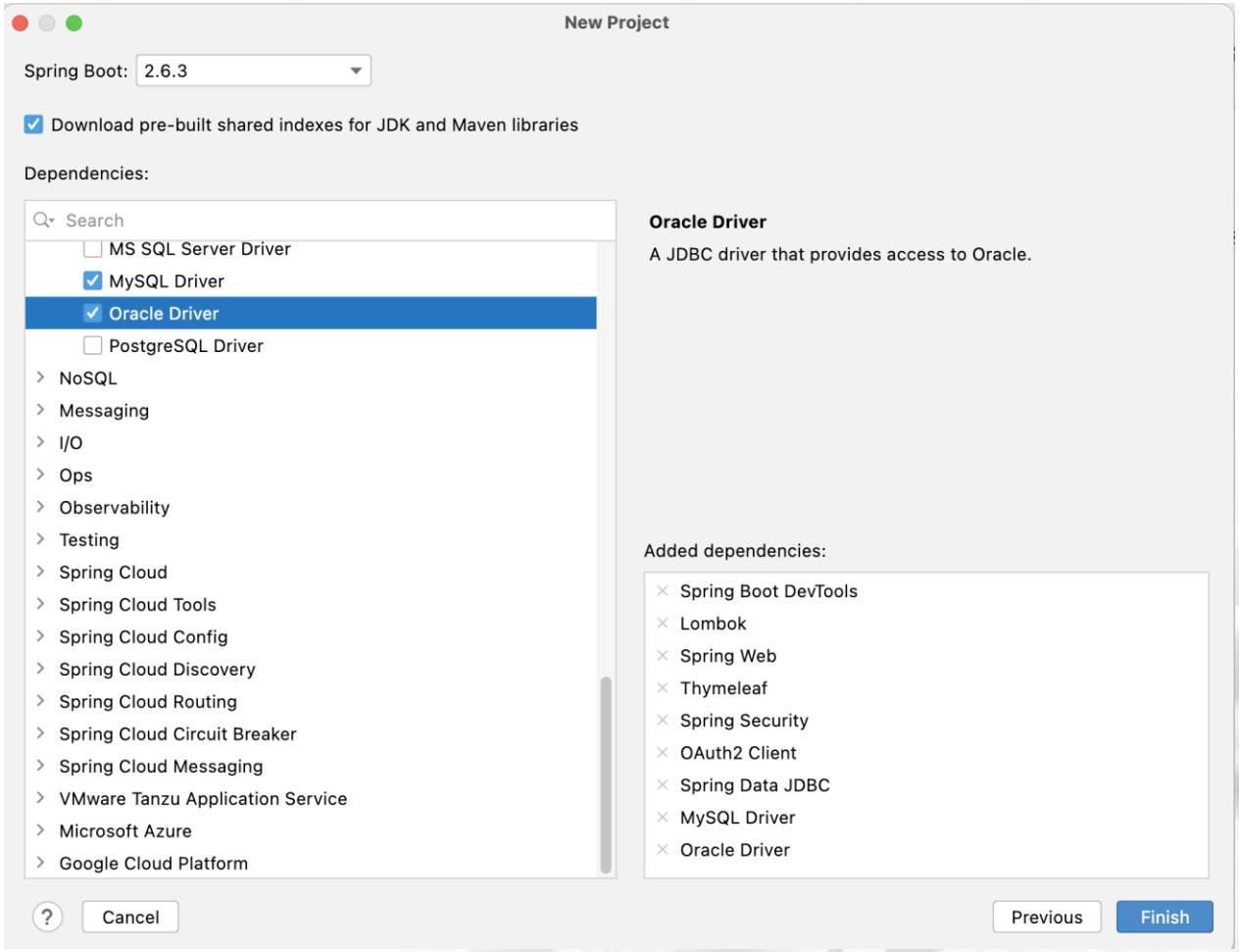
Spring Boot Security

Spring Boot Security

- 인증(Authentication) 과 인가(Authorization)
  - ✓ 인증이란 해당 리소스에 대해서 작업을 수행할 수 있는 주체인지 확인하는 것으로 어떤 커뮤니티에서 게시판의 글을 보는 것은 로그인을 하지 않아도 되지만 댓글을 작성하려면 로그인을 해야 하는데 이렇게 자격을 증명하는 과정이 인증
  - ✓ 인가는 인증 과정 이후에 일어나는데 커뮤니티를 관리하는 관리자 페이지에 접근하는 URL을 입력 했을 때 해당 URL은 커뮤니티의 관리자만 접근할 수 있어야 하는데 이때 접근하는 사용자가 인가된 회원인지를 검사하는 것으로 일반적으로 권한을 확인하는 작업

Spring Boot Security

❖ 프로젝트 생성(club) – Spring Security 와 OAuth2 Client 추가



Spring Boot Security

- 프로젝트 생성(club)
  - ✓ application.properties 파일을 application.yml 파일로 변경하고 설정

```
server:
  port: 80

spring:
  datasource:
    url: jdbc:mariadb://localhost:3306/adam
    driver-class-name: org.mariadb.jdbc.Driver
    username: root
    password: wnddkd
  jpa:
    hibernate:
      ddl-auto: update
    properties:
      format_sql: true
      show_sql: true
```

Spring Boot Security

- 프로젝트 생성(club)
  - ✓ application.properties 파일을 application.yml 파일로 변경하고 설정

```
#Live Reload
devtools:
  livereload:
    enabled: true

thymeleaf:
  cache: false

servlet:
  multipart:
    enabled: true
  location: /Users/adam/Documents/data
  max-request-size: 30MB
  max-file-size: 10MB
```

Spring Boot Security

- 프로젝트 생성(club)
  - ✓ application.properties 파일을 application.yml 파일로 변경하고 설정

```
com:
  adamsoft:
    upload:
      path: /Users/adam/Documents/data

logging:
  level:
      org:
        hibernate:
          type:
            descriptor:
              sql: trace
```

Spring Boot Security

- 프로젝트 생성(club)
  - ✓ 실행 클래스에 어노테이션 추가

```
@SpringBootApplication
@EnableJpaAuditing
public class ClubApplication {
    public static void main(String[] args) {
        SpringApplication.run(ClubApplication.class, args);
    }
}
```

Spring Boot Security

- 프로젝트 생성(club)
  - ✓ 프로젝트를 실행하면 user 계정의 패스워드가 출력됨

```
Using generated security password: d76cb156-3eba-4fdf-89c8-567a836a9753


2022-01-22 12:00:54.359 DEBUG 28019 --- [  restartedMain] edFilterInvocationSecurityMetadataSource : Adding web access control expression [authenticated] for any r
2022-01-22 12:00:54.376  INFO 28019 --- [  restartedMain] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.springframework.security.web
2022-01-22 12:00:54.444  INFO 28019 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2022-01-22 12:00:54.481  INFO 28019 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-01-22 12:00:54.495  INFO 28019 --- [  restartedMain] kr.co.adamsoft.club.ClubApplication      : Started ClubApplication in 3.042 seconds (JVM running for 6.45
```

Spring Boot Security

- 프로젝트 생성(club)
  - ✓ Spring Security는 별도의 설정이 없을 때는 모든 자원에 필요한 권한이나 로그인 여부를 확인한 후 로그인이 되어 있지 않으면 login 으로 redirect를 수행

```
2023-01-15T11:13:30.428+09:00 DEBUG 13773 --- [p-nio-80-exec-1]
o.s.security.web.FilterChainProxy         : Securing GET /
2023-01-15T11:13:30.439+09:00 DEBUG 13773 --- [p-nio-80-exec-1]
o.s.s.w.a.AnonymousAuthenticationFilter   : Set SecurityContextHolder to anonymous
SecurityContext
2023-01-15T11:13:30.466+09:00 DEBUG 13773 --- [p-nio-80-exec-1]
o.s.s.w.s.HttpSessionRequestCache         : Saved request http://localhost/?continue to
session
2023-01-15T11:13:30.469+09:00 DEBUG 13773 --- [p-nio-80-exec-1]
s.w.a.DelegatingAuthenticationEntryPoint : Trying to match using And [Not
[RequestHeaderRequestMatcher [expectedHeaderName=X-Requested-With,
expectedHeaderValue=XMLHttpRequest]], MediaTypeRequestMatcher
[contentNegotiationStrategy=org.springframework.web.accept.ContentNegotiationMana
ger@2a70ac35, matchingMediaTypes=[application/xhtml+xml, image/*, text/html,
text/plain], useEquals=false, ignoredMediaTypes=[*/*]]]
2023-01-15T11:13:30.469+09:00 DEBUG 13773 --- [p-nio-80-exec-1]
s.w.a.DelegatingAuthenticationEntryPoint : Match found! Executing
org.springframework.security.web.authentication.LoginUrlAuthenticationEntryPoint@3d7
d65b1
```

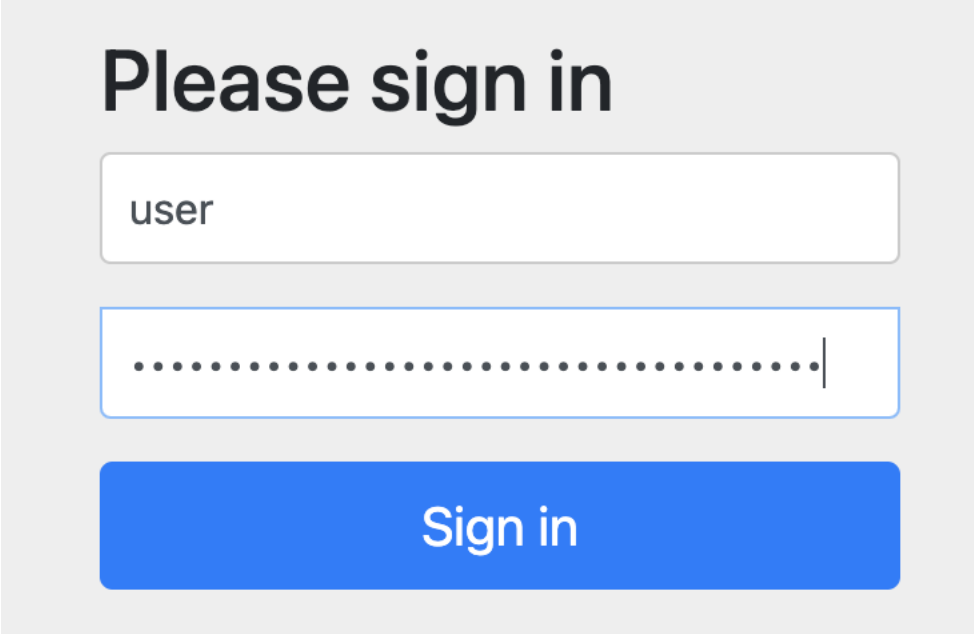
Spring Boot Security

- 프로젝트 생성(club)
  - ✓ Spring Security는 별도의 설정이 없을 때는 모든 자원에 필요한 권한이나 로그인 여부를 확인한 후 로그인이 되어 있지 않으면 login 으로 redirect를 수행

```
2023-01-15T11:13:30.470+09:00 DEBUG 13773 --- [p-nio-80-exec-1]
o.s.s.web.DefaultRedirectStrategy        : Redirecting to http://localhost/login
2023-01-15T11:13:30.478+09:00 DEBUG 13773 --- [p-nio-80-exec-2]
o.s.security.web.FilterChainProxy        : Securing GET /login
```

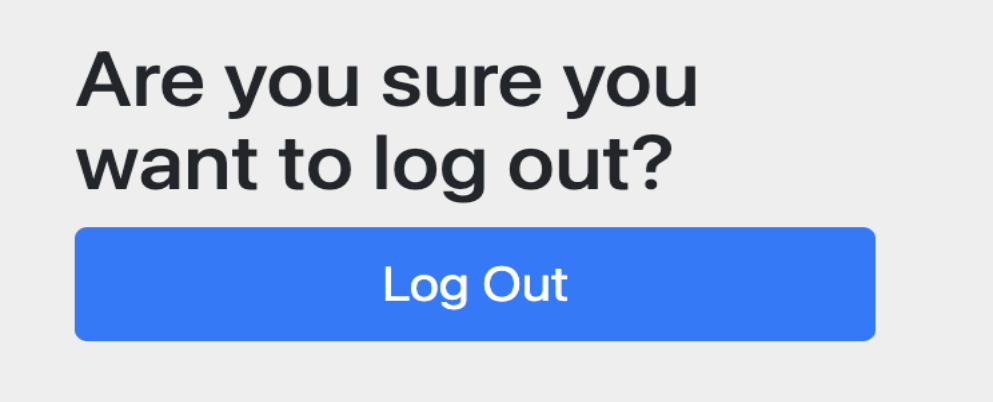
Spring Boot Security

- 프로젝트 생성(club)
  - ✓ 프로젝트 생성 시 만들어진 user 와 비밀번호로 로그인 테스트



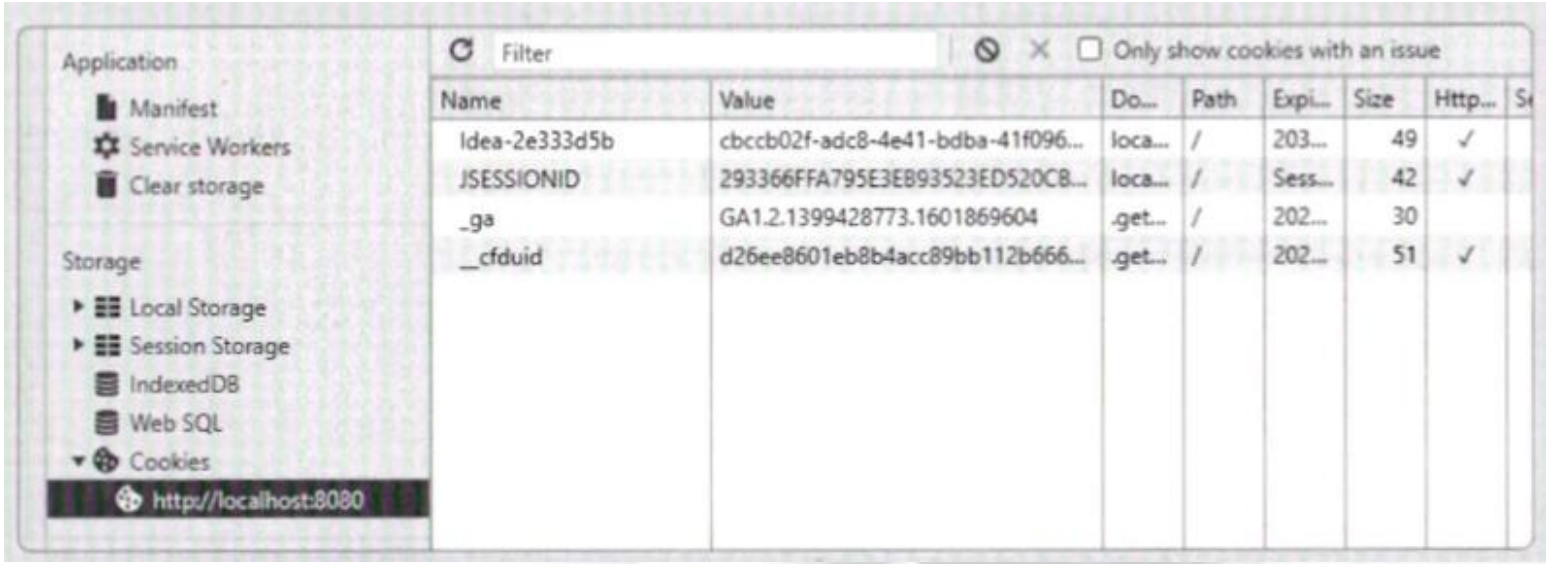
Spring Boot Security

- 프로젝트 생성(club)
  - ✓ Spring Security에서는 login 과 logout 요청 기능을 제공



Spring Boot Security

- 브라우저에서 쿠키 삭제
  - ✓ 브라우저에서 강제로 로그아웃을 하고 싶을 때는 브라우저 내의 개발자 도구를 사용
  - ✓ 개발자 도구에 있는 Application 탭 메뉴에는 Cookies 항목에서 현재 브라우저가 해당 사이트에서 사용하는 쿠키들을 확인할 수 있음
  - ✓ Tomcat은 JSESSIONID 라는 이름의 쿠키를 사용하므로 해당 쿠키를 삭제하면 로그인 정보는 서버에서 사용할 수 없게 되므로 로그아웃이나 새로운 계정으로 로그인하려는 것과 동일한 효과



Spring Boot Security

- Spring 부트 설정 클래스
  - ✓ Spring 부트가 아닌 Spring만으로 프로젝트를 생성하는 경우에는 web.xml의 설정을 변경하고 복잡한 설정이 필요하지만 Spring 부트는 자동 설정 기능이 있어 별도의 설정 없이도 연동 처리는 되지만 Spring Security를 이용하는 모든 프로젝트는 프로젝트에 맞는 설정을 추가하는 것이 일반적이므로 이를 위한 별도의 Security 설정 클래스를 사용하는 것이 일반적
  - ✓ Security 관련 기능을 쉽게 설정하기 위해서 WebSecurityConfigurerAdapter라는 클래스를 상속받아서 생성
  - ✓ @EnableWebSecurity을 추가하면 SpringSecurityFilterChain이 자동으로 포함되서 WebSecurityConfigurerAdapter 클래스를 상속받아서 메서드 오버라이딩을 통해 보안 설정을 Customizing 할 수 있으며 Bean을 통해서 설정하는 것도 가능

Spring Boot Security

- Spring 부트 설정 클래스
  - ✓ application.yml 파일에 로깅 레벨 설정 추가

```
logging:
  level:
    org:
      hibernate:
        type:
          descriptor:
            sql: trace
      springframework:
        security: trace
    com:
      adamsoft:
        club: debug
```

Spring Boot Security

- 기본 구조 설정
  - ✓ 프로젝트 내에 controller 패키지를 추가하고 SampleController 클래스를 추가

```
@Controller
@Log4j2
@RequestMapping("/sample/")
public class SampleController {

    @GetMapping("/all")
    public void exAll(){
        log.info("exAll.....");
    }

    @GetMapping("/member")
    public void exMember(){
        log.info("exMember.....");
    }

    @GetMapping("/admin")
    public void exAdmin(){
        log.info("exAdmin.....");
    }
}
```

Spring Boot Security

- ❖ 기본 구조 설정
  - ✓ templates 디렉토리에 sample 디렉토리를 추가하고 View 파일을 생성
    - all.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>For All </h1>
</body>
</html>
```

Spring Boot Security

- ❖ 기본 구조 설정
  - ✓ templates 디렉토리에 sample 디렉토리를 추가하고 View 파일을 생성
    - admin.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <h1>For Admin </h1>
</body>
</html>
```

Spring Boot Security

- ❖ 기본 구조 설정
  - ✓ templates 디렉토리에 sample 디렉토리를 추가하고 View 파일을 생성
    - member.html

```
<!DOCTYPE html>
<html xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>For Member .....</h1>

</body>
</html>
```

Spring Boot Security

❖ 기본 구조 설정

✓ URL을 입력하면 로그인 페이지가 화면에 출력되고 로그인을 성공하면 요청한 곳으로 이동

Spring Boot Security

- Spring 부트 Security 설정 클래스
  - ✓ 프로젝트 내에 config 패키지를 추가하고 CustomSecurityConfig 클래스를 추가

```
@Configuration
@Log4j2
@RequiredArgsConstructor
public class CustomSecurityConfig{
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("------------------configure-----------------------------");
        return http.build();
    }
}
```

Spring Boot Security

- Spring 부트 Security 설정 클래스
  - ✓ 다시 실행한 후 기존 URL을 이용해서 접속 – 로그인을 하지 않아도 모두 접속 가능

Spring Boot Security

- Spring 부트 Security 설정 클래스
  - ✓ 정적 파일의 경우 Spring Security가 동작하지 않도록 설정 – CustomSecurityConfig 클래스에 메서드 추가

```
@Bean
public WebSecurityCustomizer webSecurityCustomizer() {
        log.info("------------web configure-------------------");
        return (web) ->
    web.ignoring().requestMatchers(PathRequest.toStaticResources().atCommonLocation
    s());
}
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ PasswordEncoder
    - ● BCryptPasswordEncoder 테스트

```
@SpringBootTest
public class PasswordTests {

    @Test
    public void testEncode() {
        PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        String password = "1111";
        String enPw = passwordEncoder.encode(password);
        System.out.println("enPw: " + enPw);
        boolean matchResult = passwordEncoder.matches(password, enPw);
        System.out.println("matchResult: " + matchResult);
    }
}
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ Security를 위한 UserDetailsService
    - ● 로그인 처리는 일반적으로 회원 아이디와 패스워드로 데이터베이스를 조회하고 올바른 데이터가 있다면 세션이나 쿠키로 처리하는 형태로 제작을 하는 반면에 Spring Security는 기존과는 좀 다른 방식으로 동작
      - ◆ Spring Security에서는 회원이나 계정에 대해서 User라는 용어를 사용
      - ◆ 회원 아이디 라는 용어 대신에 username 이라는 단어를 사용하며 Spring Security에서는 username 이라는 단어 자체가 회원을 구별할 수 있는 식별 데이터를 의미하는데 문자열로 처리하는 점은 같지만 일반적으로 사용하는 회원의 이름이 아니라 id에 해당
      - ◆ username과 password를 동시에 사용하지 않고 Spring Security는 UserDetailsService를 이용해서 username의 존재 만을 확인하고 이후에 password가 틀리면 Bad Credential(잘못된 자격증명) 이라는 결과를 만들어 냄(인증)
      - ◆ 사용자의 username과 password로 인증 과정이 끝나면 원하는 자원(URL)에 접근할 수 있는 적절한 권한이 있는지를 확인하고 인가 과정을 실행하는데 이 과정에서는 Access Denied 와 같은 결과가 만들어 짐

Spring Boot Security

- Spring Security Customizing
  - ✓ HttpSecurity.formLogin(): 인가/인증 절차에서 문제가 발생했을 때 로그인 페이지를 보여주도록 지 정할 수 있고 화면으로 로그인 방식을 지원할 수 있다는 설정으로 연속해서 loginPage("/로그인 URL")을 호출해서 커스텀 로그인 요청을 만들 수 있고 loginProcessingUrl("/로그인 처리 URL")을 호출해서 로그인을 처리하는 URL을 설정할 수 있음
  - ✓ Security를 위한 UserDetailsService
    - UserDetailsService는 loadUserByUsername()이라는 단 하나의 메서드를 가지고 있음
    - loadUserByUsername()은 username이라는 회원 아이디와 같은 식별값으로 회원 정보를 가져옴

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ Security를 위한 UserDetailsService
    - ● CustomSecurityConfigure 클래스의 SecurityFilterChain 메서드 수정

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    log.info("------------------configure-----------------------------");
    http.formLogin();
    return http.build();
}
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ Security를 위한 UserDetailsService
    - ● 프로젝트에 로그인 관련 로직을 처리해주는 클래스를 생성하고 메서드를 오버라이딩
      – security.CustomUserDetailService

```
@Log4j2
@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        log.info("loadUserByUsername: " + username);

        return null;
    }
}
```

Spring Boot Security

- Spring Security Customizing
  - ✓ Security를 위한 UserDetailsService
    - 프로젝트를 실행한 후 login 요청을 하고 콘솔을 확인

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ Security를 위한 UserDetailsService
    - loadUserByUsername 메서드의 리턴 타입은 UserDetails 라는 인터페이스 타입인데 이를 통해서 다음과 같은 정보를 알아낼 수 있도록 구성되어 있음
      - ◆ getAuthorities() - 사용자가 가지는 권한에 대한 정보
      - ◆ getPassword() - 인증을 마무리하기 위한 패스워드
      - ◆ getUsername() - 인증에 필요한 아이디와 같은 정보
      - ◆ 계정 만료 여부 - 사용이 불가능한 계정인지 알 수 있는 정보
      - ◆ 계정 잠김 여부 - 현재 계정의 잠김 여부
    - UserDetails 처리 방식
      - ◆ Spring Security에서 제공하는 UserDetails 인터페이스를 구현한 User라는 클래스 사용
      - ◆ 기존 DTO 클래스에 UserDetails 인터페이스를 구현
      - ◆ DTO와 같은 개념으로 별도의 클래스를 구성하고 이를 활용

Spring Boot Security

- Spring Security Customizing
  - ✓ PasswordEncoder
    - PasswordEncoder는 패스워드를 암호화하는 클래스
    - 이전 Spring 부트에서는 PasswordEncoder를 지정하지 않을 수도 있었지만 Spring 부트 2.0 부터는 인증을 위해서 반드시 PasswordEncoder를 지정해야 함
    - PaswordEncoder는 인터페이스로 설계되어 있으므로 실제 설정에서는 이를 구현하거나 구현된 클래스를 이용해야만 하는데 Spring Security에는 이미 구현된 여러 종류의 PasswordEncoder를 제공하고 있으며 그 중에서도 가장 많이 사용하는 것은 BCryptPasswordEncoder
    - BCryptPasswordEncoder는 bcrypt 라는 해시 함수를 이용해서 패스워드를 암호화하는 목적으로 설계된 클래스
    - BCryptPasswordEncoder로 암호화된 패스워드는 다시 원래대로 복호화가 불가능하고 길이는 같지만 매번 암호화된 값을 다르게 만들어주지만 원본 문자열이 암호화된 결과인지를 확인할 수 는 있음

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 암호화 종류
    - SHA-1(Secure Hash Algorithm 1)
      - ◆ 입력을 받고 메시지 다이제스트라는 160비트(20바이트) 해시값을 만드는 암호화 해시 함수로 보통은 16진수 40자리로 렌더링
      - ◆ 미국 국가안보국이 설계하였으며 미국의 연방 정보 처리 표준
      - ◆ 2005년부터 SHA-1은 충분한 재원이 있는 적들에게 안전하지 않은 것으로 간주
    - SHA-2(Secure Hash Algorithm 2)
      - ◆ digest size는 224, 256, 512 bit로 해시함수로 구성되어졌음.
      - ◆ GPU를 이용한 연산 속도가 매우 빠르기 때문에 password 암호화에 권장되지 않음
      - ◆ GPU 연산 속도가 빠를수록 공격자의 하드웨어를 통한 오프라인 brute force에 더 취약
      - ◆ 빠른 해시를 사용하여 암호화를 진행 시 공격자는 오프라인 공격으로 초당 수십억개의 해시를 계산할 수 있음

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 암호화 종류
    - Bcrypt
      - ◆ Bcrypt는 Blowfish 암호에 기반을 둔 해시 함수로 1999년 USENIX에서 발표되었으며 Bcrypt 함수는 OpenBSD 및 수세 리눅스 등의 일부 리눅스 배포판을 포함한 기타 시스템용 기본 암호 해시 함수
      - ◆ Bcypt 암호화 해시 알고리즘의 큰 특징은 SHA 계열에서 약점으로 지적된 빠른 연산으로 인한 brute force 공격에 대한 대비책으로 Blowfish의 특징 중 하나인 key setup phase 라는 일종의 막대한 전처리 요구(Salting)로 느리게 만들고 반복 횟수(Key Stretching)를 변수로 지정 가능하게 하여 작업량(=해싱시간)을 조절
      - ◆ Bcrypt는 입력 값으로 72 bytes character를 사용해야 하는 단점이 있는데 PBKDF2나 scrypt는 이러한 제약이 없음
      - ◆ 암호화를 위한 BCrypt.hashpw()함수 호출 시 BCrypt.gensalt(10)을 해주는 부분이 있는데 이 부분은 랜덤하게 salt를 만들고 반복 횟수를 2의 10승으로 설정하는 부분
      - ◆ 반복 횟수 지정 없이 BCrypt.gensalt()만 해줄 수도 있는데 그럴 경우 반복 횟수는 기본 값인 10(2의 10승)으로 설정

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 암호화 종류
    - PBKDF2
      - ◆ PBKDF2는 Password-Based Key Derivation Function의 약자로 미국 NIST에서 승인받은 사용자 패스워드를 기반으로 키(Key) 유도를 하기 위한 함수
      - ◆ Bcrypt와 유사하게 사용자 패스워드에 해시 함수, 솔트(Salt), 반복 횟수 등을 지정하여 패스워드에 대한 다이제스트(Digest)를 생성하는 방식
      - ◆ PBKDF2는 아주 가볍고 구현하기 쉬우며 SHA와 같이 검증된 해시 함수 만을 사용
      - ◆ PBKDF2 함수는 다음과 같이 5개 파라미터와 결과값(DIGEST)로 구성

        DIGEST = PBKDF2(PRF, Password, Salt, c, DLen)
        - PRF: 난수(예: HMAC)
        - Password: 패스워드
        - Salt: 패스워드를 암호화하기 위한 키 값
        - c: 원하는 iteration 반복 수
        - DLen: 원하는 다이제스트 길이
      - ◆ Bcrypt와 달리 PBKDF2로 암호화된 패스워드를 복호화 하기 위해서는 DIGEST와 함께 암호화에 사용한 salt 값을 같이 관리해야 함

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 암호화 종류
    - scrypt
      - ◆ scrypt는 PBKDF2와 유사한 adaptive key derivation function으로 Colin Percival이 2012년 9월 17일 설계
      - ◆ scrypt는 DIGEST를 생성할 때 메모리 오버헤드를 갖도록 설계되어, brute force 공격을 시도할 때 병렬화 처리가 매우 어렵게 되어 있어서 PBKDF2보다 안전하다고 평가되며 미래에 bcrypt에 비해 더 경쟁력이 있다고 여겨지지만 그 경우 더욱 많은 메모리를 사용해야 한다는 단점이 존재
      - ◆ scrypt는 보안에 아주 민감한 사용자들을 위한 백업 솔루션을 제공하는 Tarsnap에서도 사용하고 있음
      - ◆ scrypt의 파라미터는 6개의 파라미터와 결과값(DIGEST)로 구성

        DIGEST = scrypt(Password, Salt, N, r, p, DLen)
        - Password: 패스워드
        - Salt: 패스워드를 암호화하기 위한 키값.
        - N: CPU 비용
        - r: 메모리 비용
        - p: 병렬화(parallelization)
        - DLen: 원하는 다이제스트 길이

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ Security를 위한 UserDetailsService
    - ● 로그인 성공을 수행하도록 CustomUserDetailService 클래스 수정

```
@Log4j2
@Service
public class CustomUserDetailsService implements UserDetailsService {
    private PasswordEncoder passwordEncoder;

    public CustomUserDetailsService(){
        this.passwordEncoder = new BCryptPasswordEncoder();
    }
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ Security를 위한 UserDetailsService
    - ● 로그인 성공을 수행하도록 CustomUserDetailService 클래스 수정

```
@Override
public UserDetails loadUserByUsername(String username) throws
 UsernameNotFoundException {
    log.info("loadUserByUsername: " + username);

    UserDetails userDetails = User.builder()
            .username("user1")
            .password(passwordEncoder.encode("1111"))
            .authorities("ROLE_USER")
            .build();

    return userDetails;
  }
}
```

Spring Boot Security

- Spring Security Customizing
  - ✓ Security를 위한 UserDetailsService
    - ● 프로젝트를 다시 실행하고 로그인 시도

Spring Boot Security

- Spring Security Customizing
  - ✓ 인가에 필요한 리소스 설정
    - 인가 설정 방식
      - 어노테이션으로 권한을 설정하려면 설정 관련 클래스에 @EnableGlobalMethodSecurity 어노테이션을 추가해 주어야 함
      - @EnableGlobalMethodSecurity의 prePostEnabled 속성은 원하는 곳에 @PreAuthorize 혹은 @PostAuthorize 어노테이션을 이용해서 사전 혹은 사후의 권한을 체크할 수 있음

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 인가에 필요한 리소스 설정
    - ● /sample/member 요청에 USER 권한을 가진 유저만 접속 가능하도록 설정
      - ◆ CustomSecurityConfig 클래스에 어노테이션 추가

```
@Configuration
@Log4j2
@RequiredArgsConstructor
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class CustomSecurityConfig{
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 인가에 필요한 리소스 설정
    - ● 접속 권한 설정
      - ◆ SampleController 클래스의 요청 처리 메서드 수정

```
@Controller
@Log4j2
@RequestMapping("/sample/")
public class SampleController {

    @GetMapping("/all")
    public void exAll(){
        log.info("exAll.....");
    }
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 인가에 필요한 리소스 설정
    - ● 접속 권한 설정
      - ◆ SampleController 클래스의 요청 처리 메서드 수정

```
    @PreAuthorize("hasRole('USER')")
    @GetMapping("/member")
    public void exMember(){
        log.info("exMember.....");
    }

    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("/admin")
    public void exAdmin(){
        log.info("exAdmin.....");
    }
}
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 인가에 필요한 리소스 설정
    - ● 접속 권한 설정
      - ◆ 실행 한 후 브라우저에서 member 와 admin 요청을 수행

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 인가에 필요한 리소스 설정
    - ● @PreAuthorize/@PostAuthorize 접근 제한 표현식
      - ◆ authenticated(): 인증된 사용자만 접근
      - ◆ permitAll(): 모두 허용
      - ◆ anonymous(): 익명의 사용자 허용
      - ◆ hasRole(표현식): 특정한 권한이 있는 사용자만 허용
      - ◆ hasAnyRole(표현식): 여러 권한 중 하나만 존재해도 허용

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 인증 설정 방식
    - ● SecurityConfigure 클래스의 메서드를 수정해서 설정
      - ◆ formLogin(): 로그인 폼 사용
      - ◆ loginPage(): 로그인 페이지 URL 설정
      - ◆ defaultSuccessUrl(): 로그인 성공했을 때 이동할 URL을 설정
      - ◆ usernameParameter(): 로그인 시 사용할 파라미터 이름 설정
      - ◆ failureUrl(): 로그인 실패 시 이동할 URL을 설정
      - ◆ logoutRequestMatcher(): 로그아웃 URL을 설정
      - ◆ logoutSuccessUrl(): 로그아웃 성공 시 이동할 URL을 설정

Spring Boot Security

- Spring Security Customizing
  - ✓ 인증 설정 방식 – 커스텀 로그인 페이지 설정
    - CustomSecurityConfig 클래스의 메서드 수정

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("------------------configure-----------------------------");
        http.formLogin().loginPage("/member/login");
        return http.build();
}
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 인증 설정 방식
    - controller 패키지에 MemberController 클래스를 생성하고 작성

```
@Controller
@Log4j2
@RequestMapping("/member")
@RequiredArgsConstructor
public class MemberController {
    @GetMapping("/login")
    //로그인 과정에 문제가 발생하거나 로그아웃 처리할 때 사용하기 위해서
     파라미터를 설정
    public void login(String error, String logout) {
        log.info("error: " + error);
        log.info("logout: " + logout);
    }
}
```

Spring Boot Security

- Spring Security Customizing
  - ✓ 인증 설정 방식
    - templates/member 디렉토리에 login.html 파일을 생성

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Custom Login Page</title>
</head>
```

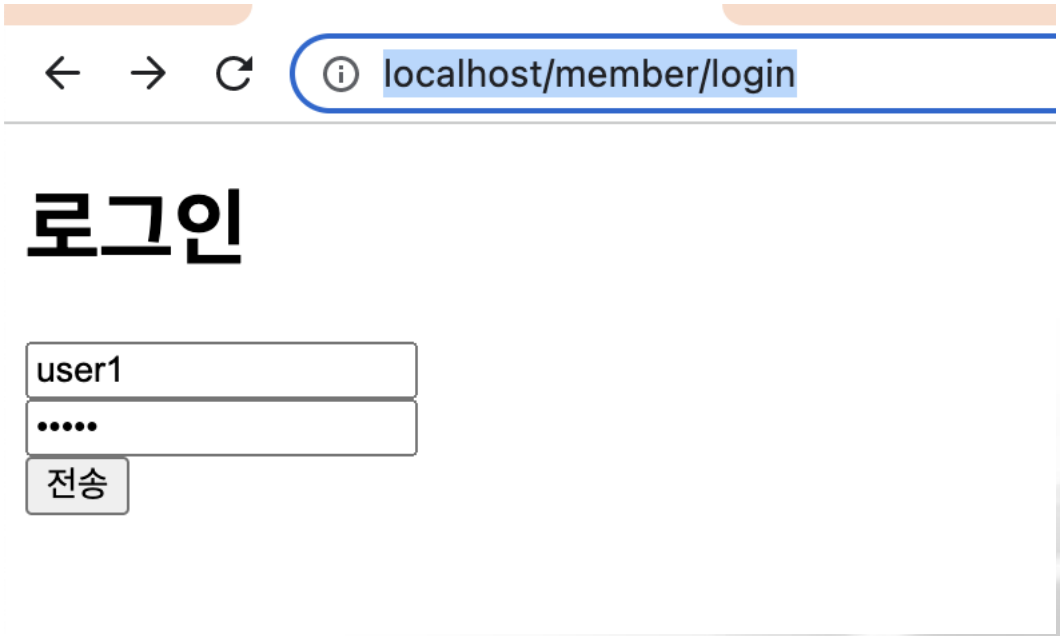
Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 인증 설정 방식
    - templates/member 디렉토리에 login.html 파일을 생성

```
<body>
<h1>로그인</h1>
<form method='post' >
    <div>
        <input type='text' name='username' value='admin'>
    </div>
    <div>
        <input type='password' name='password' value='admin'>
    </div>
    <div>
        <input type='submit' value="전송">
    </div>
</form>
</body>
</html>
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 인증 설정 방식
    - ● 실행한 후 로그인을 시도하면 에러



Spring Boot Security

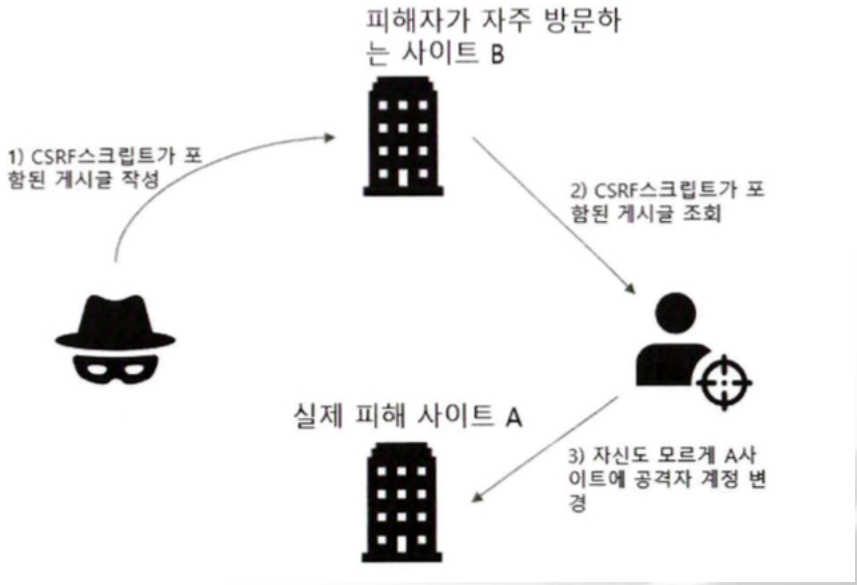
- ❖ Spring Security Customizing
  - ✓ CSRF(Cross Site Request Forgery-크로스 사이트 요청 위조) 토큰 비활성화
    - ● CSRF 공격
      - ◆ 사용자의 등급을 변경하는 URI를 알고 이 때 필요한 파라미터를 안다면 직접 로그인을 하지 않고 img 태그 나 form을 이용해서 URI 와 파라미터를 기록해 둔 상태에서 관리자가 이 링크를 클릭하게 되면 공격자가 관리자 등급으로 변경됨

        www.aaa.xxx?update?grade=admin&account= 123

        <form action='www.aaa.xxx?update? grade=admin&account=123#> <input type='submit value='축 이벤트 당첨'>
        </form>
        혹은
        <img src='www.aaa«xxx?update? grade=admin&account=123r>

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ CSRF(Cross Site Request Forgery-크로스 사이트 요청 위조) 토큰 비활성화
    - ● CSRF 공격



Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ CSRF(Cross Site Request Forgery-크로스 사이트 요청 위조) 토큰 비활성화
    - ● CSRF 공격 방어 방법
      - ◆ 사용자의 요청에 대한 출처를 의미하는 referrer 헤더를 체크
      - ◆ PUT 이나 DELETE 같은 방식을 이용
      - ◆ CSRF 토큰을 이용해서 해결

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ CSRF(Cross Site Request Forgery-크로스 사이트 요청 위조) 토큰 비활성화
    - ● CSRF 토큰
      - ◆ 사용자가 임의로 변하는 특정한 토큰 값을 서버에서 체크하는 방식
      - ◆ 서버에서 브라우저에 데이터를 전송할 때 CSRF 토큰을 같이 전송하고 사용자가 POST 방식으로 작업을 할 때 브라우저에서 전송된 CSRF 토큰 값과 서버가 보관하고 있는 CSRF 토큰의 값을 비교해서 이 값이 다르면 작업을 처리하지 않는 방식
      - ◆ Session에 보관해서 사용
      - ◆ CSRF 토큰은 Spring Security가 적용된 사이트의 GET 방식을 제외한 모든 요청 방식(POST, PUT, DELETE) 등에 포함시켜야만 정상적인 동작이 가능
      - ◆ 일반적인 세션을 이용하고 <form> 태그를 이용하는 방식에서는 CSRF 토큰이 보안상으로 권장되지만 REST 방식 등에서 매번 CSRF 토큰의 값을 알아내야 하는 불편함이 있기 때문에 경우에 따라서는 CSRF 토큰의 발행을 하지 않는 경우도 있음

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ CSRF(Cross Site Request Forgery-크로스 사이트 요청 위조) 토큰 비활성화
    - ● CSRF 토큰 비활성화
      - ◆ CustomSecurityConfig 클래스의 configure 메서드 수정

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("------------------configure-----------------------------");
        http.formLogin().loginPage("/member/login");
        http.csrf().disable();
        return http.build();
}
```

Spring Boot Security

- Spring Security Customizing
  - ✓ logout 설정
    - formLogin()과 마찬가지로 logout() 메서드를 이용하면 로그아웃 처리 가능
    - logout() 역시 로그인과 마찬가지로 별도의 설정이 없는 경우에는 Spring Security가 제공하는 웹 페이지를 보게 됨
    - 로그아웃은 세션을 유지하는데 사용하는 쿠키(JSESSIONID)를 삭제하면 자동으로 로그아웃 처리가 됨
    - 기본적으로 /logout 이라는 경로를 제공해서 CSRF 토큰이 비활성화 되는 경우에는 GET 방식으로도 로그아웃이 가능

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ logout 설정
    - ● MemberController 클래스의 login 처리 메서드 수정

```
public void login(String error, String logout) {
        log.info("error: " + error);
        log.info("logout: " + logout);

        if(logout != null){
            log.info("user logout.....");
        }
    }
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ logout 설정
    - ● login.html 파일에 메시지 출력 코드 추가

```
<th:block th:if="${param.logout != null}">
    <h2>Logout...</h2>
</th:block>
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ logout 설정
    - ● 애플리케이션을 실행한 후 login을 하고 logout을 호출

Spring Boot Security

- Spring Security Customizing
  - ✓ 자동 로그인
    - 쿠키를 이용해서 브라우저에 로그인 했던 정보를 유지하는 방식으로 구현
    - 쿠키에 유효 기간을 지정해서 쿠키를 브라우저가 보관하게 하고 쿠키의 값인 특정한 문자열을 보관시켜서 로그인 관련 정보를 유지하는 방식
    - 현재 프로젝트가 사용하는 데이터베이스에 persistent_logins 라는 테이블을 추가해야 함

```
create table persistent_logins(
    username varchar(64) not null,
    series varchar(64) primary key,
    token varchar(64) not null,
    last_used timestamp not null
);
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 자동 로그인
    - ● CustomSecurityConfig 클래스 수정

```
@Configuration
@Log4j2
@RequiredArgsConstructor
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class CustomSecurityConfig{
    private final DataSource dataSource;
    private final CustomUserDetailsService userDetailsService;

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

    @Bean
    public PersistentTokenRepository persistentTokenRepository() {
        JdbcTokenRepositoryImpl repo = new JdbcTokenRepositoryImpl();
        repo.setDataSource(dataSource);
        return repo;
    }
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 자동 로그인
    - ● CustomSecurityConfig 클래스 수정

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    log.info("------------------configure-----------------------------");
    http.formLogin().loginPage("/member/login");
    http.csrf().disable();

    http.rememberMe()
            .key("12345678")
            .tokenRepository(persistentTokenRepository())
            .userDetailsService(userDetailsService)
            .tokenValiditySeconds(60*60*24*30);

    return http.build();
}
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 자동 로그인
    - ● CustomSecurityConfig 클래스 수정

```
@Bean
public WebSecurityCustomizer webSecurityCustomizer() {
    log.info("------------web configure------------------");
    return (web) ->
 web.ignoring().requestMatchers(PathRequest.toStaticResources().atCommonLoc
 ations());
}
}
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 자동 로그인
    - ● login.html 파일에 자동 로그인을 위한 화면 설정

```
<form method='post'>
    <div>
        <input type='text' name='username' value='admin'>
    </div>
    <div>
        <input type='password' name='password' value='admin'>
    </div>
    <div>
        <input type='checkbox' name="remember-me">
        <label>자동 로그인</label>
    </div>
    <div>
        <input type='submit' value="전송">
    </div>
</form>
```

Spring Boot Security

- Spring Security Customizing
  - ✓ 자동 로그인

	username	series	token	last_used
1	user1	LigYOSFbit1/W	Ckg+pRvjD7b	1-16 07:47:01.000

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 화면에서 인증 처리 과 컨트롤러
    - ● 화면 상에서 로그인 혹은 특정 권한 별 제어
      - ◆ Thymeleaf 에서 인증 정보 활용 – member.html 파일 수정

```
<!DOCTYPE html>
<html xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<div>
<span>사용자</span>
<input type="text" name="username"
th:value="${#authentication.principal.username}" readonly>
</div>

<h1>For Member .....</h1>

</body>
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 화면에서 인증 처리 과 컨트롤러
    - ● 화면 상에서 로그인 혹은 특정 권한 별 제어
      - ◆ Thymeleaf 에서 인증 정보 활용 – member.html 파일 수정

```
<script th:inline="javascript">
  const auth = [[${#authentication.principal}]];
  console.log(auth);
  const errors = [[${errors}]] ;
  console.log(errors);
</script>
</html>
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ Controller에서 로그인 정보 확인
    - ● MemberController 클래스의 메서드 수정

```
@PreAuthorize("hasRole('USER')")
@GetMapping("/member")
    public void exMember(){
        log.info("exMember.....");
        Authentication authentication =
     SecurityContextHolder.getContext().getAuthentication();
        String currentPrincipalName = authentication.getName();
        log.warn(currentPrincipalName);
}
```

Spring Boot Security

- Spring Security Customizing
  - ✓ 403 에러 처리
    - 403(Forbidden) 에러는 서버에서 사용자의 요청을 거부했다는 의미로 대부분 사용자가 로그인은 되었지만 해당 작업을 수행할 수 없는 경우에 해당
    - 스프링 시큐리티에서 @PreAuthorize(UisAuthenticated()")인 경우 사용자 로그인이 안 되었다면 302 메시지와 함께 로그인 경로로 이동하지만 403 에러는 에러가 발생
    - 403 에러가 발생했을 때 에러 페이지를 보여주는 방식 대신에 AccessDeniedHandler 인터페이스를 구현해서 상황에 맞게 처리해야 함

Spring Boot Security

- Spring Security Customizing
  - ✓ 403 에러 처리
    - 프로젝트에 403 에러 발생 시 처리할 핸들러 클래스 생성

```
@Log4j2
public class Custom403Handler implements AccessDeniedHandler {
    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
     AccessDeniedException accessDeniedException) throws IOException,
     ServletException {
        log.info("--------ACCESS DENIED--------------");
        response.sendRedirect("/member/login?error=ACCESS_DENIED");
    }
}
```

Spring Boot Security

- ❖ Spring Security Customizing
  - ✓ 403 에러 처리
    - ● CustomSecurityConfig 클래스의 메서드 수정

```
@Bean
public AccessDeniedHandler accessDeniedHandler() {
    return new Custom403Handler();
}

@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("------------------configure-----------------------------");
        http.formLogin().loginPage("/member/login");
        http.csrf().disable();
        http.rememberMe()
                .key("12345678")
                .tokenRepository(persistentTokenRepository())
                .userDetailsService(userDetailsService)
                .tokenValiditySeconds(60*60*24*30);

        http.exceptionHandling().accessDeniedHandler(accessDeniedHandler());
        return http.build();
}
```

Spring Boot Security

- Spring Security JPA
  - ✓ 회원 정보 - ClubMember
    - 회원 아이디(mid)
    - 패스워드(mpw)
    - 이메일(email)
    - 이름(name)
    - 소셜 가입 여부(social)
    - 탈퇴 여부(del)
    - 기타(등록일/수정일)
  - ✓ 권한 – ClubMemberRole
    - USER: 일반회원
    - ADMIN: 총괄 관리자

Spring Boot Security

- Spring Security JPA
  - ✓ 공통 속성을 가진 Entity 생성 - model.BaseEntity

```
@MappedSuperclass
@EntityListeners(value = { AuditingEntityListener.class })
@Getter
abstract class BaseEntity {

    @CreatedDate
    @Column(name = "regdate" , updatable = false)
    private LocalDateTime regDate;

    @LastModifiedDate
    @Column(name ="moddate" )
    private LocalDateTime modDate;

}
```

Spring Boot Security

- Spring Security JPA
  - ✓ 권한을 나타낼 상수들을 위한 enum을 생성 – model.ClubMemberRole

```
public enum  ClubMemberRole {
    USER, ADMIN
}
```

Spring Boot Security

- Spring Security JPA
  - ✓ 회원 정보를 위한 entity를 생성 – model.ClubMember

```
@Entity
@Getter
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString(exclude = "roleSet")
public class ClubMember extends BaseEntity{

    @Id
    private String mid;

    private String mpw;
    private String email;
    private String name;
    private boolean del;
    private boolean social;

    @ElementCollection(fetch = FetchType.LAZY)
    @Builder.Default
    private Set<ClubMemberRole> roleSet = new HashSet<>();
```

Spring Boot Security

- Spring Security JPA
  - ✓ 회원 정보를 위한 entity를 생성 – model.ClubMember

```
    public void changePassword(String mpw ){
        this.mpw = mpw;
    }

    public void changeEmail(String email){
        this.email = email;
    }

    public void changeDel(boolean del){
        this.del = del;
    }

    public void addRole(ClubMemberRole memberRole){
        this.roleSet.add(memberRole);
    }

    public void clearRoles() {
        this.roleSet.clear();
    }

    public void changeSocial(boolean social){this.social = social;}
}
```

Spring Boot Security

- Spring Security JPA
  - ✓ ClubMember Entity의 데이터베이스 작업을 위한 Repository 인터페이스를 생성 – persistence.ClubMemberRepository

```
public interface ClubMemberRepository extends JpaRepository<ClubMember, String> {
    @EntityGraph(attributePaths = "roleSet")
    @Query("select m from ClubMember m where m.mid = :mid and m.social = false")
    Optional<ClubMember> getWithRoles(String mid);
}
```

Spring Boot Security

- Spring Security JPA
  - ✓ 테스트 코드를 이용해서 100명의 회원 추가

```
@SpringBootTest
public class ClubMemberTests {

    @Autowired
    private ClubMemberRepository clubMemberRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;
```

Spring Boot Security

- Spring Security JPA
  - ✓ 테스트 코드를 이용해서 100명의 회원 추가

```
@Test
public void insertMembers() {
    IntStream.rangeClosed(1,100).forEach(i -> {
        ClubMember clubMember = ClubMember.builder()
                .mid("member" + i)
                .mpw(  passwordEncoder.encode("1111") )
                .email("user"+i+"@gmail.com")
                .name("사용자"+i)
                .social(false)
                .roleSet(new HashSet<ClubMemberRole>())
                .build();
        System.out.println(clubMember);
        //default role
        clubMember.addRole(ClubMemberRole.USER);
        if(i > 90){
            clubMember.addRole(ClubMemberRole.ADMIN);
        }
        clubMemberRepository.save(clubMember);
    });
}
}
```

Spring Boot Security

- Spring Security JPA
  - ✓ 회원 정보 확인

```
@Test
public void testRead() {
        Optional<ClubMember> result =
clubMemberRepository.getWithRoles("member100");
        ClubMember member = result.orElseThrow();
        System.out.println(member);
        System.out.println(member.getRoleSet());
}
```

Spring Boot Security

- Spring Security JPA
  - ✓ 회원 데이터 조회
    - ClubMemberRepository 인터페이스에 이메일을 기준으로 조회하는 메서드 생성

```
@EntityGraph(attributePaths = {"roleSet"}, type = EntityGraph.
EntityGraphType.LOAD)
@Query("select m from ClubMember m where m.email =:email")
Optional<ClubMember> findByEmail(@Param("email") String emai);
```

Spring Boot Security

- Spring Security JPA
  - ✓ 회원 데이터 조회
    - ClubMemberRepository 인터페이스의 메서드 테스트

```
@Test
public void testReadEmail() {
        Optional<ClubMember> result =
clubMemberRepository.findByEmail("user95@gmail.com");
        ClubMember clubMember = result.get();
        System.out.println(clubMember);
}
```

Spring Boot Security

- ❖ Spring Security JPA
  - ✓ Security를 위한 UserDetailsService
    - ● security.dto 패키지에 ClubMemberSecurityDTO 클래스를 생성

```
@Getter
@Setter
@ToString
public class ClubMemberSecurityDTO extends User {
    private String mid;
    private String mpw;
    private String email;
    private String name;
    private boolean del;
    private boolean social;
```

Spring Boot Security

- ❖ Spring Security JPA
  - ✓ Security를 위한 UserDetailsService
    - ● security.dto 패키지에 ClubMemberSecurityDTO 클래스를 생성

```
public ClubMemberSecurityDTO(String username, String password, String
email, String name, boolean del, boolean social, Collection<? extends
GrantedAuthority> authorities) {
        super(username, password, authorities);
        this.mid = username;
        this.mpw = password;
        this.email = email;
        this.name = name;
        this.del = del;
        this.social = social;
    }
}
```

Spring Boot Security

- ❖ Spring Security JPA
  - ✓ Security를 위한 UserDetailsService 구현
    - ● CustomUserDetailsService 클래스를 수정

```
@Log4j2
@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {
    private final ClubMemberRepository clubMemberRepository;


    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        log.info("loadUserByUsername: " + username);

        Optional<ClubMember> result =
clubMemberRepository.getWithRoles(username);

        if(result.isEmpty()){
            throw new UsernameNotFoundException("없는 사용자 이름...");
        }
```

Spring Boot Security

- ❖ Spring Security JPA
  - ✓ Security를 위한 UserDetailsService 구현
    - ● CustomUserDetailsService 클래스를 수정

```
        ClubMember member = result.get();

        ClubMemberSecurityDTO clubMemberSecurityDTO =
                new ClubMemberSecurityDTO(member.getMid(), member.getMpw(),
member.getEmail(),
                        member.getName(), member.isDel(), false,
                        member.getRoleSet().stream().map(memberRole ->
                                new SimpleGrantedAuthority("ROLE_" +
memberRole.name()))
                                .collect(Collectors.toList()));
        return clubMemberSecurityDTO;
    }
}
```

Spring Boot Security

- Spring Security JPA
  - ✓ Security를 위한 UserDetailsService 구현
    - 애플리케이션을 실행하고 데이터베이스의 정보를 이용해서 로그인

Spring Boot Security

- Spring Security JPA
  - ✓ 로그인 한 유저 정보 출력
    - member.html 파일 수정

```
<!DOCTYPE html>
<html xmlns:sec="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>For Member .....</h1>
<div sec:authorize="hasRole('USER')">Has USER ROLE</div>
<div sec:authorize="hasRole('MANAGER')">Has MANAGER ROLE</div>
<div sec:authorize="hasRole('ADMIN')">Has ADMIN ROLE</div>

<div sec:authorize="isAuthenticated()">
    로그인 성공한 유저에게만 보여지는 텍스트
</div>
```

Spring Boot Security

- ❖ Spring Security JPA
  - ✓ 로그인 한 유저 정보 출력
    - member.html 파일 수정

```
Authenticated username:
<div sec:authentication="name"></div>
Authenticated user roles:
<div sec:authentication="principal.authorities">
</div>

<a href="/logout">로그아웃</a>
</body>
</html>
```

Spring Boot Security

- ❖ Spring Security JPA
  - ✓ Controller 에서의 유저 정보 출력
    - 방법
      - ◆ SecurityContextHollder라는 객체 사용
      - ◆ @AuthenticationPrincipal 어노테이션 사용
    - SampleController 클래스의 메서드 수정

```
@PreAuthorize("hasRole('USER')")
@GetMapping("/member")
public void exMember(@AuthenticationPrincipal ClubMemberSecurityDTO
    clubAuthMember){
        log.info("exMember.....");
        Authentication authentication =
    SecurityContextHolder.getContext().getAuthentication();
        String currentPrincipalName = authentication.getName();
        log.warn(currentPrincipalName);
        log.info(clubAuthMember);
    }
```

Spring Boot Security

- 회원 가입
  - ✓ 회원 가입 DTO 생성 – ClubMemberJoinDTO

```
@Data
public class ClubMemberJoinDTO {
    private String mid;
    private String mpw;
    private String email;
    private String name;
    private boolean del;
    private boolean social;
}
```

Spring Boot Security

- 회원 가입
  - ✓ 회원 가입을 서비스 인터페이스 생성 – service.MemberService

```
public interface MemberService {
    static class MidExistException extends Exception {
    }

    void join(ClubMemberJoinDTO memberJoinDTO)throws MidExistException ;
}
```

Spring Boot Security

- 회원 가입
  - ✓ 회원 가입을 서비스 클래스 생성 – service.MemberServiceImpl

```
@Log4j2
@Service
@RequiredArgsConstructor
public class MemberServiceImpl implements MemberService{

    private final ClubMemberRepository memberRepository;

    private final PasswordEncoder passwordEncoder;
```

Spring Boot Security

- 회원 가입
  - ✓ 회원 가입을 서비스 클래스 생성 – service.MemberServiceImpl

```
@Override
public void join(ClubMemberJoinDTO memberJoinDTO) throws MidExistException{

    String mid = memberJoinDTO.getMid();

    boolean exist = memberRepository.existsById(mid);

    if(exist){
        throw new MidExistException();
    }
```

Spring Boot Security

- 회원 가입
  - ✓ 회원 가입을 서비스 클래스 생성 – service.MemberServiceImpl

```
        ClubMember member = ClubMember.builder()
                .mid(memberJoinDTO.getMid())
                .mpw(memberJoinDTO.getMpw())
                .email(memberJoinDTO.getEmail())
                .name(memberJoinDTO.getName())
                .del(memberJoinDTO.isDel())
                .social(memberJoinDTO.isSocial())
                .build();
        log.warn(member);
        member.changePassword(passwordEncoder.encode(memberJoinDTO.getMpw()));
        member.addRole(ClubMemberRole.USER);
        log.warn(member);
        log.info("======================");
        log.info(member);
        log.info(member.getRoleSet());

        memberRepository.save(member);

    }
}
```

Spring Boot Security

- 회원 가입
  - ✓ MemberController 클래스에 회원 가입을 위한 요청을 처리하는 메서드를 작성

```
private final MemberService memberService;

@GetMapping("/join")
    public void join(){
        log.info("join get...");
}
```

Spring Boot Security

- 회원 가입
  - ✓ MemberController 클래스에 회원 가입을 위한 요청을 처리하는 메서드를 작성

```
@PostMapping("/join")
public String joinPOST(ClubMemberJoinDTO memberJoinDTO, RedirectAttributes
    redirectAttributes){

        log.info("join post...");
        log.info(memberJoinDTO);

        try {
            memberService.join(memberJoinDTO);
        } catch (MemberService.MidExistException e) {

            redirectAttributes.addFlashAttribute("error", "mid");
            return "redirect:/member/join";
        }

        redirectAttributes.addFlashAttribute("result", "success");

        return "redirect:/member/login"; //회원 가입 후 로그인
}
```

Spring Boot Security

- 회원 가입
  - ✓ member 디렉토리에 join.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity4"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">

<head>
  <title>회원 가입</title>
</head>
```

Spring Boot Security

- 회원 가입
  - ✓ member 디렉토리에 join.html 파일을 생성하고 작성

```
<div>
        <div class="card-header">
          회원 가입
        </div>
          <form id="registerForm" action="/member/join" method="post">
            <div>
              <span>MID</span>
              <input type="text" name="mid">
            </div>

            <div>
              <span>MPW</span>
              <input type="password" name="mpw">
            </div>

            <div>
              <span >EMAIL</span>
              <input type="email" name="email">
            </div>
```

Spring Boot Security

- 회원 가입
  - ✓ member 디렉토리에 join.html 파일을 생성하고 작성

```
<div>
  <span >NAME</span>
  <input type="text" name="name">
</div>

<div>
  <div class="float-end">
    <button type="submit">회원가입</button>
    <button type="reset">폼초기화</button>
  </div>
</div>
</form>
</div>
```

Spring Boot Security

- 회원 가입
  - ✓ member 디렉토리에 join.html 파일을 생성하고 작성

```
<script   th:inline="javascript">

  const error = [[${error}]]

  if(error && error === 'mid'){
    alert("동일한 MID를 가진 계정이 존재합니다.")
  }

</script>
```

Spring Boot Security

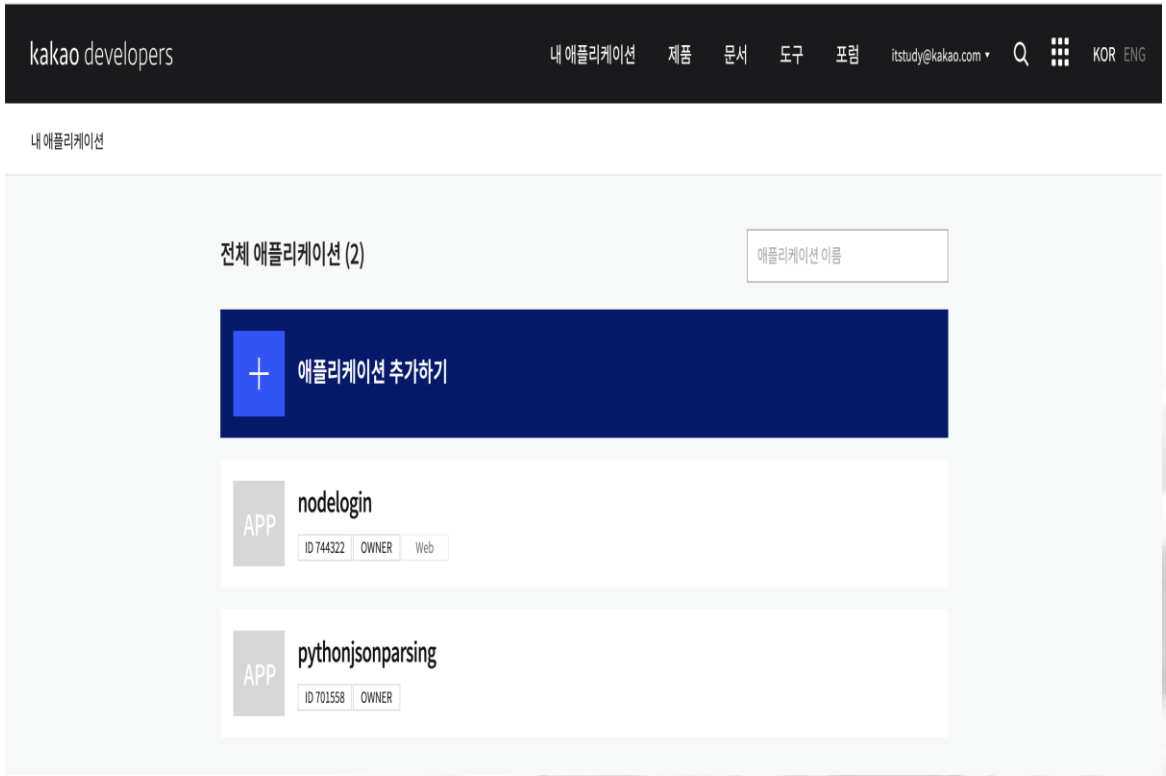
- OAuth
  - ✓ 웹을 사용하면서 겪는 여러 불편함이 있지만 가장 많이 겪는 어려움은 매번 비밀번호를 기억해야 한다는 점인데 이러한 불편함을 해소하기 위해서 기존 서비스의 인증을 사용하는 방식을 소셜 로그인 이라고 함
  - ✓ 서비스를 제공하는 업체들이 각자 다른 방식으로 로그인하지 않도록 공통의 인증 방식을 제공하는데 이를 OAuth(Open Authorization)라고 함
  - ✓ OAuth를 제공하는 서비스 업체들을 이용해서 로그인을 처리하면 사용자 관리에 대한 부담을 줄일 수 있음

Spring Boot Security

- ❖ Kakao Login
  - ✓ 카카오 클라이언트 아이디 발급
    - ● 카카오 개발자 페이지에 로그인: https://developers.kakao.com/

Spring Boot Security

- Kakao Login
  - ✓ 카카오 클라이언트 아이디 발급
    - 애플리케이션 추가



Spring Boot Security

- ❖ Kakao Login
  - ✓ 카카오 클라이언트 아이디 발급
    - ● 애플리케이션 추가

Spring Boot Security

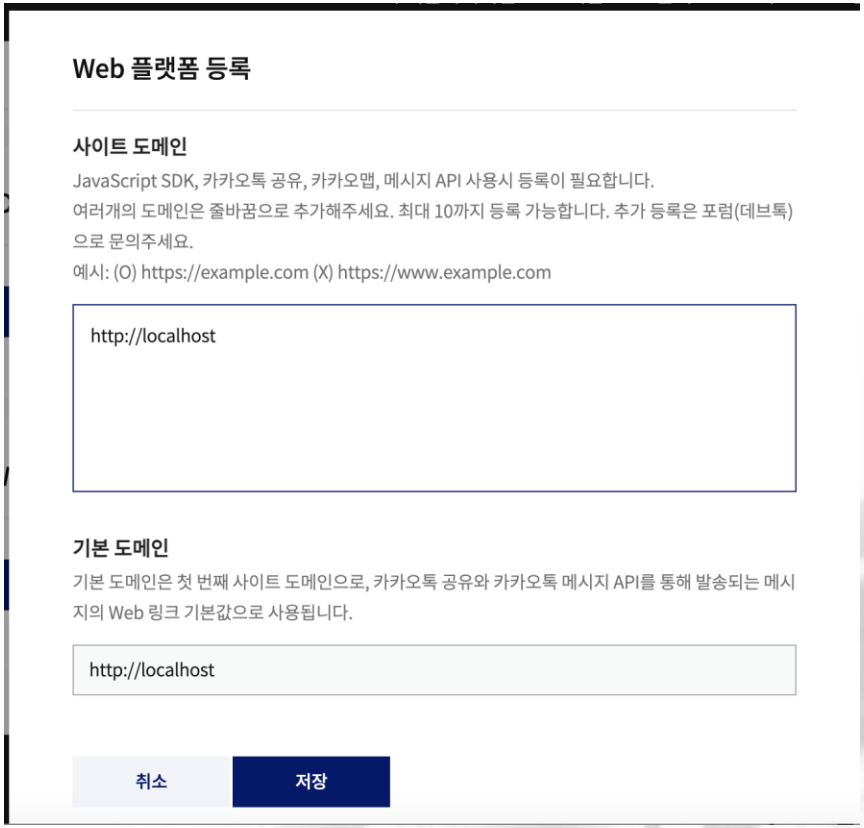
- ❖ Kakao Login
  - ✓ 카카오 클라이언트 아이디 발급
    - ● REST API 키 복사

앱 키

네이티브 앱 키	1c27cd7585258f6fee08e0748a5b95ae
REST API 키	6a92e72defaeb1bd45b361490ae7f2b8
JavaScript 키	001fb4c571ae0f8a89b993d5aa1c5bbb
Admin 키	4672367f605b189d64c90c7990142cf6

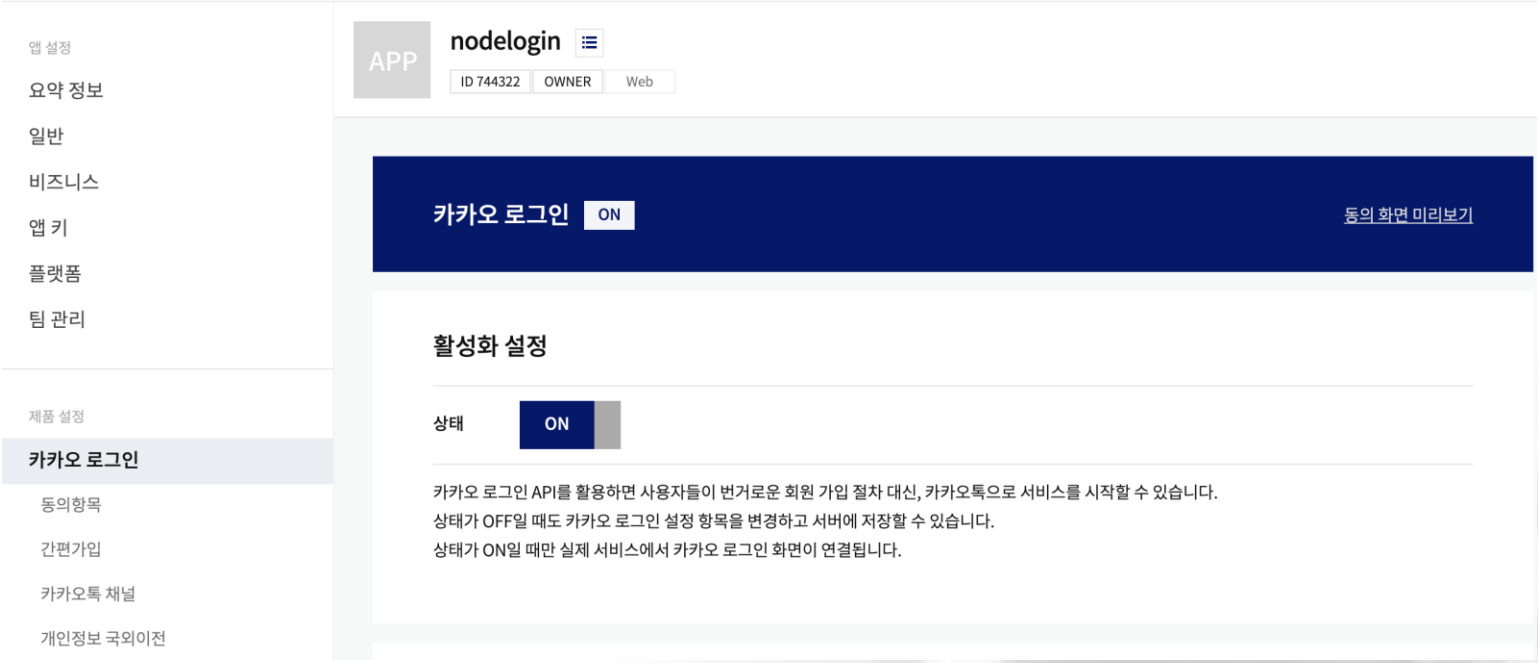
Spring Boot Security

- ❖ Kakao Login
  - ✓ 카카오 클라이언트 아이디 발급
    - ● 플랫폼 등록: 사용할 도메인을 등록



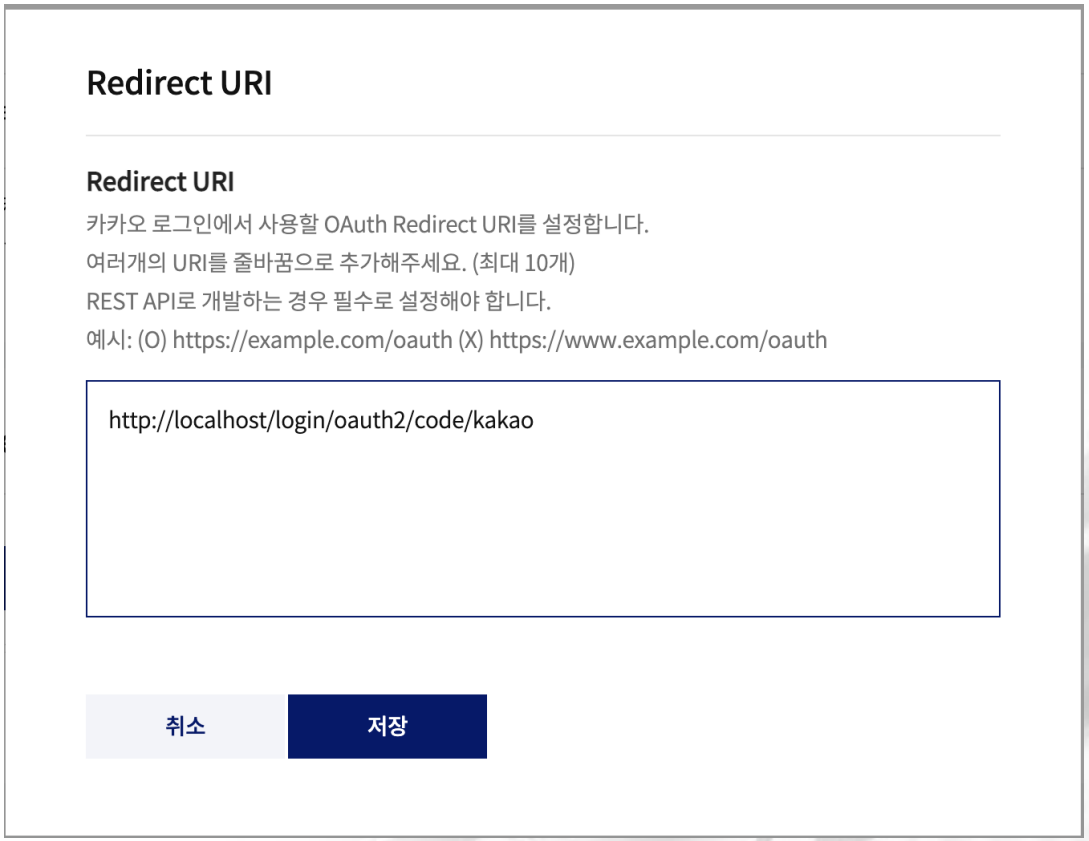
Spring Boot Security

- Kakao Login
  - ✓ 카카오 클라이언트 아이디 발급
    - 로그인 활성화



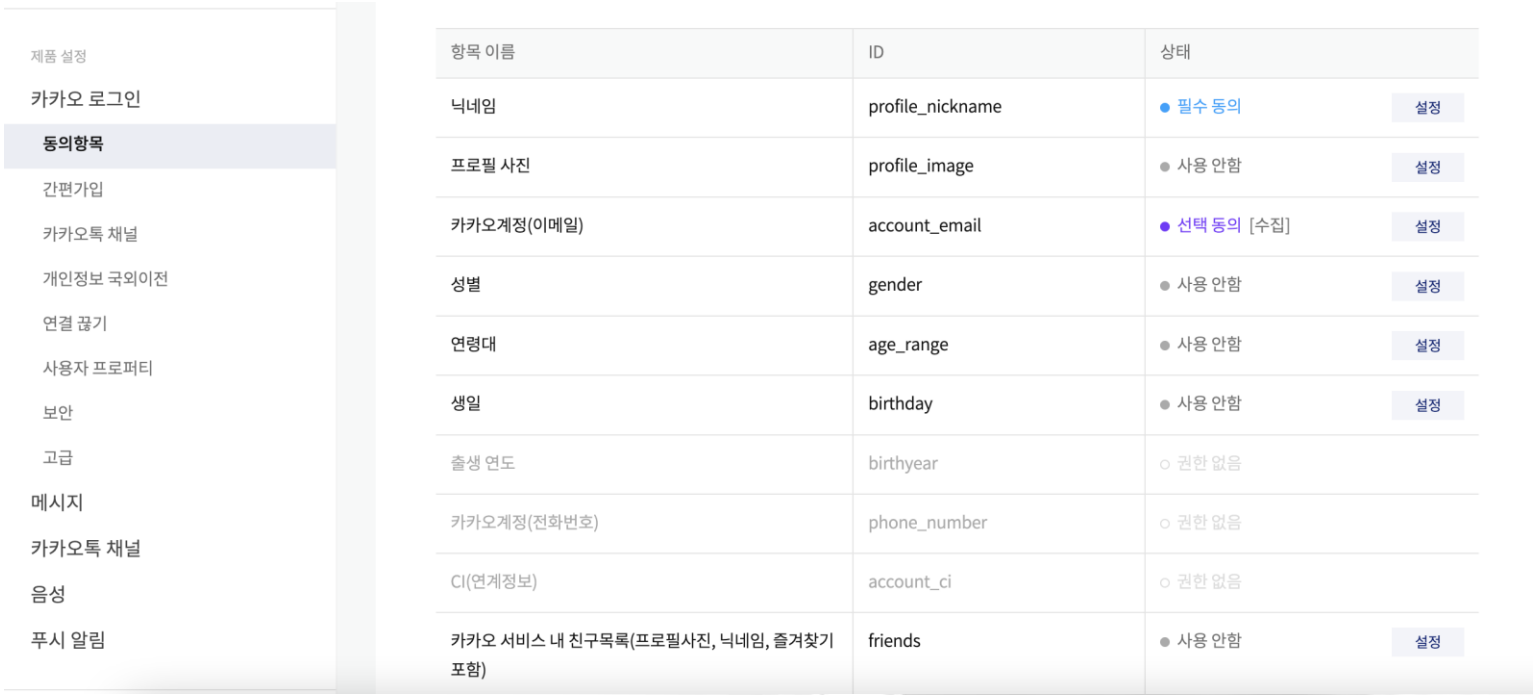
Spring Boot Security

- ❖ Kakao Login
  - ✓ 카카오 클라이언트 아이디 발급
    - Redirect URI 등록



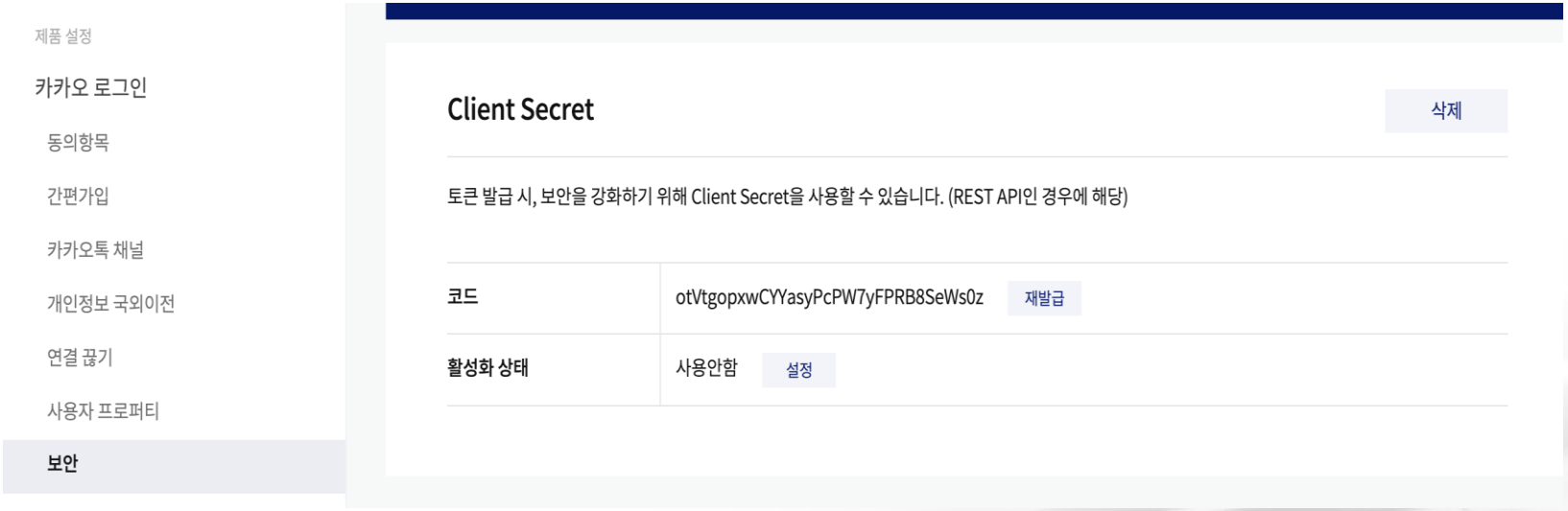
Spring Boot Security

- ❖ Kakao Login
  - ✓ 카카오 클라이언트 아이디 발급
    - ● 동의 항목 설정



Spring Boot Security

- ❖ Kakao Login
  - ✓ 카카오 클라이언트 아이디 발급
    - ● Client Secret 키 발급

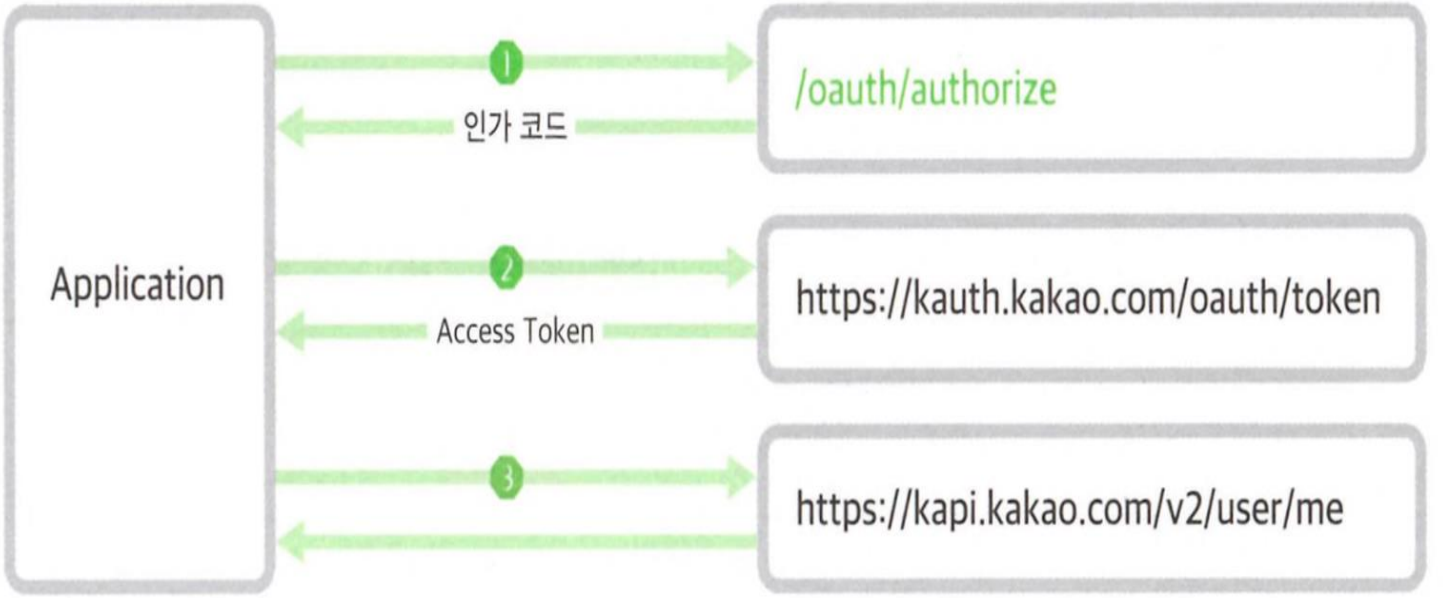


Spring Boot Security

- ❖ Kakao Login
  - ✓ build.gradle 파일의 dependencies에 OAuth 라이브러리 추가
    - implementation 'org.springframework.boot:spring-boot-starter-oauth2-client'

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 처리 과정
    - 소셜 로그인은 OAuth2(https://oauth.net/2/) 방식으로 데이터를 처리하는데 문자열로 구성된 Token 을 주고 받는 방식으로 서비스 간 데이터를 교환
    - 설정할 때 받은 REST API 키를 이용해서 인가 코드를 받음



Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 처리 과정
    - Request

URL

```
GET /oauth/authorize?client_id=${REST_API_KEY}&redirect_uri=${REDIRECT_URI}&response_ty
Host: kauth.kakao.com
```

- ◆ 인가 코드는 리다이렉트 URI 로 지정된 곳으로 전달되는데 스프링 부트의 OAuth2 Client를 이용하는 경우에는 리다이렉트 URI 는 정해진 패턴을 사용
- ◆ 1에서 받은 인가 코드는 2에서 자신의 비밀키(Client Secret)와 같이 이용되어 Access Token을 생성할 때 사용

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 처리 과정
    - ● Request

Request

URL

```
POST /oauth/token HTTP/1.1
Host: kauth.kakao.com
Content-type: application/x-www-form-urlencoded;charset=utf-8
```

- ◆ Access Token은 원하는 데이터에 접근할 수 있는 권한 역할을 하는데 Access Token이 외부에 탈취당하면 위험하기 때문에 유효 기간은 짧게 설정하는 것이 일반적

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 처리 과정
    - Access Token을 얻으면 이를 이용해서 사용자 정보를 요청하는데 사용자가 동의했던 정보들을 얻어오는데 주로 이메일을 얻어오는 경우가 대부분

Request: 액세스 토큰 사용

URL

```
GET/POST /v2/user/me HTTP/1.1
Host: kapi.kakao.com
Authorization: Bearer ${ACCESS_TOKEN}
Content-type: application/x-www-form-urlencoded;charset=utf-8
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 설정
    - ● application.yml 에 로그인 설정 관련 코드를 추가

```
spring:
  security:
    oauth2:
      client:
        registration:
          kakao:
            client-id: df1ba560c0cb8327f3f24166c0fd193a
            client-secret: otVtgopxwCYYasyPcPW7yFPRB8SeWs0z
            redirect-uri: http://localhost/login/oauth2/code/kakao
            authorization-grant-type: authorization_code
            client-authentication-method: POST
            client-name: Kakao
            scope:
              - profile_nickname
              - account_email
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 설정
    - ● application.yml 에 로그인 설정 관련 코드를 추가

```
provider:
  kakao:
    authorization-uri: https://kauth.kakao.com/oauth/authorize
    token-uri: https://kauth.kakao.com/oauth/token
    user-info-uri: https://kapi.kakao.com/v2/user/me
    user-name-attribute: id
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 설정
    - CustomSecurityConfig.java 파일에 설정 추가

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("------------------configure-----------------------------");
        http.formLogin().loginPage("/member/login");
        http.oauth2Login().loginPage("/member/login");
        http.csrf().disable();

        http.rememberMe()
                .key("12345678")
                .tokenRepository(persistentTokenRepository())
                .userDetailsService(userDetailsService)
                .tokenValiditySeconds(60*60*24*30);

        http.exceptionHandling().accessDeniedHandler(accessDeniedHandler());

        return http.build();
}
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 설정
    - login.html 파일에 카카오 로그인 링크 추가

```
<div>
    <a href="/oauth2/authorization/kakao">KAKAO</a>
</div>
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 설정
    - ● 소셜 로그인 연동 후 이메일 확인 – security.CustomOAuth2UserService

```
@Log4j2
@Service
@RequiredArgsConstructor
public class CustomOAuth2UserService extends DefaultOAuth2UserService {
    private String getKakaoEmail(Map<String, Object> paramMap){
        log.info("KAKAO----------------------------------------");
        Object value = paramMap.get("kakao_account");
        log.info(value);
        LinkedHashMap accountMap = (LinkedHashMap) value;
        String email = (String)accountMap.get("email");
        log.info("email..." + email);
        return email;
    }
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 설정
    - ● 소셜 로그인 연동 후 이메일 확인 – security.CustomOAuth2UserService

```
@Override
public OAuth2User loadUser(OAuth2UserRequest userRequest) throws
 OAuth2AuthenticationException {

    log.info("userRequest....");
    log.info(userRequest);

    log.info("oauth2 user.....................................");

    ClientRegistration clientRegistration = userRequest.getClientRegistration();
    String clientName = clientRegistration.getClientName();

    log.info("NAME: "+clientName);

    OAuth2User oAuth2User = super.loadUser(userRequest);
    Map<String, Object> paramMap = oAuth2User.getAttributes();
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 로그인 설정
    - ● 소셜 로그인 연동 후 이메일 확인 – security.CustomOAuth2UserService

```
        String email = null;
        log.warn(clientName);
        switch (clientName.toLowerCase()){
            case "kakao":
                email = getKakaoEmail(paramMap);
                break;
        }

        log.info(email);
        log.info("================");

        return oAuth2User;
    }
}
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 소셜 로그인 후 처리
    - ● 회원 정보를 저장하기 위한 ClubMemberSecurityDTO 클래스 수정

```
@Getter
@Setter
@ToString
public class ClubMemberSecurityDTO extends User implements OAuth2User{
    private String mid;
    private String mpw;
    private String email;
    private String name;
    private boolean del;
    private boolean social;
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 소셜 로그인 후 처리
    - ● 회원 정보를 저장하기 위한 ClubMemberSecurityDTO 클래스 수정

```
public ClubMemberSecurityDTO(String username, String password, String
 email, String name, boolean del, boolean social, Collection<? extends
 GrantedAuthority> authorities) {
    super(username, password, authorities);
    this.mid = username;
    this.mpw = password;
    this.email = email;
    this.name = name;
    this.del = del;
    this.social = social;
}
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 소셜 로그인 후 처리
    - ● 회원 정보를 저장하기 위한 ClubMemberSecurityDTO 클래스 수정

```
    private Map<String, Object> props; //소셜 로그인 정보

    @Override
    public Map<String, Object> getAttributes(){
        return this.getProps();
    }

    @Override
    public String getName(){
        return this.mid;
    }
}
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 소셜 로그인 후 처리
    - ● 소셜 로그인을 수행한 이메일을 가진 사용자를 찾아보고 없는 경우에는 자동으로 회원 가입을 하고 ClubMemberSecurityDTO를 반환하도록 CustomOAuth2UserService 클래스 수정

```
@Log4j2
@Service
@RequiredArgsConstructor
public class CustomOAuth2UserService extends DefaultOAuth2UserService {
    private final ClubMemberRepository memberRepository;
    private final PasswordEncoder passwordEncoder;

    private String getKakaoEmail(Map<String, Object> paramMap){
        log.info("KAKAO----------------------------------------");
        Object value = paramMap.get("kakao_account");
        log.info(value);
        LinkedHashMap accountMap = (LinkedHashMap) value;
        String email = (String)accountMap.get("email");
        log.info("email..." + email);
        return email;
    }
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 소셜 로그인 후 처리
    - ● 소셜 로그인을 수행한 이메일을 가진 사용자를 찾아보고 없는 경우에는 자동으로 회원 가입을 하고 ClubMemberSecurityDTO를 반환하도록 CustomOAuth2UserService 클래스 수정

```
private ClubMemberSecurityDTO generateDTO(String email, Map<String,
 Object> params){

    Optional<ClubMember> result = memberRepository.findByEmail(email);

    //데이터베이스에 해당 이메일을 사용자가 없다면
    if(result.isEmpty()){
        //회원 추가 -- mid는 이메일 주소/ 패스워드는 1111
        ClubMember member = ClubMember.builder()
                .mid(email)
                .mpw(passwordEncoder.encode("1111"))
                .email(email)
                .social(true)
                .build();
        member.addRole(ClubMemberRole.USER);
        memberRepository.save(member);
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 소셜 로그인 후 처리
    - ● 소셜 로그인을 수행한 이메일을 가진 사용자를 찾아보고 없는 경우에는 자동으로 회원 가입을 하고 ClubMemberSecurity를 반환하도록 CustomOAuth2UserService 수정

```
        //MemberSecurityDTO 구성 및 반환
        ClubMemberSecurityDTO memberSecurityDTO =
                new ClubMemberSecurityDTO(email, "1111", email, null, false, true,
Arrays.asList(new SimpleGrantedAuthority("ROLE_USER")));
        memberSecurityDTO.setProps(params);

        return memberSecurityDTO;
    }
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 소셜 로그인 후 처리
    - ● 소셜 로그인을 수행한 이메일을 가진 사용자를 찾아보고 없는 경우에는 자동으로 회원 가입을 하고 ClubMemberSecurity를 반환하도록 CustomOAuth2UserService 수정

```
else {
        ClubMember member = result.get();
        ClubMemberSecurityDTO memberSecurityDTO =
                new ClubMemberSecurityDTO(
                        member.getMid(),
                        member.getMpw(),
                        member.getEmail(),
                        member.getName(),
                        member.isDel(),
                        member.isSocial(),
                        member.getRoleSet()
                                .stream().map(memberRole -> new
SimpleGrantedAuthority("ROLE_"+memberRole.name()))
                                .collect(Collectors.toList())
                );

        return memberSecurityDTO;
    }
}
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 소셜 로그인 후 처리
    - ● 소셜 로그인을 수행한 이메일을 가진 사용자를 찾아보고 없는 경우에는 자동으로 회원 가입을 하고 ClubMemberSecurity를 반환하도록 CustomOAuth2UserService 수정

```
@Override
public OAuth2User loadUser(OAuth2UserRequest userRequest) throws
 OAuth2AuthenticationException {

    log.info("userRequest....");
    log.info(userRequest);

    log.info("oauth2 user.....................................");

    ClientRegistration clientRegistration = userRequest.getClientRegistration();
    String clientName = clientRegistration.getClientName();

    log.info("NAME: "+clientName);

    OAuth2User oAuth2User = super.loadUser(userRequest);
    Map<String, Object> paramMap = oAuth2User.getAttributes();
```

Spring Boot Security

- ❖ Kakao Login
  - ✓ 소셜 로그인 후 처리
    - ● 소셜 로그인을 수행한 이메일을 가진 사용자를 찾아보고 없는 경우에는 자동으로 회원 가입을 하고 ClubMemberSecurity를 반환하도록 CustomOAuth2UserService 수정

```
        String email = null;
        log.warn(clientName);
        switch (clientName.toLowerCase()){
            case "kakao":
                email = getKakaoEmail(paramMap);
                break;
        }

        log.info(email);
        log.info("================");

        return generateDTO(email, paramMap);
    }
}
```

Spring Boot Security

- 로그인 후 처리
  - ✓ 스프링 Security는 로그인 성공 과 실패를 커스터마이징할 수 있도록 AuthenticationSuccessHandler 와 AuthenticationFailureHandler 인터페이스를 제공
  - ✓ 소셜로 로그인 한 경우 회원 가입을 다시 수행하도록 하거나 비밀번호 변경 등의 작업을 수행하도록 할 수 있음

Spring Boot Security

- 로그인 후 처리
  - ✓ 소셜 로그인 후 처리를 위한 Handler 클래스를 생성 – security
    CustomSocialLoginSuccessHandler

```
@Log4j2
@RequiredArgsConstructor
public class CustomSocialLoginSuccessHandler implements
    AuthenticationSuccessHandler {

    private final PasswordEncoder passwordEncoder;
```

Spring Boot Security

- 로그인 후 처리
  - ✓ 소셜 로그인 후 처리를 위한 Handler 클래스를 생성 – security.CustomSocialLoginSuccessHandler

```
@Override
public void onAuthenticationSuccess(HttpServletRequest request,
 HttpServletResponse response, Authentication authentication) throws IOException {

    log.info("------------------------------------------------------------");
    log.info("CustomLoginSuccessHandler onAuthenticationSuccess .....");
    log.info(authentication.getPrincipal());

    ClubMemberSecurityDTO memberSecurityDTO = (ClubMemberSecurityDTO)
 authentication.getPrincipal();

    String encodedPw = memberSecurityDTO.getMpw();
```

Spring Boot Security

- 로그인 후 처리
  - ✓ 소셜 로그인 후 처리를 위한 Handler 클래스를 생성 – security.CustomSocialLoginSuccessHandler

```
        //소셜로그인이고 회원의 패스워드가 1111이라면
        if (memberSecurityDTO.isSocial()
                && (memberSecurityDTO.getMpw().equals("1111")
                ||  passwordEncoder.matches("1111", memberSecurityDTO.getMpw())
        )) {
            log.info("Should Change Password");

            log.info("Redirect to Member Modify ");
            response.sendRedirect("/member/modify");

            return;
        } else {
            response.sendRedirect("/");
        }
    }
}
```

Spring Boot Security

- 로그인 후 처리
  - ✓ 소셜 로그인 후 처리를 위한 Handler 클래스를 적용 – security. CustomSecurityConfig

```
@Bean
public AuthenticationSuccessHandler authenticationSuccessHandler(){
        return new CustomSocialLoginSuccessHandler(passwordEncoder());
}

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("------------------configure-----------------------------");
        http.formLogin().loginPage("/member/login");
    http.oauth2Login().loginPage("/member/login").successHandler(authenticationSucces
    sHandler());
        http.csrf().disable();

        http.rememberMe()
                .key("12345678")
                .tokenRepository(persistentTokenRepository())
                .userDetailsService(userDetailsService)
                .tokenValiditySeconds(60*60*24*30);

        http.exceptionHandling().accessDeniedHandler(accessDeniedHandler());
        return http.build();
}
```

Spring Boot Security

- 로그인 후 처리
  - ✓ ClubMemberRepository 인터페이스에 비밀번호 변경 메서드 선언

```
@Modifying
@Transactional
@Query("update ClubMember m set m.mpw =:mpw where m.mid = :mid ")
void updatePassword(@Param("mpw") String password, @Param("mid") String mid);
```

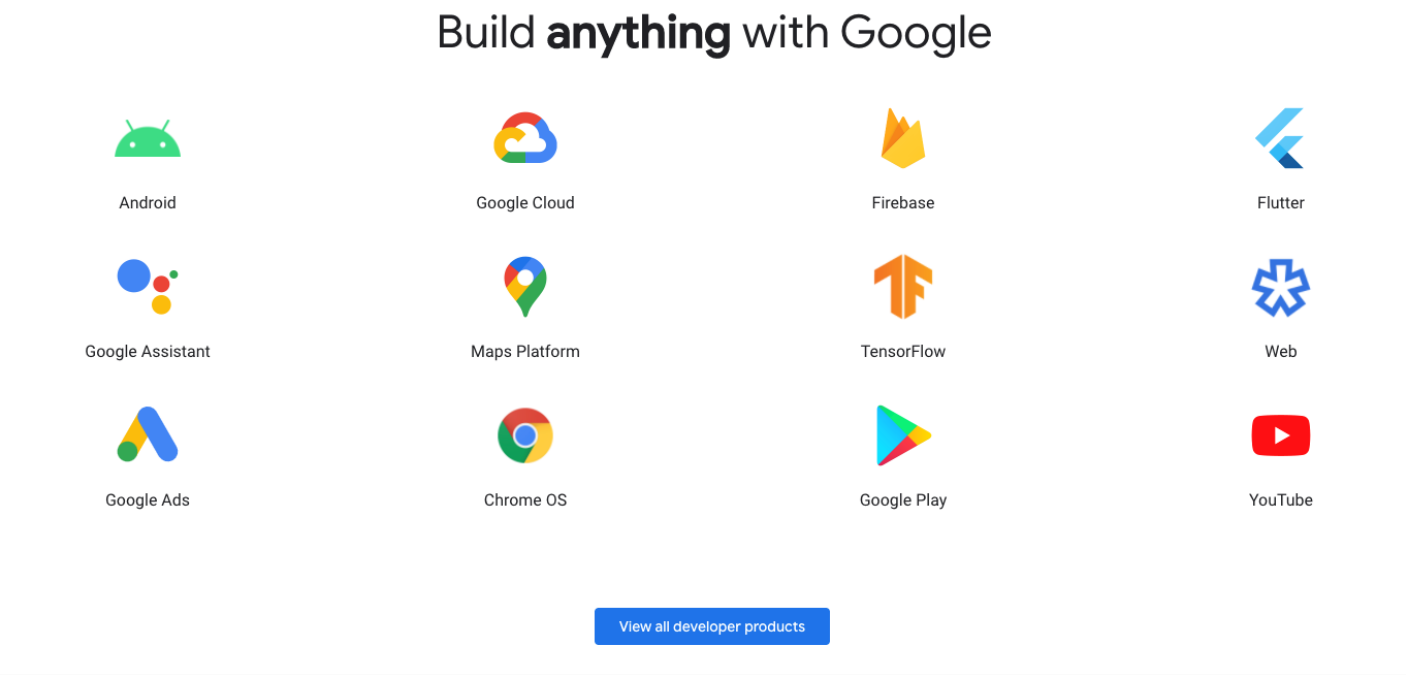
Spring Boot Security

- 로그인 후 처리
  - ✓ Test 클래스에서 변경

```
@Commit
@Test
public void testUpdate(){
        String mid = "itstudy@kakao.com";
        String mpw = passwordEncoder.encode("1234");
        clubMemberRepository.updatePassword(mpw, mid);
}
```

Spring Boot Security

- ❖ 구글 로그인
  - ✓ 구글 프로젝트에 현재 프로젝트 등록
    - developers.google.com 에 접속해서 Cloud 에 접속해서 Console 을 선택



Spring Boot Security

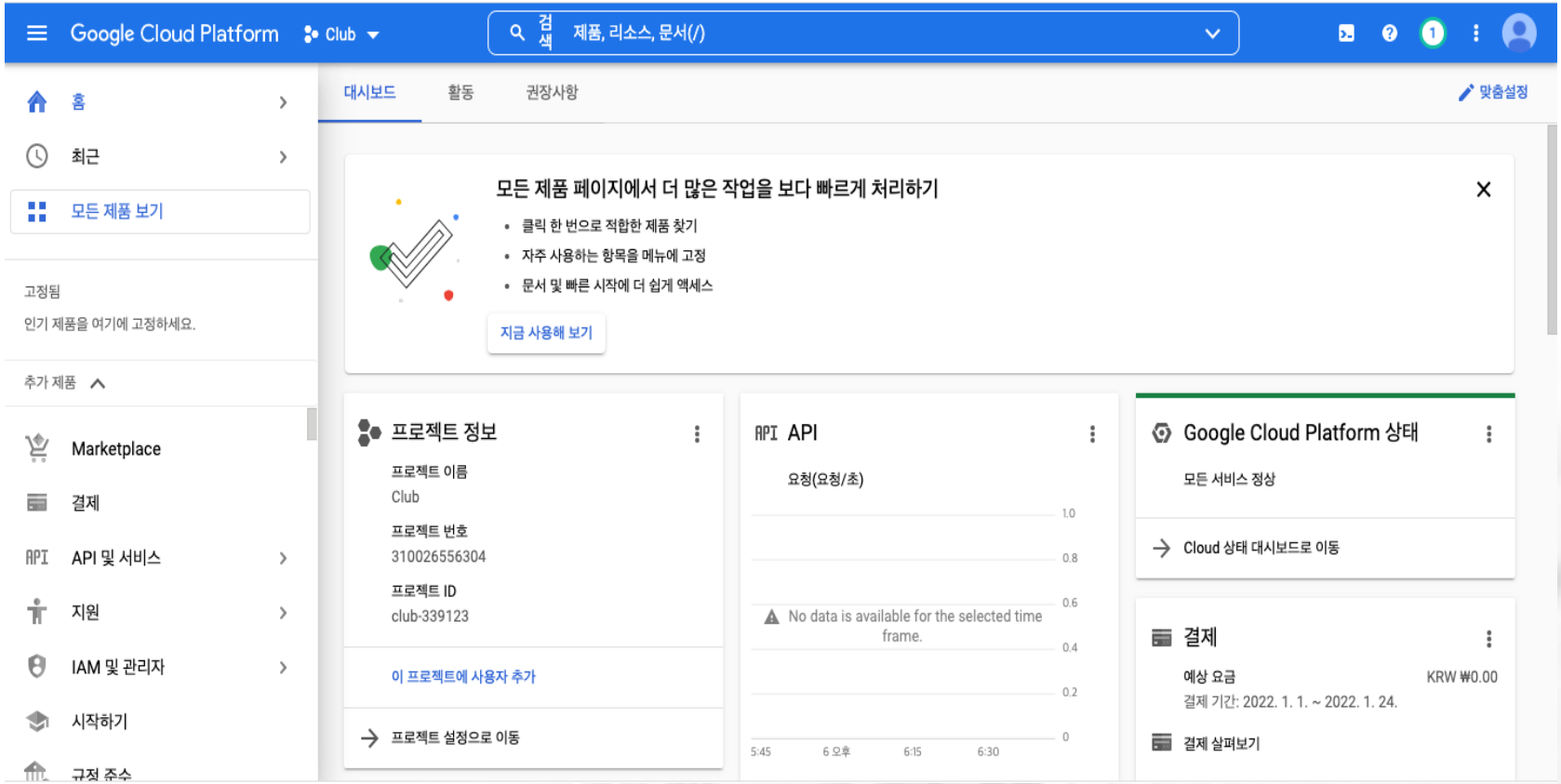
- ❖ 구글 로그인
  - ✓ 구글 프로젝트에 현재 프로젝트 등록
    - developers.google.com 에 접속해서 Cloud 에 접속해서 Console 을 선택

Spring Boot Security

❖ 구글 로그인

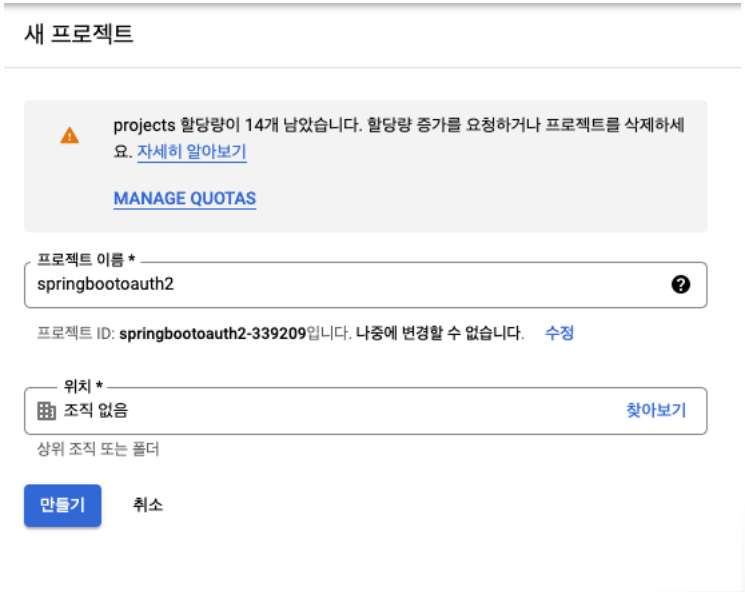
✓ 구글 프로젝트에 현재 프로젝트 등록

- developers.google.com 에 접속해서 Cloud 에 접속해서 Console 을 선택



Spring Boot Security

- ❖ 구글 로그인
  - ✓ 구글 프로젝트에 현재 프로젝트 등록
    - ● 새 프로젝트 선택

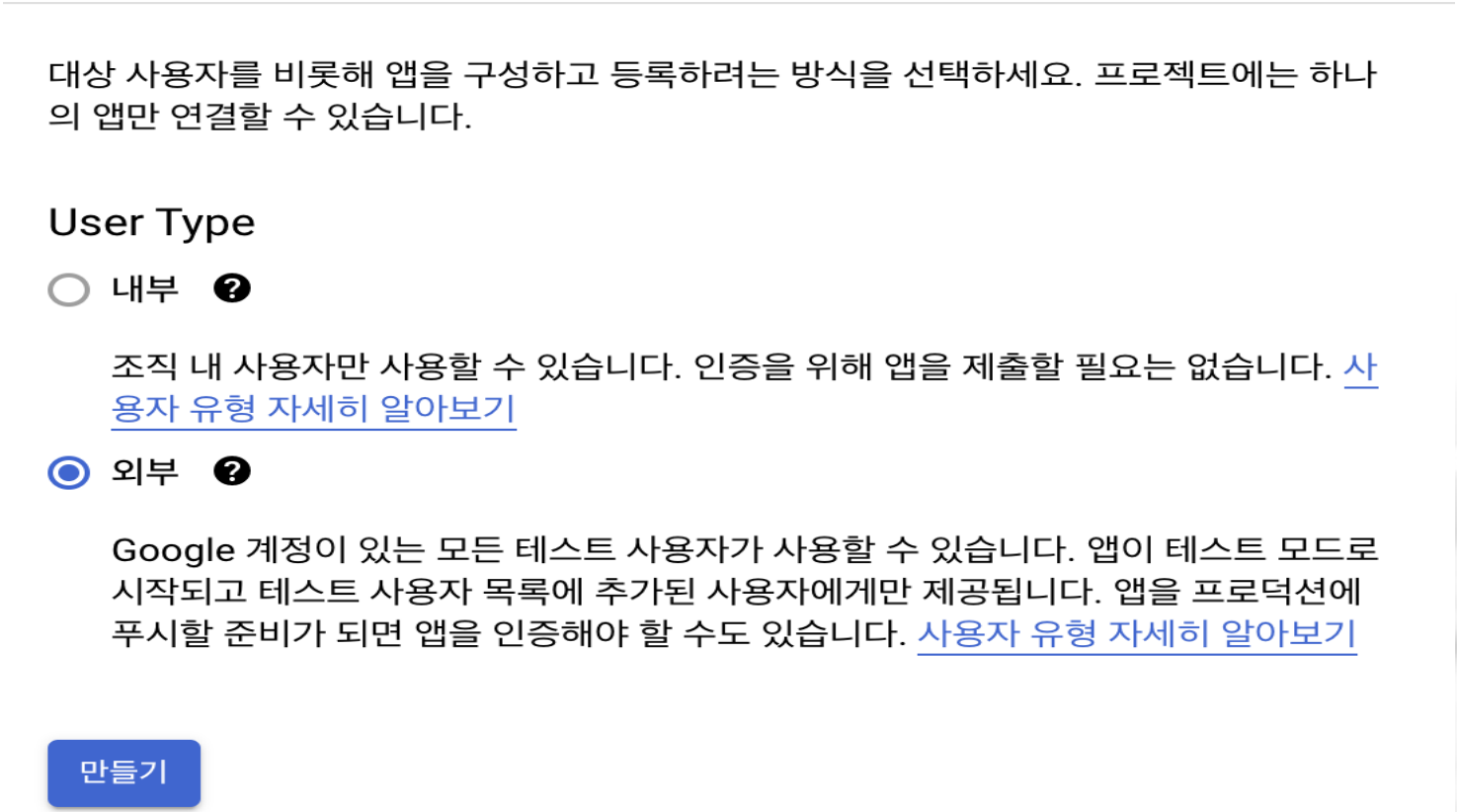


Spring Boot Security

❖ 구글 로그인

✓ 구글 프로젝트에 현재 프로젝트 등록

- API 및 서비스에서 OAuth 동의 화면을 선택하고 User Type을 선택

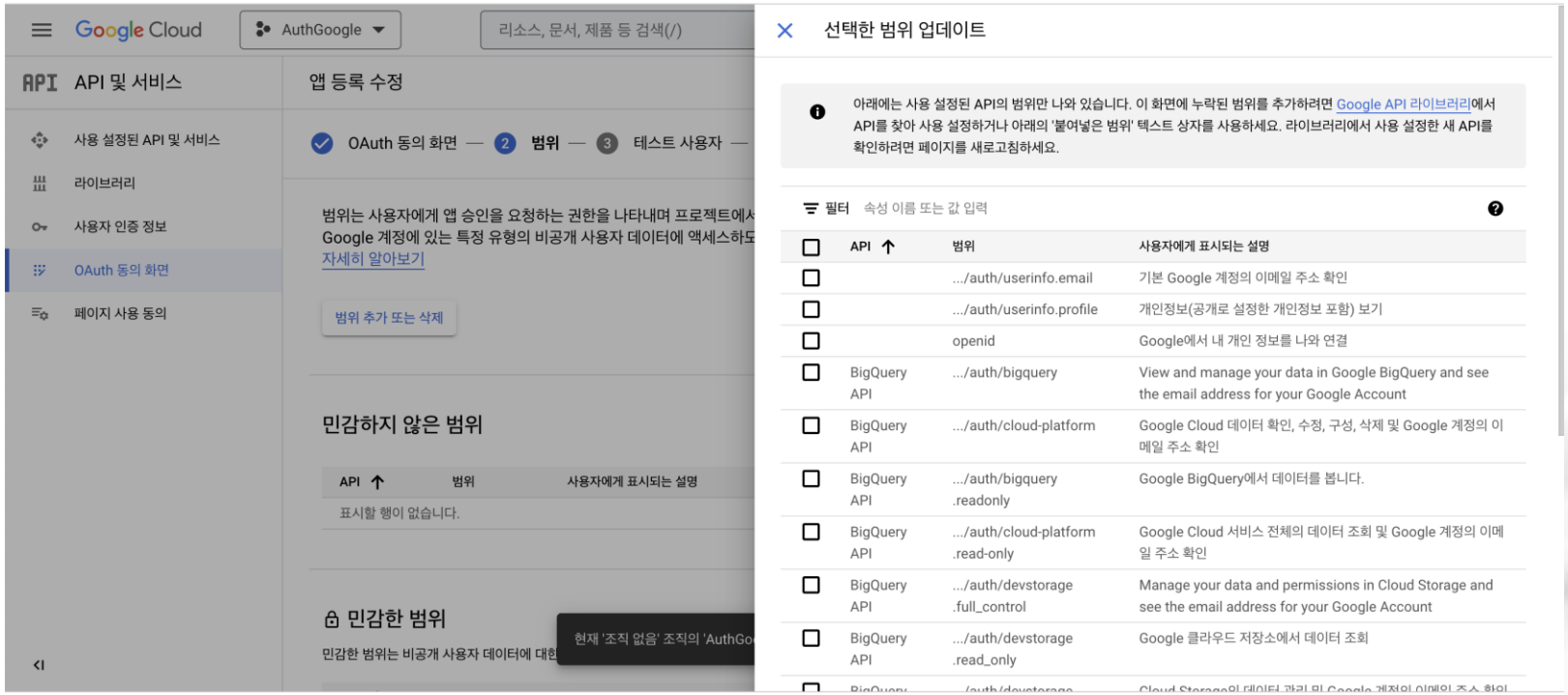


Spring Boot Security

- ❖ 구글 로그인
  - ✓ 구글 프로젝트에 현재 프로젝트 등록
    - ● 앱 이름 과 이메일 설정

Spring Boot Security

- 구글 로그인
  - ✓ 구글 프로젝트에 현재 프로젝트 등록
    - 범위 선택



Spring Boot Security

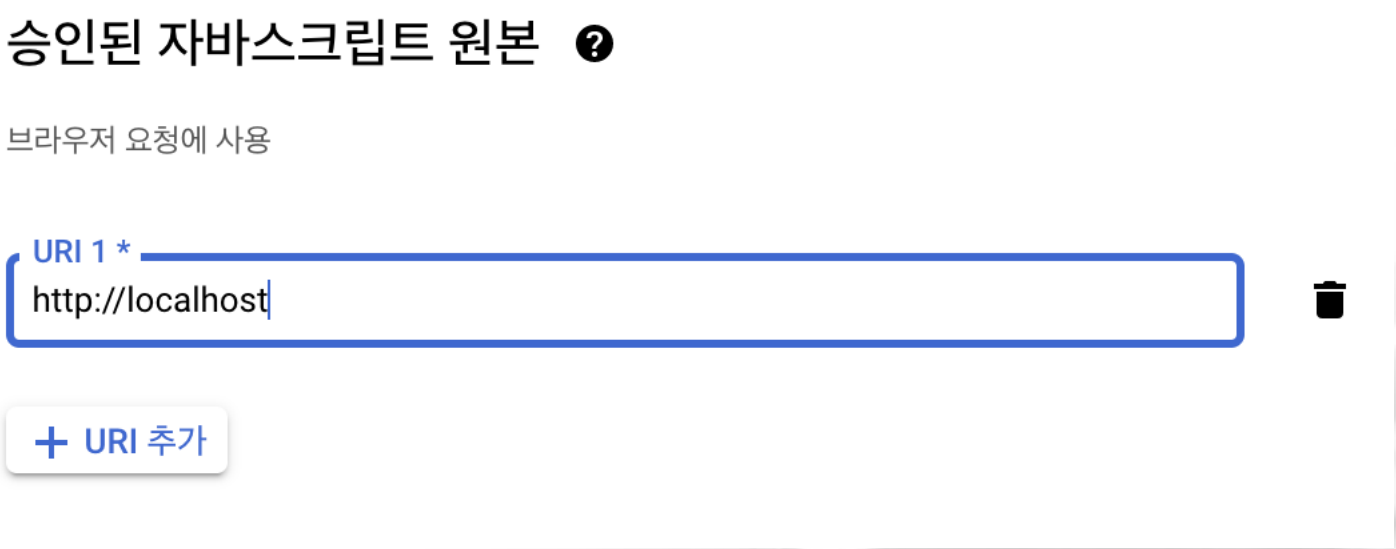
❖ 구글 로그인

✓ 구글 프로젝트에 현재 프로젝트 등록

- API 및 서비스에서 사용자 인증 정보 만들기를 선택하고 Oauth 2.0 클라이언트 ID 만들기 선택

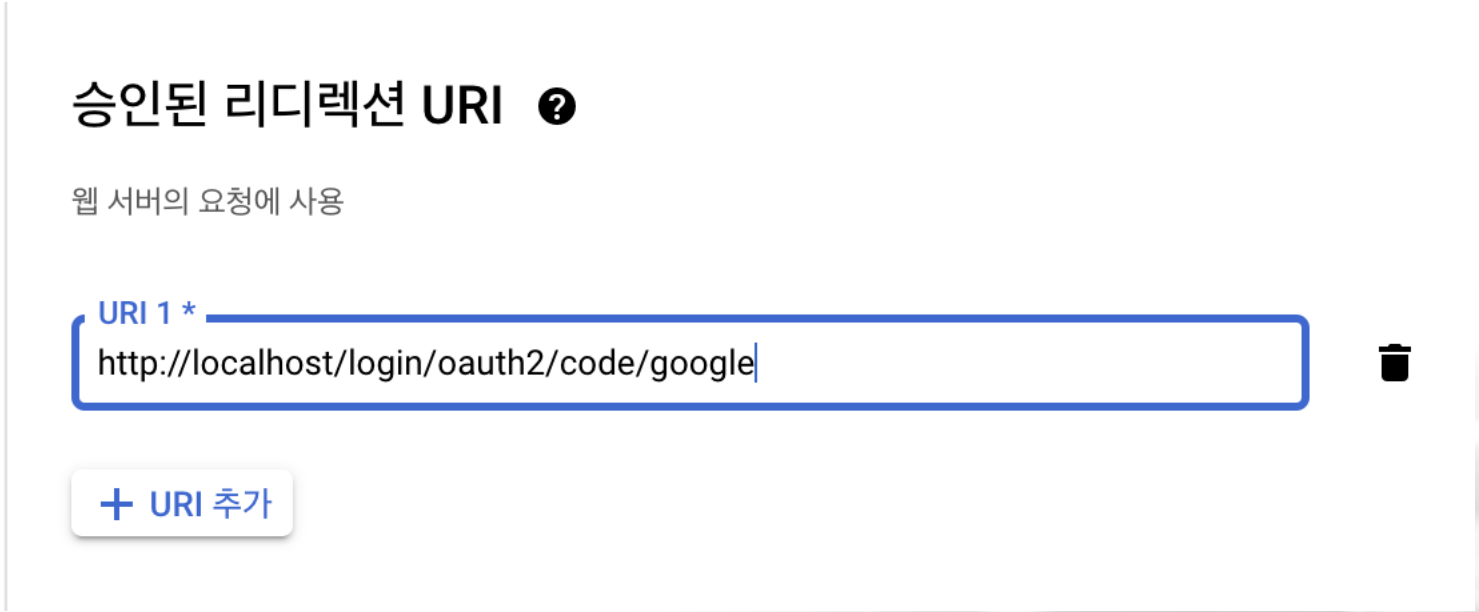
Spring Boot Security

- ❖ 구글 로그인
  - ✓ 구글 프로젝트에 현재 프로젝트 등록
    - API 및 서비스에서 사용자 인증 정보 만들기를 선택하고 Oauth 2.0 클라이언트 ID 만들기 선택



Spring Boot Security

- ❖ 구글 로그인
  - ✓ 구글 프로젝트에 현재 프로젝트 등록
    - API 및 서비스에서 사용자 인증 정보 만들기를 선택하고 Oauth 2.0 클라이언트 ID 만들기 선택



Spring Boot Security

- ❖ 구글 로그인
  - ✓ 구글 프로젝트에 현재 프로젝트 등록
    - API 및 서비스에서 사용자 인증 정보 만들기를 선택하고 Oauth 2.0 클라이언트 ID 만들기 선택

Spring Boot Security

- 구글 로그인
  - ✓ 프로젝트에 적용
    - application.yml 파일에 추가

```
spring.security.oauth2.client.registration.google.client-id=310026556304-
    kma77c8upeuc2tbachi6vuebf9nhfrfb.apps.googleusercontent.com
spring.security.oauth2.client.registration.google.client-secret=GOCSPX-Zg-
    npsaQbHXOGGlWdHkNMeQrbNxB
spring.security.oauth2.client.registration.google.scope=email
spring.profiles.include=oauth
```

Spring Boot Security

- ❖ 구글 로그인
  - ✓ 프로젝트에 적용
    - CustomOAuth2UserService 클래스의 loadUser 메서드 수정

```
@Override
public OAuth2User loadUser(OAuth2UserRequest userRequest) throws
    OAuth2AuthenticationException {

        log.info("userRequest....");
        log.info(userRequest);

        log.info("oauth2 user.....................................");

        ClientRegistration clientRegistration = userRequest.getClientRegistration();
        String clientName = clientRegistration.getClientName();

        log.info("NAME: "+clientName);
```

Spring Boot Security

- ❖ 구글 로그인
  - ✓ 프로젝트에 적용
    - CustomOAuth2UserService 클래스의 loadUser 메서드 수정

```
OAuth2User oAuth2User = super.loadUser(userRequest);
Map<String, Object> paramMap = oAuth2User.getAttributes();
System.out.println("paramMap:" + paramMap);
String email = null;
log.warn(clientName);
switch (clientName.toLowerCase()){
    case "kakao":
        email = getKakaoEmail(paramMap);
        break;
    case "google":
        email = (String)paramMap.get("email");
        break;
}

log.info(email);
log.info("================");

return generateDTO(email, paramMap);
}
```

Spring Boot Security

- 세션 저장소로 데이터베이스 사용
  - ✓ 세션을 WAS의 메모리에 저장했을 때 문제점
    - 애플리케이션을 재실행하면 세션의 내용이 모두 사라지는 문제
    - 2대 이상의 서버에서 서비스하고 있다면 WAS마다 세션 동기화 설정을 해야만 하는 문제
  - ✓ 세션 저장소를 사용하는 방법
    - WAS의 세션 사용
    - 데이터베이스 사용
    - Redis, Memcached 와 같은 Memory DB 사용 – 실제로는 Embedded Redis 와 같은 방식이 아닌 외부 메모리 서버가 필요

Spring Boot Security

- 세션 저장소로 데이터베이스 사용
  - ✓ 외부 데이터베이스 설정
    - build.gradle 의 dependencies에 추가

      implementation 'org.springframework.session:spring-session-jdbc'
    - application.yml에 추가

```
spring:
  session:
    timeout: 600
    store-type: jdbc
    jdbc:
      initialize-schema: always
      table-name: SPRING_SESSION
```

Spring Boot Security

- 세션 저장소로 데이터베이스 사용
  - ✓ 외부 데이터베이스 설정

```
SELECT * FROM SPRING_SESSION;
```

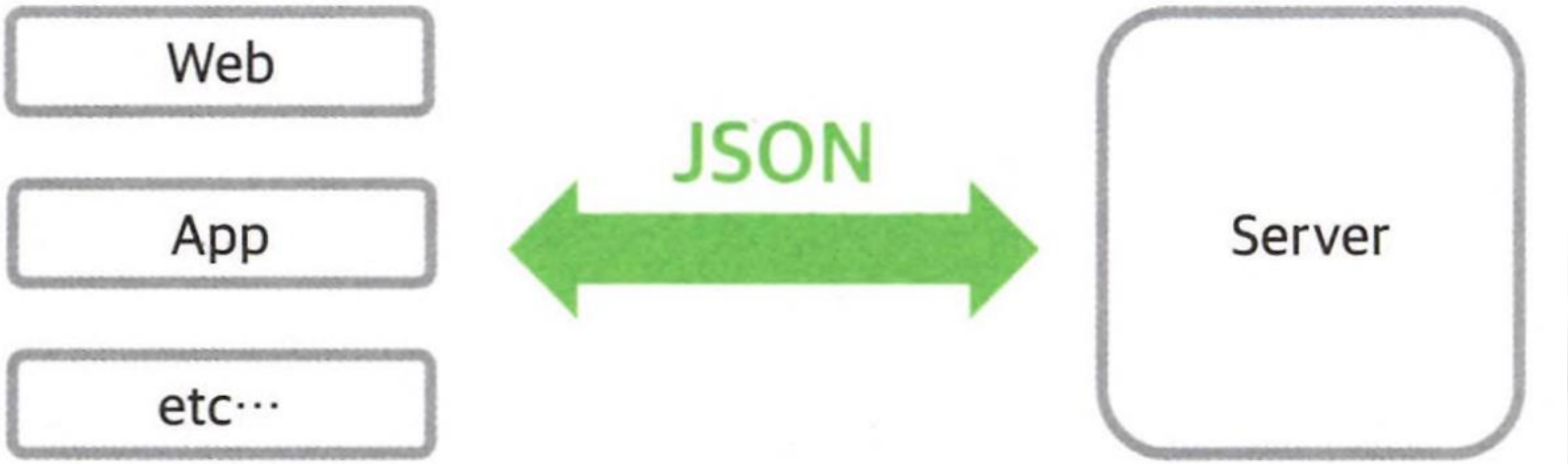
	PRIMARY_ID	SESSION_ID	CREATION_TIME	LAST_ACCESS_TIME
1	5433b10e-b8e4-4486-b275-baafcbf39a	a507f6d4-7911-4f5a-a334-2b5879a75a3	1,678,055,201,336	1,678,055,220,361

JWT 인증

- SSR
  - ✓ 브라우저에 필요한 화면의 모든 코드(HTML)를 서버에서 만들어 전송하는 방식을 서버 사이드 렌더링(Server Side Rendering - SSR) 이라고 하는데 JSP 와 Thymeleaf를 이용해서 출력하는 형태

JWT 인증

- CSR
  - ✓ API 서버는 필요한 데이터만 제공하는 서버를 의미



JWT 인증

- CSR
  - ✓ API 서버는 화면 구성은 별도의 클라이언트 프로그램에서 처리하고 서버에서는 순수한 데이터만을 전송하는데 이러한 구성이 클라이언트 사이드 렌더링(Client Side Rendering - CSR)
  - ✓ CSR 방식은 클라이언트에서 데이터를 가공해서 화면에 보여주기 때문에 데이터를 어떻게 구성해서 주고받을 것인지가 중요한데 주로 JSON/XML 포맷으로 데이터를 구성하고 호출 방식은 REST 방식을 이용하는 경우가 많음
  - ✓ API 서버는 화면을 구성하지 않는다는 특징 외에도 무상태(stateless) 라는 특징도 있는데 REST나 HTTP의 특징이지만 개발에 약간의 차이가 있기 때문에 주의가 필요
  - ✓ 전통적인 SSR 방식의 서비스는 쿠키와 세션을 이용해서 서버에서 사용자 정보를 추적할 수 있었는데 쿠키의 경우 쿠키를 발행한 서버를 호출할 때에만 전달되는 방식이고 세션의 경우 서버 내부에서 JSESSIONID와 같은 이름의 쿠키를 통해서 사용자 정보를 보관하고 처리할 수가 있는데 반면 API 서버는 쿠키를 이용해서 데이터를 교환하는 방식이 아니고 API 서버는 단순히 요청(request) 과 응답(response)에서 발생한 부수적인 결과를 유지하지 않는 방식으로 API 서버는 JSESSIONID 이름의 쿠키를 발행하거나 개발자가 직접 쿠키를 생성하지는 않는 형태로 동작
  - ✓ API 서버는 데이터를 요청하고 응답받는 방식으로 구성

JWT 인증

- 토큰 기반의 인증
  - ✓ API 서버가 단순히 데이터만을 주고받을 때 외부에서 누구나 호출하는 URI를 알게 되면 문제가 발생하기 때문에 당연히 API 서버에는 다양한 방법으로 특정한 사용자나 프로그램에서만 API 서버를 호출할 수 있도록 제한하는 방법이 필요
  - ✓ 초창기 API 서버는 주로 API를 호출하는 프로그램의 IP 주소를 이용했는데 API 서버를 호출하는 쪽의 IP와 API 서버 내에 보관된 IP를 비교해서 허용된 IP에서만 API 서버에 서 결과를 만들어주는 방식인데 IP와 더불어서 정해진 키(key) 값을 같이 사용하는 것이 일반적인 형태
  - ✓ 다른 방법은 토큰을 이용하는 방식인데 토큰은 일종의 표식과 같은 역할을 하는 데이터로 토큰은 서버와 클라이언트가 주고받는 문자열에 불과한데 API 서버를 이용하고자 하는 사람들은 API 서버에서 토큰을 받아 보관하고 호출할 때 자신이 가지고 있는 토큰을 같이 전달해서 API 서버에서 이를 확인하는 방식인데 API 서버에서는 입장권을 발행해 주고 클라이언트는 입장권을 가지고 방문하게 하는 원리

JWT 인증

- ❖ 토큰 기반의 인증
  - ✓ Access Token / Refresh Token
    - 권한을 체크하기 위한 토큰(token)을 API 서버에서는 Access Token 이라고 하는데 Access Token은 특정한 자원에 접근할 권한이 있는지를 검사하기 위한 용도로 외부에서 API 서버를 호출할 때 함께 전달하면 이를 이용해서 검증하고 그 결과에 따라서 요청을 처리
    - Access Token을 악의적인 사용자에게 탈취당한다면 문제가 발생하기 때문에 Access Tokens 이용할 때는 최대한 유효 시간을 짧게 지정하고 사용자에게는 Access Token을 새로 발급받을 수 있는 Refresh Token이라는 것을 같이 생성해 주어서 필요할 때 다시 Access Token을 발급 받을 수 있도록 구성
    - Refresh Token이 반드시 필수라고 할 수는 없는데 다시 Access Token을 발급 받기만 한다면 다른 방법도 고려할 수 있음

JWT 인증

- ❖ 토큰 기반의 인증
  - ✓ Access Token / Refresh Token
    - ● 토큰(token) 방식에서 걸림돌이 되는 것은 실제로는 단순한 문자열인 토큰의 보안 문제로 누군가 토큰을 탈취해서 서버에 요청하게 된다면 서버에서 정상적인 사용자로 간주하기 때문에 Access Token을 작성할 때는 기본적으로 암호화나 인코딩 처리가 필요하고 Access Token 내부에 유효 시간을 지정해서 짧은 시간 동안만 토큰을 사용할 수 있도록 처리해야 함

JWT 인증

❖ 토큰 기반의 인증

✓ Access Token / Refresh Token

● Access Token이 만료되는 상황

◆ 사용자가 Access Token을 전달하면 API 서버에서는 Access Token을 검증하는데 이 과정에서 만료된(expired) 토큰 임을 확인하고 사용자에게 만료된 토큰 임을 알려줌

◆ 사용자는 자신이 가지고 있는 Refresh Token을 전송해서 새로운 Access Token을 요구하고 API 서버에서는 Refresh Token에 문제가 없다면 새로운 Access Token을 생성해서 전달하고 이 과정에서 Refresh Token이 거의 만료된 상황이라면 새로운 Refresh Token을 같이 전송할 수 도 있음

JWT 인증

- ❖ 토큰 기반의 인증
  - ✓ Access Token / Refresh Token
    - 토큰을 이용하는 방식은 네트워크로 데이터를 주고받기 때문에 항상 보안 문제가 있는데 Access Token과 마찬가지로 Refresh Token 역시 탈취될 때는 문제가 될 수 있음
    - Access Token과 Refresh Token 모두를 탈취당하면 API 서버에서는 사용자를 구분할 수 없는 문제가 발생하는데 이를 해결하기 위해 비밀키를 지정할 수도 있겠지만 비밀키가 탈취당하면 같은 문제가 발생하므로 근본적인 해결책은 아닌데 그나마 현실적인 해결책은 Refresh Token으로 새로운 Access Token이 생성되는 것을 원래의 사용자가 알 수 있도록 하는 것 - 구글이나 네이버 로그인을 평소와 다른 장소에서 하면 본인 스스로 한 것이 맞는지 확인하는 메일이나 핸드폰 알림 등이 발송되는 형태를 사용하는데 Access Token 과 Refresh Token 역시 이처럼 원래의 사용자에게 활동 여부를 알려주거나 특정한 IP에서만 사용할 수 있도록 하는 방법 등을 사용해서 조금 더 안전하게 만들 수 있음

API Server

- 프로젝트 생성
  - ✓ 의존성
    - Spring Boot DevTools
    - Lombok
    - Spring Web
    - Spring Security
    - Spring Data JPA
    - MySQL

API Server

- 프로젝트 생성
  - ✓ application.properties 파일을 application.yml 파일로 수정하고 작성

```
server:
  port: 80

spring:
  profiles:
    include: oauth
```

API Server

- 프로젝트 생성
  - ✓ application.properties 파일을 application.yml 파일로 수정하고 작성

```
security:
  oauth2:
    client:
      registration:
        kakao:
          client-id: df1ba560c0cb8327f3f24166c0fd193a
          client-secret: otVtgopxwCYYasyPcPW7yFPRB8SeWs0z
          redirect-uri: http://localhost/login/oauth2/code/kakao
          authorization-grant-type: authorization_code
          client-authentication-method: POST
          client-name: Kakao
          scope:
            - profile_nickname
            - account_email
        google:
          client-id: 551789175632-
gbgr3rjci9o6oto3gndmjsrpo81m0mvs.apps.googleusercontent.com
          client-secret: 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
          scope: email
```

API Server

- 프로젝트 생성
  - ✓ application.properties 파일을 application.yml 파일로 수정하고 작성

```
provider:
  kakao:
    authorization-uri: https://kauth.kakao.com/oauth/authorize
    token-uri: https://kauth.kakao.com/oauth/token
    user-info-uri: https://kapi.kakao.com/v2/user/me
    user-name-attribute: id
```

API Server

- 프로젝트 생성
  - ✓ application.properties 파일을 application.yml 파일로 수정하고 작성

```
#Live Reload
devtools:
  livereload:
    enabled: true

datasource:
  url: jdbc:mysql://localhost:3306/adam
  driver-class-name: com.mysql.cj.jdbc.Driver
  username: root
  password: wnddkd

jpa:
  hibernate:
    ddl-auto: update
  properties:
    hibernate:
      format_sql: true
      show_sql: true
```

API Server

- 프로젝트 생성
  - ✓ application.properties 파일을 application.yml 파일로 수정하고 작성

```
servlet:
  multipart:
    enabled: true
  location: /Users/adam/Documents/data
  max-request-size: 30MB
  max-file-size: 10MB

com:
  adamsoft:
    upload:
      path: /Users/adam/Documents/data

logging:
  level:
    org:
      hibernate:
        type:
          descriptor:
            sql: trace
      springframework:
        security: trace
```

API Server

- 프로젝트 생성
  - ✓ Security 설정 클래스를 추가하고 작성 – config.CustomSecurityConfig

```
@Configuration
@Log4j2
@RequiredArgsConstructor
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class CustomSecurityConfig{

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

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("-------------------configure-------------------------------");
        http.csrf().disable();
        //세션을 사용하지 않음

     http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        return http.build();
    }
```

API Server

- 프로젝트 생성
  - ✓ Security 설정 클래스를 추가하고 작성 – config.CustomSecurityConfig

```
@Bean
public WebSecurityCustomizer webSecurityCustomizer() {
    log.info("------------web configure-------------------");
    return (web) ->
 web.ignoring().requestMatchers(PathRequest.toStaticResources().atCommonLocation
 s());
}
}
```

API Server

- ❖ 프로젝트 생성
  - ✓ ModelMapper 라이브러리
    - ● 객체들 사이의 매핑을 자동으로 수행해주는 라이브러리
    - ● ModelMapper.map(Object source, Class<D> destinationType) 메서드를 호출하면 source를 읽어서 동일한 이름의 속성을 찾아서 destinationType에 자동으로 매핑
    - ● build.gradle 파일에 의존성 추가
      - ◆ implementation 'org.modelmapper:modelmapper:3.1.0'

API Server

- 프로젝트 생성
  - ✓ ModelMapper를 사용하기 위한 설정 클래스 – config.RootConfig

```
@Configuration
public class RootConfig {

    @Bean
    public ModelMapper getMapper() {
        ModelMapper modelMapper = new ModelMapper();
        modelMapper.getConfiguration()
                .setFieldMatchingEnabled(true)
                .setFieldAccessLevel(org.modelmapper.config.Configuration.AccessLevel.PRI
     VATE)
                .setMatchingStrategy(MatchingStrategies.LOOSE);

        return modelMapper;
    }
}
```

API Server

- 프로젝트 생성
  - ✓ Swagger
    - 개발자가 REST 웹 서비스를 설계, 빌드, 문서화, 소비하는 일을 도와주는 대형 도구 생태계의 지원을 받는 오픈 소스 소프트웨어 프레임워크
    - build.gradle 파일에 의존성 추가
      - implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.0.2'

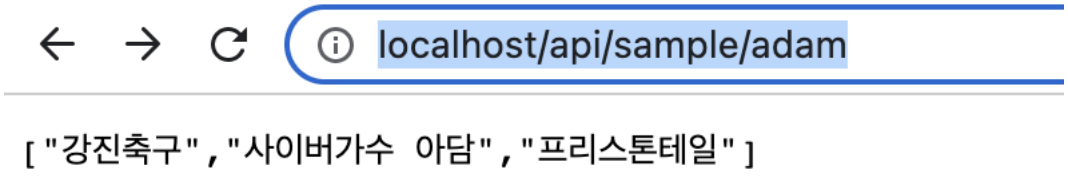
API Server

- 프로젝트 생성
  - ✓ Controller 역할을 수행할 클래스를 만들고 샘플 코드 작성 – controller.SampleController

```
@RestController
@RequestMapping("/api/sample")
public class SampleController {
    @GetMapping("/adam")
    public List<String> adam(){
        return Arrays.asList("강진축구", "사이버가수 아담", "프리스톤테일");
    }
}
```

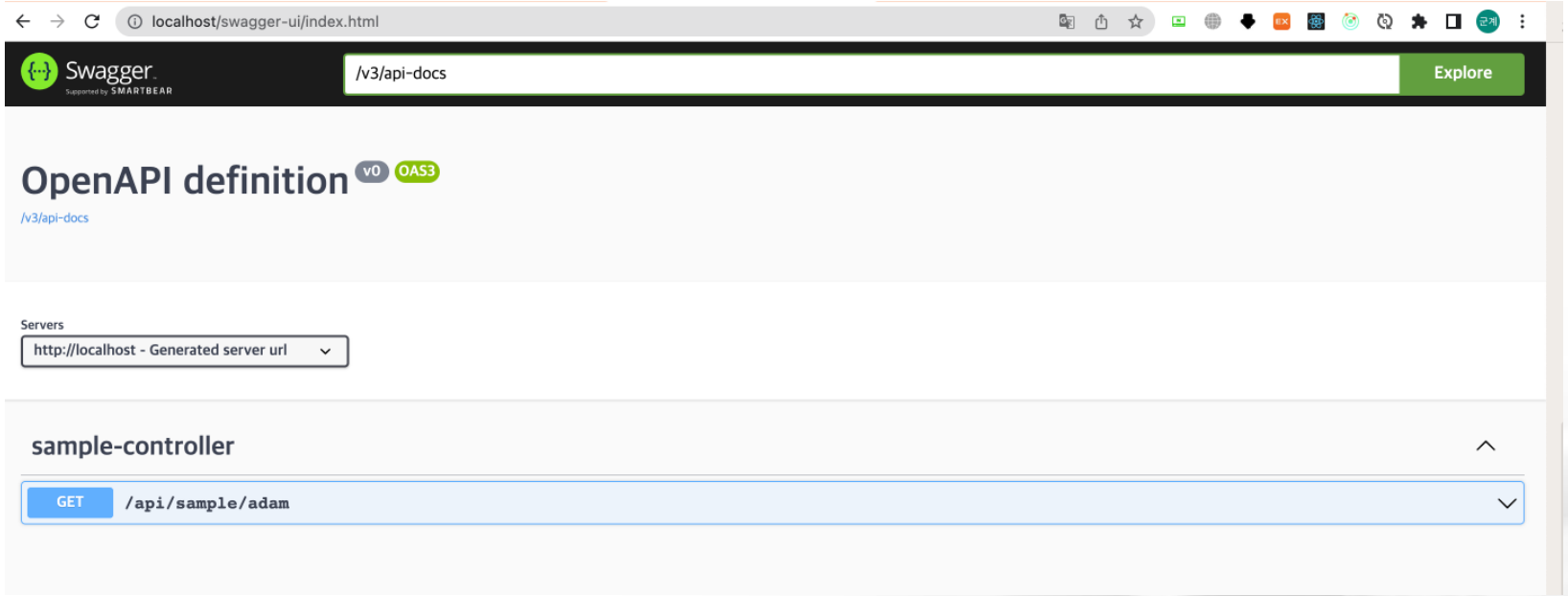
API Server

- 프로젝트 생성
  - ✓ 프로젝트를 실행하고 브라우저에서 접속: http://localhost/api/sample/adam



API Server

- 프로젝트 생성
  - ✓ 프로젝트를 실행하고 브라우저에서 접속: http://localhost/swagger-ui/index.html



API Server

- 프로젝트 생성
  - ✓ config 패키지에 웹 설정을 위한 클래스를 추가하고 설정 – CustomServletConfig

```
@Configuration
@EnableWebMvc
public class CustomServletConfig implements WebMvcConfigurer {
    @Override
    //클라이언트에서 자원을 요청할 때 처리할 메서드
    public void addResourceHandlers(ResourceHandlerRegistry registry){
        //files 로 요청하면 static 디렉토리에서 뷰를 찾아오는 설정
        registry.addResourceHandler("/files/**")
                .addResourceLocations("classpath:/static/");
    }
}
```

API Server

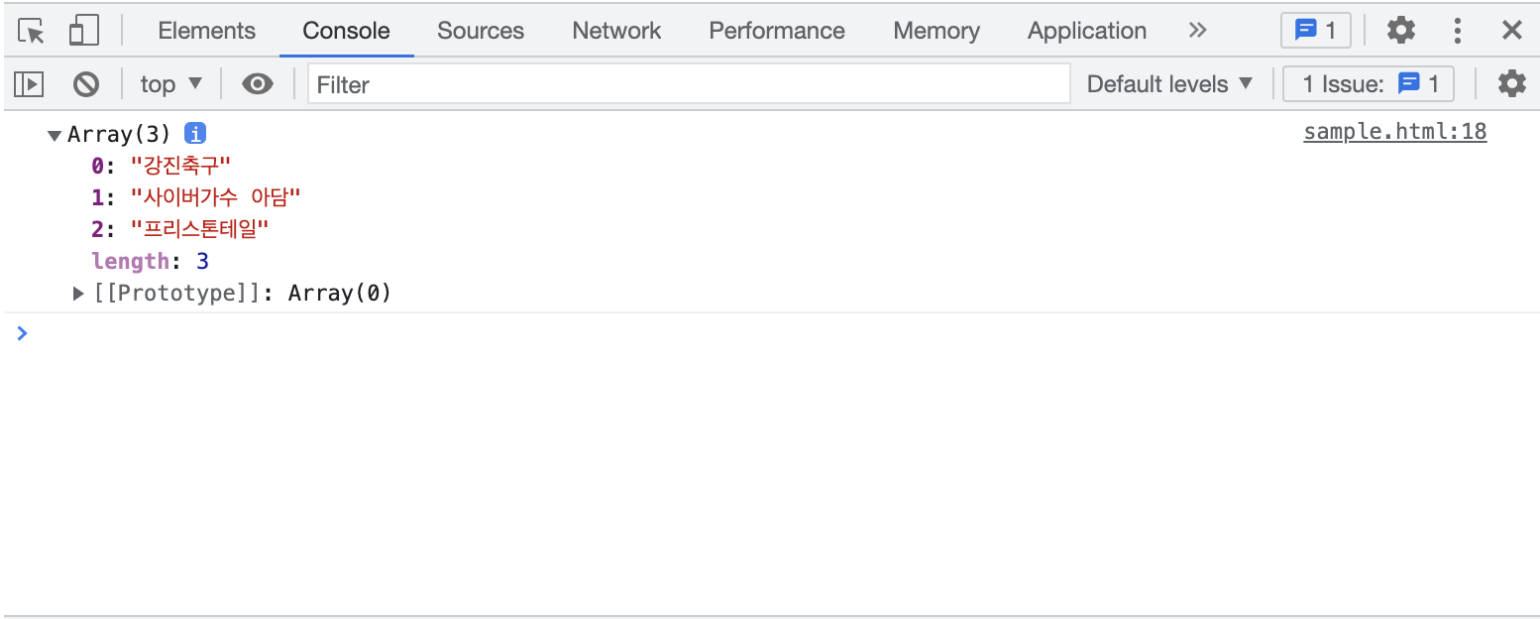
- 프로젝트 생성
  - ✓ ajax로 데이터 요청 - static 디렉토리에 sample.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>API Server</title>
</head>
<body>
<h1>Sample Client</h1>
</body>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  async function callTest(){
    const response = await axios.get("http://localhost/api/sample/adam")
    return response.data
  }

  callTest()
          .then(data => console.log(data))
          .catch(e => console.log(e));
</script>
</html>
```

API Server

❖ 프로젝트 생성

✓ 실행 후 호출: http://localhost/files/sample.html



API Server

- API 사용자 처리
  - ✓ 사용자 정보를 위한 테이블을 생성 – domain.APIUser

```
@Entity
@Getter
@Builder
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class APIUser {

    @Id
    private String mid;
    private String mpw;

    public void changePw(String mpw){
        this.mpw = mpw;
    }
}
```

API Server

- API 사용자 처리
  - ✓ 사용자 정보를 위한 Repository 인터페이스 생성 – persistence.APIUserRepository

```
public interface APIUserRepository extends JpaRepository<APIUser, String> {
}
```

API Server

- API 사용자 처리
  - ✓ 테스트 클래스를 이용해서 샘플 데이터 삽입 – APIUserRepositoryTests

```
@SpringBootTest
@Log4j2
public class APIUserRepositoryTests {
    @Autowired
    private PasswordEncoder passwordEncoder;

    @Autowired
    private APIUserRepository apiUserRepository;

    @Test
    public void testInserts(){
        IntStream.rangeClosed(1, 100).forEach(i -> {
            APIUser apiUser = APIUser.builder()
                    .mid("apiuser" + i)
                    .mpw(passwordEncoder.encode("1111"))
                    .build();
            apiUserRepository.save(apiUser);
        });
    }
}
```

API Server

- API 사용자 처리
  - ✓ 스프링 Security를 위한 DTO 클래스를 생성 – dto.APIUserDTO

```
@Getter
@Setter
@ToString
public class APIUserDTO extends User {

    private String mid;
    private String mpw;

    public APIUserDTO(String username, String password, Collection<GrantedAuthority>
     authorities) {
        super(username, password, authorities);
        this.mid = username;
        this.mpw = password;
    }
}
```

API Server

- API 사용자 처리
  - ✓ 스프링 Security를 위한 UserDetailsService 클래스를 생성 – security.APIUserDetailsService

```
@Service
@Log4j2
@RequiredArgsConstructor
public class APIUserDetailsService implements UserDetailsService {

    //주입
    private final APIUserRepository apiUserRepository;
```

API Server

- API 사용자 처리
  - ✓ 스프링 Security를 위한 UserDetailsService 클래스를 생성 – security.APIUserDetailsService

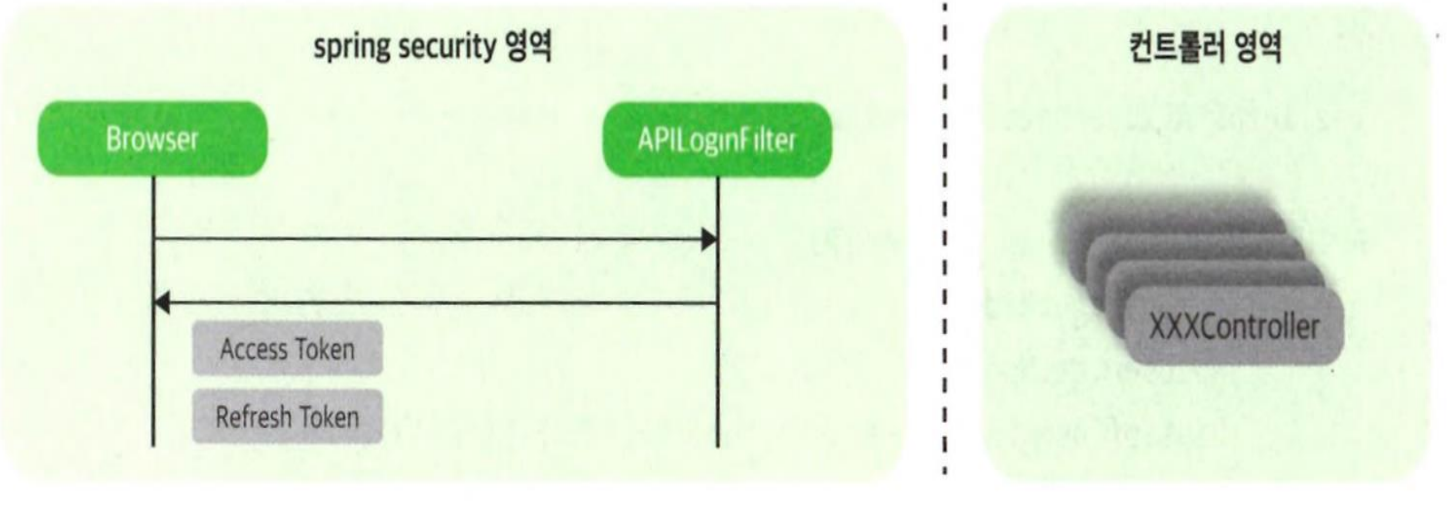
```
@Override
public UserDetails loadUserByUsername(String username) throws
 UsernameNotFoundException {

    Optional<APIUser> result = apiUserRepository.findById(username);
    APIUser apiUser = result.orElseThrow(() -> new
 UsernameNotFoundException("Cannot find mid"));

    log.info("APIUserDetailsService apiUser----------------------------------");

    APIUserDTO dto =  new APIUserDTO(
            apiUser.getMid(),
            apiUser.getMpw(),
            List.of(new SimpleGrantedAuthority("ROLE_USER")));

    log.info(dto);

    return dto;
  }
}
```

JWT 인증

- ❖ 토큰 인증을 위한 Security 필터
  - ✓ 필터의 기능
    - 사용자가 아이디(mid)와 패스워드(mpw)를 이용해서 Access Token과 Refresh Token을 얻으려는 단계 구현
    - 사용자가 Access Token을 이용해서 컨트롤러를 호출하고자 할 때 인증과 권한을 체크하는 기능 구현

JWT 인증

- ❖ 토큰 인증을 위한 Security 필터
  - ✓ 인증 과 JWT 발행 처리
    - 사용자의 아이디(mid)와 패스워드(mpw)를 이용해서 JWT 문자열을 발행하는 기능은 컨트롤러를 이용할 수도 있지만 스프링 Security의 AbstractAuthenticationProcessingFilter 클래스를 이용하면 좀 더 완전한 분리가 가능



JWT 인증

❖ 토큰 인증을 위한 Security 필터

✓ 필터 클래스 생성 – security.filter.APILoginFilter

```
@Log4j2
public class APILoginFilter extends AbstractAuthenticationProcessingFilter {

    public APILoginFilter(String defaultFilterProcessesUrl) {
        super(defaultFilterProcessesUrl);
    }

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request,
     HttpServletResponse response) throws AuthenticationException, IOException {

        log.info("APILoginFilter-----------------------------------");

        return null;
    }
}
```

JWT 인증

- ❖ 토큰 인증을 위한 Security 필터
  - ✓ AbstractAuthenticationProcessingFilter 설정
    - AbstractAuthenticationProcessingFilter는 로그인 처리를 담당하기 때문에 다른 필터들과 달리 로그인을 처리하는 경로에 대한 설정과 실제 인증 처리를 담당하는 AuthenticationManager 객체의 설정이 필수인데 이에 대한 설정은 CustomSecurityConfig를 이용해서 처리
    - CustomSecurityConfig에는 APIUserDetailsService를 주입하도록 설정하고 이를 이용해서 AuthenticationManager를 생성하고 APILoginFilter를 설정

JWT 인증

❖ 토큰 인증을 위한 Security 필터

✓ AbstractAuthenticationProcessingFilter 설정을 위한 부분을 CustomSecurityConfig 에 설정

```
@Configuration
@Log4j2
@RequiredArgsConstructor
@EnableGlobalMethodSecurity(prePostEnabled = true)
@EnableWebSecurity
public class CustomSecurityConfig{
    private final APIUserDetailsService apiUserDetailsService;

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

    @Bean
    public WebSecurityCustomizer webSecurityCustomizer() {
        log.info("------------web configure------------------");
        return (web) ->
     web.ignoring().requestMatchers(PathRequest.toStaticResources().atCommonLocation
     s());
    }
```

JWT 인증

- 토큰 인증을 위한 Security 필터
  - ✓ AbstractAuthenticationProcessingFilter 설정을 위한 부분을 CustomSecurityConfig 에 설정

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    log.info("-------------------configure-----------------------------");
    //AuthenticationManager설정
    AuthenticationManagerBuilder authenticationManagerBuilder =
            http.getSharedObject(AuthenticationManagerBuilder.class);
    authenticationManagerBuilder.userDetailsService(apiUserDetailsService)
            .passwordEncoder(passwordEncoder());
    // Get AuthenticationManager
    AuthenticationManager authenticationManager =
 authenticationManagerBuilder.build();

    //반드시 필요
    http.authenticationManager(authenticationManager);
```

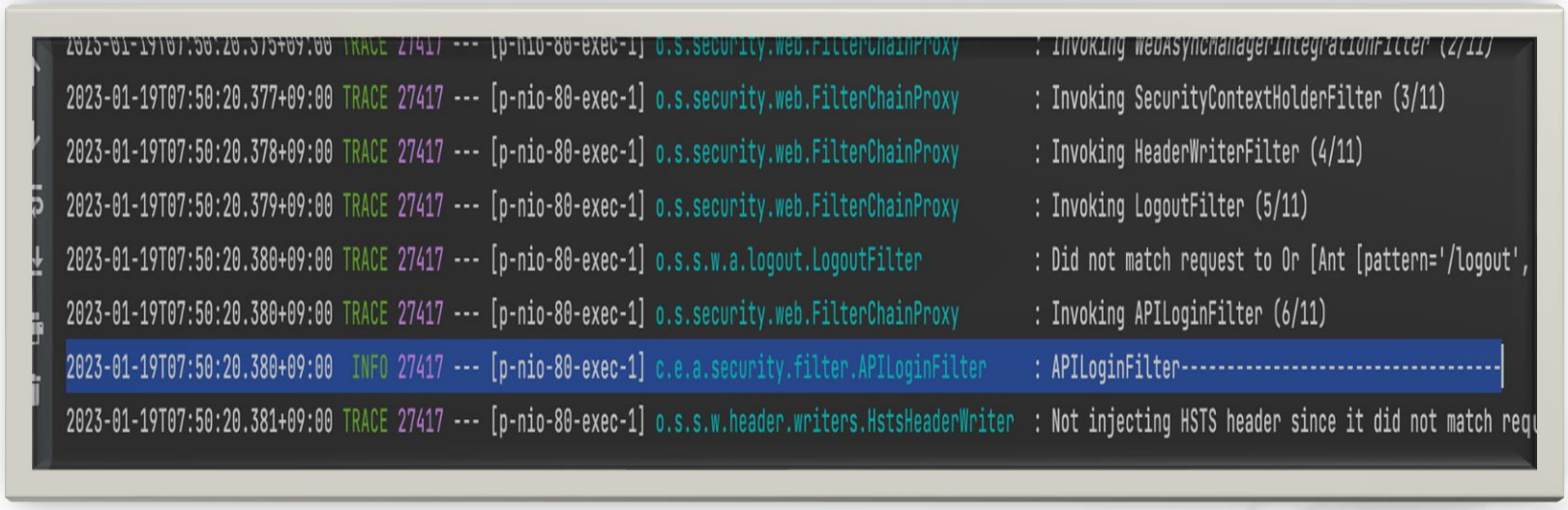
JWT 인증

- 토큰 인증을 위한 Security 필터
  - ✓ AbstractAuthenticationProcessingFilter 설정을 위한 부분을 CustomSecurityConfig 에 설정

```
            //APILoginFilter
            //스프링 Security에서 username 과 password를 처리하는
        UsernamePasswordAuthenticationFilter 의 앞쪽에서 동작하도록 설정
            APILoginFilter apiLoginFilter = new APILoginFilter("/generateToken");
            apiLoginFilter.setAuthenticationManager(authenticationManager);

            //APILoginFilter의 위치 조정
            http.addFilterBefore(apiLoginFilter, UsernamePasswordAuthenticationFilter.class);

            http.csrf().disable();
            //세션을 사용하지 않음

        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
            return http.build();
        }
    }
```

JWT 인증

- 토큰 인증을 위한 Security 필터
  - ✓ 실행 후 localhost/generateToken을 호출해서 필터가 동작하는지 확인



JWT 인증

- ❖ 토큰 인증을 위한 Security 필터
  - ✓ APILoginFilter 의 JSON 처리
    - APILoginFilter는 사용자의 아이디(mid)와 패스워드(mpw)를 이용해서 JWT 문자열을 생성하는 기능을 수행하기 위해서 사용자가 전달하는 mid, mpw 값을 알아낼 수 있어야 하는데 API 서버는 POST 방식으로 JSON 문자열을 이용하는 것이 일반적이므로 이를 APILoginFilter에 반영해야 함
    - JWT 문자열들을 얻기 위해서 전송되는 mid, mpw는 JSON 문자열로 전송되므로 HttpServletRequest로 처리하려면 JSON 처리를 쉽게 할 수 있는 라이브러리를 활용하는 것이 효율적
    - build.gradle 파일에 gson 라이브러리의 의존성을 추가

      implementation 'org.springframework.boot:spring-boot-starter-validation'

      implementation 'com.google.code.gson:gson:2.9.1'

JWT 인증

- ❖ 토큰 인증을 위한 Security 필터
  - ✓ APILoginFilter 의 JSON 처리
    - ● APILoginFilter 클래스를 JSON 문자열을 확인할 수 있도록 수정

```
@Log4j2
public class APILoginFilter extends AbstractAuthenticationProcessingFilter {

    public APILoginFilter(String defaultFilterProcessesUrl) {
        super(defaultFilterProcessesUrl);
    }

    //파라미터를 읽어서 Map으로 만들어주는 메서드
    private Map<String,String> parseRequestJSON(HttpServletRequest request) {
        //JSON 데이터를 분석해서 mid, mpw 전달 값을 Map으로 처리
        try(Reader reader = new InputStreamReader(request.getInputStream())){
            Gson gson = new Gson();
            return gson.fromJson(reader, Map.class);
        }catch(Exception e){
            log.error(e.getMessage());
        }
        return null;
    }
```

JWT 인증

❖ 토큰 인증을 위한 Security 필터

✓ APILoginFilter 의 JSON 처리

● APILoginFilter 클래스를 JSON 문자열을 확인할 수 있도록 수정

```
@Override
public Authentication attemptAuthentication(HttpServletRequest request,
 HttpServletResponse response) throws AuthenticationException, IOException {

    log.info("APILoginFilter----------------------------------");

    if(request.getMethod().equalsIgnoreCase("GET")){
        log.info("GET METHOD NOT SUPPORT");
        return null;
    }

    Map<String, String> jsonData = parseRequestJSON(request);
    log.info("jsonData: "+jsonData);

    return null;
  }
}
```

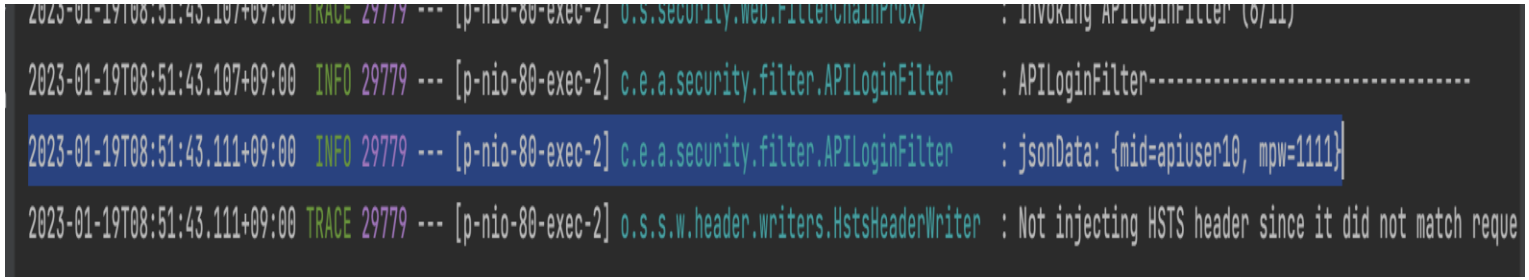
JWT 인증

- ❖ 토큰 인증을 위한 Security 필터
  - ✓ APILoginFilter 의 JSON 처리
    - static 디렉토리에 apiLogin.html 파일을 추가하고 작성

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<button id="btn1">generateToken</button>

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  document.getElementById("btn1").addEventListener("click",()=> {
    const data = {mid: "apiuser10", mpw: "1111"}
    axios.post("http://localhost/generateToken", data);
  });
</script>

</body>
</html>
```

JWT 인증

- ❖ 토큰 인증을 위한 Security 필터
  - ✓ APILoginFilter 의 JSON 처리
    - 실행하고 http://files/apiLogin.html 에 접속하고 콘솔 확인



JWT 인증

- ❖ 토큰 인증을 위한 Security 필터
  - ✓ APILoginFilter 의 JSON 처리
    - Map으로 처리된 mid, mpw를 이용해서 로그인을 처리하는 부분은 UsernamePasswordAuthenticationToken 인증 정보를 만들어서 다음 필터(UsernamePasswordAuthenticationFilter) 에서 사용하도록 APILoginFilter 클래스의 메서드를 수정

```
@Override
public Authentication attemptAuthentication(HttpServletRequest request,
    HttpServletResponse response) throws AuthenticationException, IOException {
        log.info("APILoginFilter----------------------------------");
        if(request.getMethod().equalsIgnoreCase("GET")){
            log.info("GET METHOD NOT SUPPORT");
            return null;
        }
        Map<String, String> jsonData = parseRequestJSON(request);
        log.info("jsonData: "+jsonData);
        UsernamePasswordAuthenticationToken authenticationToken =
                new UsernamePasswordAuthenticationToken(jsonData.get("mid"),
    jsonData.get("mpw"));

        return getAuthenticationManager().authenticate(authenticationToken);
}
```

JWT 인증

- ❖ 토큰 인증을 위한 Security 필터
  - ✓ APILoginFilter 의 JSON 처리
    - Map으로 처리된 mid, mpw를 이용해서 로그인을 처리하는 부분은 UsernamePasswordAuthenticationToken 인증 정보를 만들어서 다음 필터(UsernamePasswordAuthenticationFilter 에서 사용하도록 APILoginFilter 클래스의 메서드를 수정

```
2023-01-24T15:10:27.846+09:00 TRACE 91287 --- [p-nio-80-exec-2] o.s.s.authentication.ProviderManager     : Authenticating request with DaoAuthenticationProvider
Hibernate: select a1_0.mid,a1_0.mpw from apiuser a1_0 where a1_0.mid=?
2023-01-24T15:10:28.013+09:00  INFO 91287 --- [p-nio-80-exec-2] c.k.a.security.APIUserDetailsService     : APIUserDetailsService apiUser-----------------------
2023-01-24T15:10:28.015+09:00  INFO 91287 --- [p-nio-80-exec-2] c.k.a.security.APIUserDetailsService     : APIUserDTO(mid=apiuser10, mpw=$2a$10$OTZq4E/tY2cn4UHDW
2023-01-24T15:10:28.118+09:00 DEBUG 91287 --- [p-nio-80-exec-2] o.s.s.a.dao.DaoAuthenticationProvider    : Authenticated user
2023-01-24T15:10:28.120+09:00 TRACE 91287 --- [p-nio-80-exec-2] RequestAwareAuthenticationSuccessHandler : Using default url /
2023-01-24T15:10:28.120+09:00 DEBUG 91287 --- [p-nio-80-exec-2] o.s.s.web.DefaultRedirectStrategy        : Redirecting to /
```

JWT 인증

- 인증 성공 후 처리
  - ✓ 인증 성공 후 처리 작업을 담당하기 위한 클래스를 생성 – security.handler.APILoginSuccessHandler

```
@Log4j2
@RequiredArgsConstructor
public class APILoginSuccessHandler implements AuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
                                        HttpServletResponse response,
                                        Authentication authentication) throws IOException {

        log.info("Login Success Handler.....................");

        response.setContentType(MediaType.APPLICATION_JSON_VALUE);

    }
}
```

JWT 인증

- 인증 성공 후 처리
  - ✓ 인증 성공 후 처리 작업을 담당하기 위한 클래스를 적용 – CustomSecurityConfig 클래스의 filterChain 메서드에 추가

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("------------------configure-----------------------------");
        AuthenticationManagerBuilder authenticationManagerBuilder =
                http.getSharedObject(AuthenticationManagerBuilder.class);
        authenticationManagerBuilder.userDetailsService(apiUserDetailsService)
                .passwordEncoder(passwordEncoder());
        AuthenticationManager authenticationManager =
    authenticationManagerBuilder.build();
        //반드시 필요
        http.authenticationManager(authenticationManager);
        //APILoginFilter
        //스프링 Security에서 username 과 password를 처리하는
    UsernamePasswordAuthenticationFilter 의 앞쪽에서 동작하도록 설정
        APILoginFilter apiLoginFilter = new APILoginFilter("/generateToken");
        apiLoginFilter.setAuthenticationManager(authenticationManager);

        APILoginSuccessHandler successHandler = new APILoginSuccessHandler();
        apiLoginFilter.setAuthenticationSuccessHandler(successHandler);
```

JWT 인증

- 인증 성공 후 처리
  - ✓ 인증 성공 후 처리 작업을 담당하기 위한 클래스가 적용되는지 확인

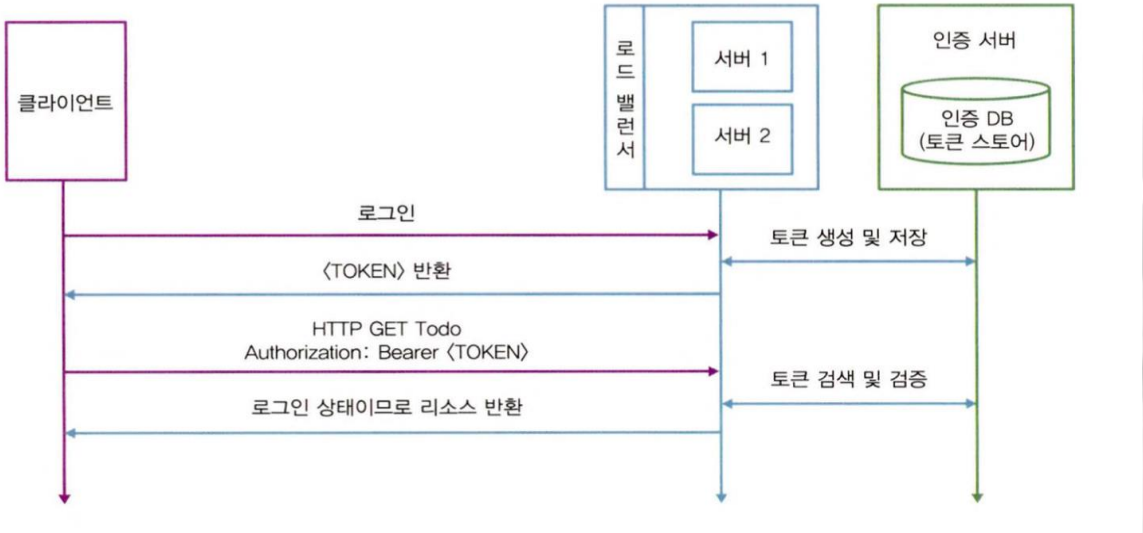
```
2023-01-24T15:34:29.191+09:00  INFO 92967 --- [p-nio-80-exec-2] c.k.a.security.APIUserDetailsService     : APIUserDTO(mid=apiuser10, mpw=$2a$10$OTZq4E/tY2cn4UHDW
2023-01-24T15:34:29.271+09:00 DEBUG 92967 --- [p-nio-80-exec-2] o.s.s.a.dao.DaoAuthenticationProvider    : Authenticated user
2023-01-24T15:34:29.272+09:00  INFO 92967 --- [p-nio-80-exec-2] c.k.a.s.handler.APILoginSuccessHandler   : Login Success Handler.....................
2023-01-24T15:34:29.272+09:00 TRACE 92967 --- [p-nio-80-exec-2] o.s.s.w.header.writers.HstsHeaderWriter  : Not injecting HSTS header since it did not match reque
```

JWT 인증

- 인증 기법
  - ✓ Cookie 와 Session 기반 인증
    - 클라이언트 와 서버에 정보를 남겨서 요청 마다 전송해서 인증하는 방식
    - 동일 도메인에서의 요청이 아니라면 쿠키가 넘어가지 않음(서버에서 쿠키를 거부)
    - javascript에서 ajax로 헤더에 쿠키를 강제로 담아서 보낼 수가 있는데 서버에서는 HTTP Only 설정을 통해 외부에서 HTTP 요청이 아닌 javascript 요청이 들어오면 거부되는 설정을 하는데 HTTP Only = false로 풀어주면 외부에서 javascript 로 조작을 많이 치기 때문에 true로 설정

JWT 인증

- 인증 기법
  - ✓ Token 기반 인증
    - 토큰은 사용자를 구별할 수 있는 문자열
    - 서버로 요청을 보낼 때 요청 헤더에 Authorization : <type> <credentials> 을 담아서 전송
    - 토큰은 최초 로그인 시 서버가 만들어 주는데 서버가 자기만의 노하우로 토큰을 만들어 반환하면 클라이언트는 이후 요청에 아이디와 비밀번호 대신 토큰을 계속 넘겨 자신이 인증된 사용자 임을 알리는 방식



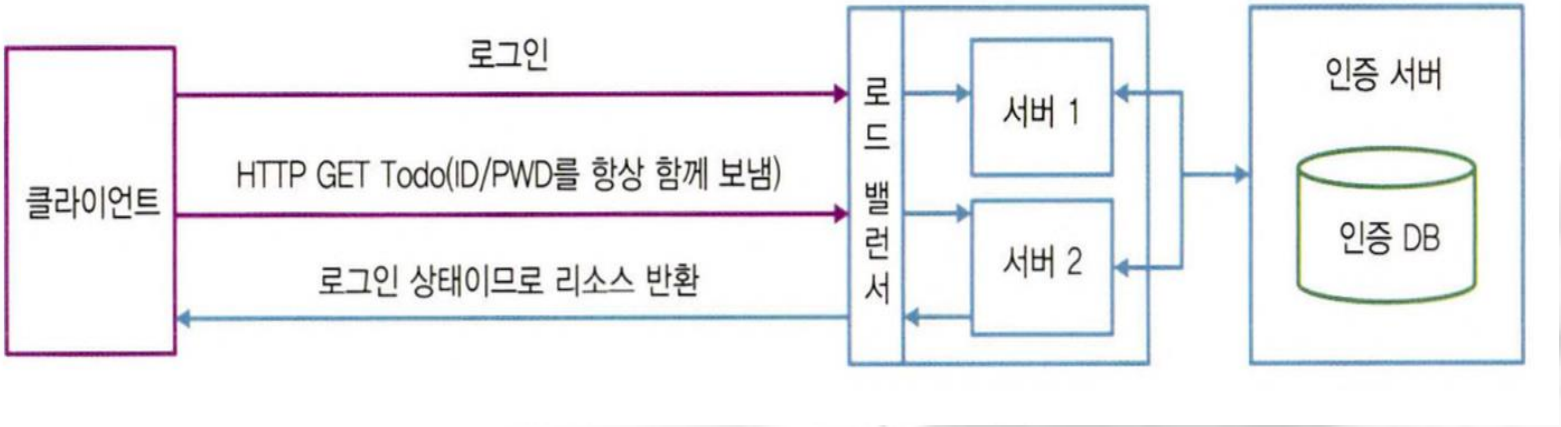
JWT 인증

- 인증 기법
  - ✓ Token 기반 인증
    - Basic Token
      - ◆ 아이디와 비밀번호를 콜론으로 붙인 후 Base64로 인코딩한 문자열을 함께 전송

        Authorization: Basic aGVsbG93b33sZEBnbWFpbC5jb206MTIzNA==
      - ◆ HTTP 요청을 수신한 서버는 인코딩 된 문자열을 디코딩해 아이디와 비밀번호를 찾아낸 후 사용자 정보가 저장된 데이터베이스 또는 인증 서버의 레코드와 비교한 후 아이디와 비밀번호가 일치하면 요청을 처리하고 아니면 거부
      - ◆ 이 방식은 아이디와 비밀번호를 노출하는데 인코딩은 보안을 목적으로 하는 것이 아니기 때문에 육안으로 아이디 와 비밀번호를 찾아내기 힘들지만 디코더만 있다면 누구나 디코딩해 원래의 아이디 와 비밀번호를 확인할 수 있기 때문에 Basic 인증은 HTTP 와 사용하기엔 취약해서 누군가 HTTP 요청을 가로채 문자열을 디코딩하면 아이디와 비밀번호를 알아낼 수 있기(MITM - Man In the Middle Attack) 때문에 반드시 HTTPS 와 사용

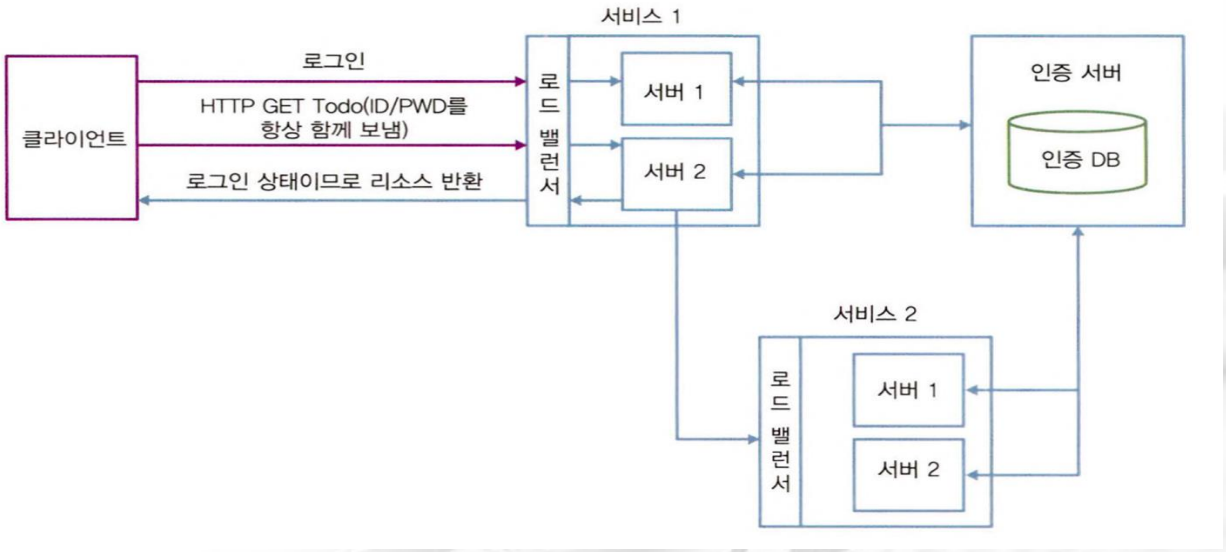
JWT 인증

- ❖ 인증 기법
  - ✓ Token 기반 인증
    - Basic Token
      - ◆ 이 방식을 사용하면 사용자를 로그아웃 시킬 수 없는데 모든 요청이 로그인 요청이기 때문
      - ◆ 여러 디바이스에서 로그인이 가능한 경우 한꺼번에 로그아웃을 하거나 디바이스별로 로그아웃을 할 수 있는 기능이 있는 애플리케이션들이 있는데 Basic Token을 이용하면 이런 기능을 제공하기 힘듬



JWT 인증

- ❖ 인증 기법
  - ✓ Token 기반 인증
    - ● Basic Token
      - ◆ 사용자의 계정 정보가 있는 저장 장소(Identity Management System) 인 인증 서버와 인증 DB에 과부하가 걸릴 확률이 높은데 1초에 10만 개를 처리하는 서버가 있다고 가정하면 10만 개의 요청을 확인하려면 10만 번 계정 정보의 저장 장소에 접근하게 되는데 최근에는 서비스가 커지면 마이크로서비스로 나누어서 인증 서버에 요청해야 하는 일이 너무 많아지기 때문에 문제가 발생할 가능성이 있음



JWT 인증

- ❖ 인증 기법
  - ✓ Token 기반 인증
    - Bearer Token
      - ◆ 서버가 랜덤한 문자와 숫자를 섞어 UUID로 토큰을 작성해 넘기면 서버는 토큰을 생성해 인증 서버를 통해 저장하고 요청을 받을 때마다 헤더의 토큰을 서버의 토큰과 비교해 클라이언트를 인증
      - ◆ 아이디와 비밀번호를 매번 네트워크를 통해 전송해야 할 필요가 없으므로 보안 측면에서 좀 더 안전하고 서버가 토큰을 마음대로 생성할 수 있으므로 사용자의 인가 정보(예: User, Admin 등) 또는 유효 시간을 정해 관리할 수 있으며 디바이스마다 다른 토큰을 생성해 주고 디바이스마다 유효 시간을 다르게 정하거나 임의로 로그아웃을 할 수 도 있음
      - ◆ 이 방식에서의 토큰은 이름만 바뀐 세션과 유사해서 스케일 문제를 해결할 수 없음

JWT 인증

- ❖ 인증 기법
  - ✓ Token 기반 인증
    - JSON Web Token
      - ◆ 전자 서명(Digital Signature)된 토큰을 이용해 스케일 문제를 해결
      - ◆ 전자 서명된 토큰 중 하나가 JSON Web Token
      - ◆ https://tools.ietf.org/html/rfc7519
      - ◆ JWT는 JSON 형태로 된 토큰으로 {header}.{payload}.{signature}로 구성
        - JWT Token

          Authorization: Bearer
          eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXN0ZXJAdGVzdC5jb20iLCJpYXQiOjEl
          OTU3MzM2NTcsImV4cCI6MTU5NjU5NzYlN30.Nn4dlMOVLZg79sfFACTIpCPKqWmpZMZQsbNrXdJJNWkRv50_l7bPLQPwhMobT4vBOG6Q33YjhDrKFlBSaUxZOg

JWT 인증

- ❖ 인증 기법
  - ✓ Token 기반 인증
    - ● JSON Web Token
      - ◆ JWT는 JSON 형태로 된 토큰으로 {header}.{payload}.{signature}로 구성
        - ▪ 디코딩 된 결과

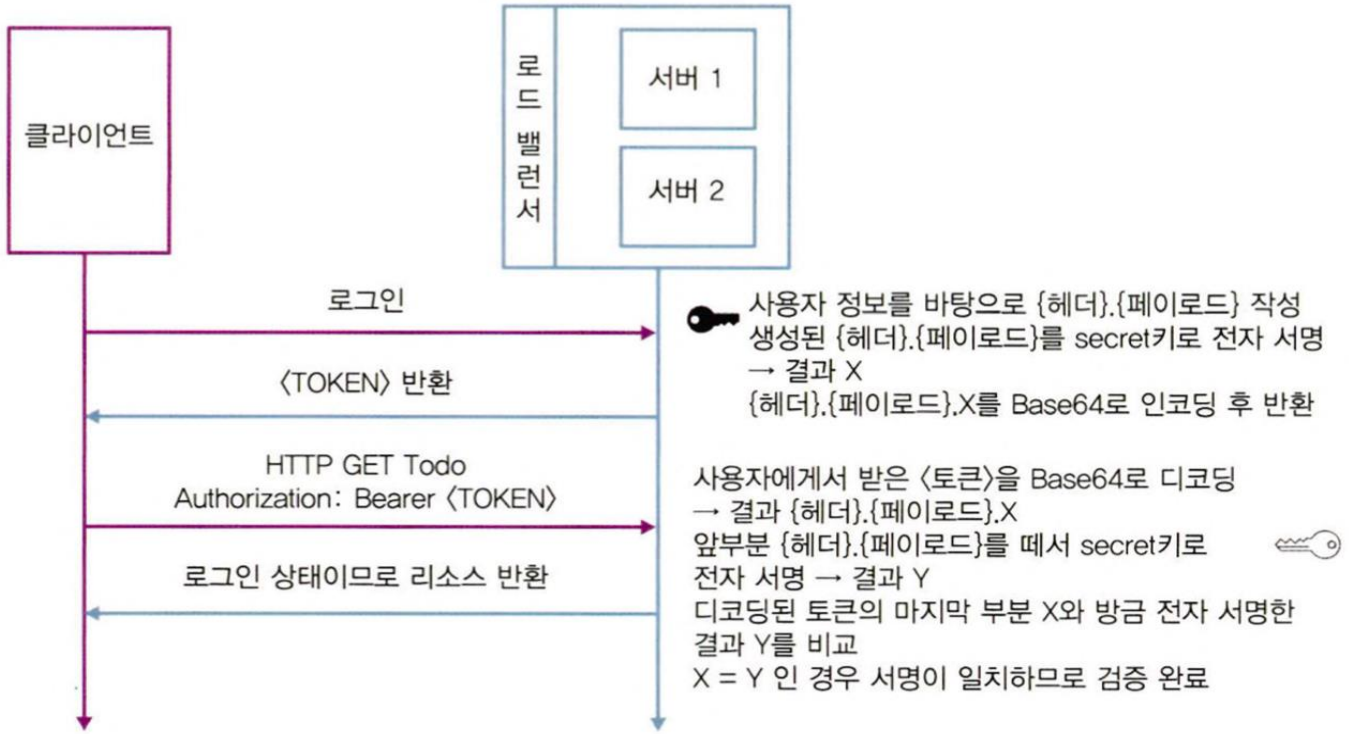
```
{ // header
 "typ" : "JWT",
 "alg" : "HS512"
},
{ // payload
 "sub":"40288093784915d201784916a40c0001",
 "iss" : "demo app",
 "iat" : 1595733657,
 "exp" : 1595733657
},
Nn4dlMOVLZg79sfFACTIpCPKqWmpZMZQsbNrXdJ]NWkRv50_17bPLQ
    PwhMobT4vBOG6Q3]YjhDrKFlBSaUxZOg
// signature
```

JWT 인증

- ❖ 인증 기법
  - ✓ Token 기반 인증
    - ● JSON Web Token
      - ◆ 구성
        - ▪ Header
          - typ: Type을 줄인 말로 토큰의 타입
          - alg: 토큰의 서명을 발행하는 데 사용된 해시 알고리즘의 종류
        - ▪ Payload
          - iss: Issuer를 줄인 말로 토큰을 발행한 주체
          - sub: Subject를 줄인 말로 토큰의 제목
          - exp: expiration을 줄인 말로 토큰이 만료되는 시간
          - iat: issued at을 줄인 말로 토큰이 발행된 날짜와 시간
          - Aud: 토큰 대상자
          - nfb: 토큰 활성 시간
          - jti: JWT 고유 식별자
        - ▪ Signature
          - 토큰을 발행한 Issuer가 발행한 서명으로 토큰의 유효성 검사에 사용

JWT 인증

- ❖ 인증 기법
  - ✓ Token 기반 인증
    - ● JSON Web Token
      - ◆ JWT 토큰 생성 과 인증



JWT 인증

- ❖ 인증 기법
  - ✓ Token 기반 인증
    - ● JSON Web Token
      - ◆ JWT 토큰 생성 과 인증
        - ▪ JWT 에서 전자 서명이란 {헤더}.{페이로드} 와 시크릿 키를 이용해 해시 함수에 돌린 암호화한 결과 값
        - ▪ 시크릿 키란 문자열, 비밀번호 같은 것으로 너무 간단하지만 않으면 아무것이나 상관없음
        - ▪ 최초 로그인 시 서버는 사용자의 아이디와 비밀번호를 서버에 저장된 아이디와 비밀번호에 비교해 인증하는데 인증된 사용자인 경우 사용자의 정보를 이용해 {헤더}. {페이로드} 부분을 작성하고 자신의 시크릿 키로 {헤더}.{페이로드} 부분을 전자 서명한 후 전자 서명의 결과로 나온 값을 {헤더}.{페이로드}.{서명}으로 이어붙이고 Base64로 인코딩 한 후 반환
        - ▪ 이후에 누군가 이 토큰으로 리소스 접근을 요청하면 서버는 일단 이 토큰을 Base64로 디코딩하고 디코딩해서 얻은 JSON을 {헤더}.{페이로드} 와 {서명} 부분으로 나누고 서버는 {헤더}.{페이로드} 와 자신이 갖고 있는 Secret 으로 전자 서명을 만든 후 방금 만든 전자 서명을 HTTP 요청이 갖고 온 {서명} 부분과 비교해 이 토큰의 유효성을 검사하고 서버가 시크릿 키를 이용해 만든 전자 서명과 HTTP 요청의 {서명} 부분이 일치하면 토큰이 위조되지 않았다는 의미

JWT 인증

- ❖ 인증 기법
  - ✓ Token 기반 인증
    - ● JSON Web Token
      - ◆ JWT 토큰 생성 과 인증
        - ▪ 헤더나 페이로드 부분을 변경했다면 서명이 일치하지 않음
        - ▪ 이 방식은 인증 서버에 토큰의 유효성에 대해 물어 볼 필요가 없는데 이는 인증 서버에 부하를 일으키지 않는다는 의미이고 더 이상 인증 서버가 단일 장애점이 아니라는 의미이기도 함
        - ▪ 토큰을 훔쳐가면 해당 계정의 리소스에 접근할 수 있기 때문에 반드시 HTTPS를 통해 통신해야 함

JWT 인증

- JWT 생성 과 인증
  - ✓ build.gradle 파일에 JWT를 생성하고 넘겨받은 JWT를 확인할 수 있는 라이브러리 의존성 설정

```
implementation 'javax.xml.bind:jaxb-api:2.3.0'
implementation 'io.jsonwebtoken:jjwt:0.9.1'
```

JWT 인증

- JWT 생성 과 인증
  - ✓ application.yml 파일에 비밀키 등록

```
com:
  kakao:
    apiserver:
      secret: adam1234567890
```

JWT 인증

- JWT 생성 과 인증
  - ✓ JWT를 쉽게 이용하기 위한 클래스 생성 – util.JWTUtil

```
@Component
@Log4j2
public class JWTUtil {

    @Value("${com.kakao.apiserver.secret}")
    private String key;

    public String generateToken(Map<String, Object> valueMap, int days){
        log.info("generateKey..." + key);
        return null;
    }


    public Map<String, Object> validateToken(String token)throws JwtException {
        Map<String, Object> claim = null;
        return claim;
    }
}
```

JWT 인증

- JWT 생성 과 인증
  - ✓ 비밀키 로드 확인 – JWTUtilTests.java

```
@SpringBootTest
@Log4j2
public class JWTUtilTests {

    @Autowired
    private JWTUtil jwtUtil;

    @Test
    public void testGenerate() {
        Map<String, Object> claimMap = Map.of("mid","ABCDE");
        String jwtStr = jwtUtil.generateToken(claimMap,1);
        System.out.println(jwtStr);
    }
}
```

JWT 인증

- JWT 생성 과 인증
  - ✓ 비밀키 로드 확인 – JWTUtilTests.java

```
: ------------web configure------------------
: You are asking Spring Security to ignore org.springframework.boot.autoconfigure.security.servlet.StaticResourceRequest$Stat
: Will not secure org.springframework.boot.autoconfigure.security.servlet.StaticResourceRequest$StaticResourceRequestMatcher@
: Started JWTUtilTests in 9.275 seconds (process running for 10.508)
: generateKey...adam1234567890

: Closing JPA EntityManagerFactory for persistence unit 'default'
: HikariPool-1 - Shutdown initiated...
: HikariPool-1 - Shutdown completed.
```

JWT 인증

- JWT 생성 과 인증
  - ✓ JWT 생성을 위한 메서드 수정 – util.JWTUtil 수정

```
public String generateToken(Map<String, Object> valueMap, int days){

    log.info("generateKey..." + key);
    //헤더 부분
    Map<String, Object> headers = new HashMap<>();
    headers.put("typ","JWT");
    headers.put("alg","HS256");

    //payload 부분 설정
    Map<String, Object> payloads = new HashMap<>();
    payloads.putAll(valueMap);

    //테스트 시에는 짧은 유효 기간
    int time = (1) * days; //테스트는 분 단위로 나중에 60*24 (일)단위변경

    //10분 단위로 조정
    //int time = (10) * days; //테스트는 분단위로 나중에 60*24 (일)단위변경
```

JWT 인증

- JWT 생성 과 인증
  - ✓ JWT 생성을 위한 메서드 수정 – util.JWTUtil 수정

```
        String jwtStr = Jwts.builder()
                .setHeader(headers)
                .setClaims(payloads)
                .setIssuedAt(Date.from(ZonedDateTime.now().toInstant()))
                .setExpiration(Date.from(ZonedDateTime.now().plusMinutes(time).toInstant()
    ))
                .signWith(SignatureAlgorithm.HS256, key.getBytes())
                .compact();

        return jwtStr;
    }
```

JWT 인증

- JWT 생성 과 인증
  - ✓ 테스트:
    eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NzQ1NDQxMDIsIm1pZCI6IkFCQ0RFIiwiaWF0IjoxNjc0NTQ0MDQyfQ.hZc6a8yhj7vHRV6NsZaCtE23MAru33qF6XQ9LCurT-I

```
2023-01-24T16:07:22.439+09:00  INFO 95280 --- [    Test worker] com.kakao.apiserver.util.JWTUtil         : generateKey...adam123
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NzQ1NDQxMDIsIm1pZCI6IkFCQ0RFIiwiaWF0IjoxNjc0NTQ0MDQyfQ.hZc6a8yhj7vHRV6NsZaCtE23
2023-01-24T16:07:22.515+09:00  INFO 95280 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityMan
2023-01-24T16:07:22.517+09:00  INFO 95280 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdo
2023-01-24T16:07:22.537+09:00  INFO 95280 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdo
```

JWT 인증

- JWT 생성 과 인증
  - ✓ 문자열 검증을 위한 메서드 수정 – util.JWTUtil 수정

```
public Map<String, Object> validateToken(String token)throws JwtException {
        Map<String, Object> claim = null;

        claim = Jwts.parser()
                .setSigningKey(key.getBytes()) // Set Key
                .parseClaimsJws(token) // 파싱 및 검증, 실패 시 에러
                .getBody();
        return claim;
}
```

JWT 인증

- JWT 생성 과 인증
  - ✓ 테스트 클래스에서 확인 –JWTUtilTests 수정

```
@Test
public void testValidate(){
        String jwtStr =
    "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NzQ1NDQxMDIsIm1pZCI6IkFC
    Q0RFIiwiaWF0IjoxNjc0NTQ0MDQyfQ.hZc6a8yhj7vHRV6NsZaCtE23MAru33qF6XQ9LC
    urT-I";
        Map<String, Object> claim = jwtUtil.validateToken(jwtStr);
        System.out.println(claim);
}
```

```
sonwebtoken.ExpiredJwtException: JWT expired at 2023-01-24T16:08:22Z. Current time: 2023-01-24T16:16:16Z, a difference of 47411
at app//io.jsonwebtoken.impl.DefaultJwtParser.parse(DefaultJwtParser.java:385)
at app//io.jsonwebtoken.impl.DefaultJwtParser.parse(DefaultJwtParser.java:481)
at app//io.jsonwebtoken.impl.DefaultJwtParser.parseClaimsJws(DefaultJwtParser.java:541)
at app//com.kakao.apiserver.util.JWTUtil.validateToken(JWTUtil.java:57)
at app//com.kakao.apiserver.JWTUtilTests.testValidate(JWTUtilTests.java:28)
```

JWT 인증

- JWT 생성 과 인증
  - ✓ 유효 시간을 10분 단위로 변경 – JWTUtilTests 수정

```
//테스트 시에는 짧은 유효 기간
//int time = (1) * days; //테스트는 분단위로 나중에 60*24 (일)단위변경

//10분 단위로 조정
int time = (10) * days; //테스트는 분단위로 나중에 60*24 (일)단위변경
```

JWT 인증

- JWT 생성 과 인증
  - ✓ 테스트 클래스에서 확인 –JWTUtilTests 수정

```
{exp=1674545441, mid=ABCDE, iat=1674544841}
2023-01-24T16:21:11.478+09:00  INFO 96292 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityMar
2023-01-24T16:21:11.480+09:00  INFO 96292 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdo
2023-01-24T16:21:11.500+09:00  INFO 96292 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdo
```

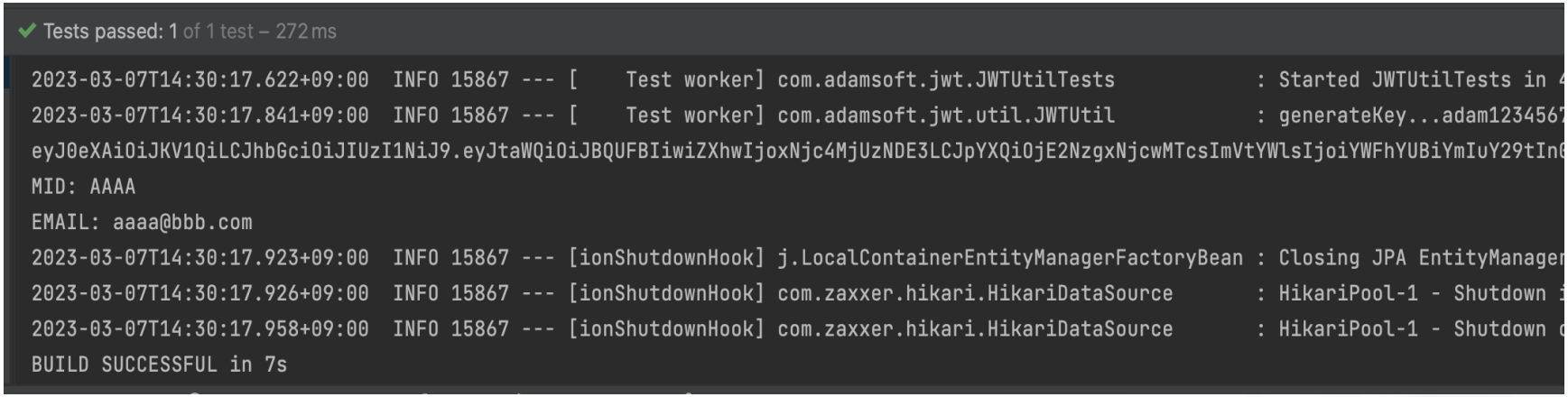
JWT 인증

- JWT 생성 과 인증
  - ✓ 테스트 클래스에서 확인 –JWTUtilTests 수정

```
@Test
public void testAll() {
        String jwtStr =
    jwtUtil.generateToken(Map.of("mid","AAAA","email","aaaa@bbb.com"),1);
        System.out.println(jwtStr);
        Map<String, Object> claim = jwtUtil.validateToken(jwtStr);
        System.out.println("MID: " + claim.get("mid"));
        System.out.println("EMAIL: " + claim.get("email"));
}
```

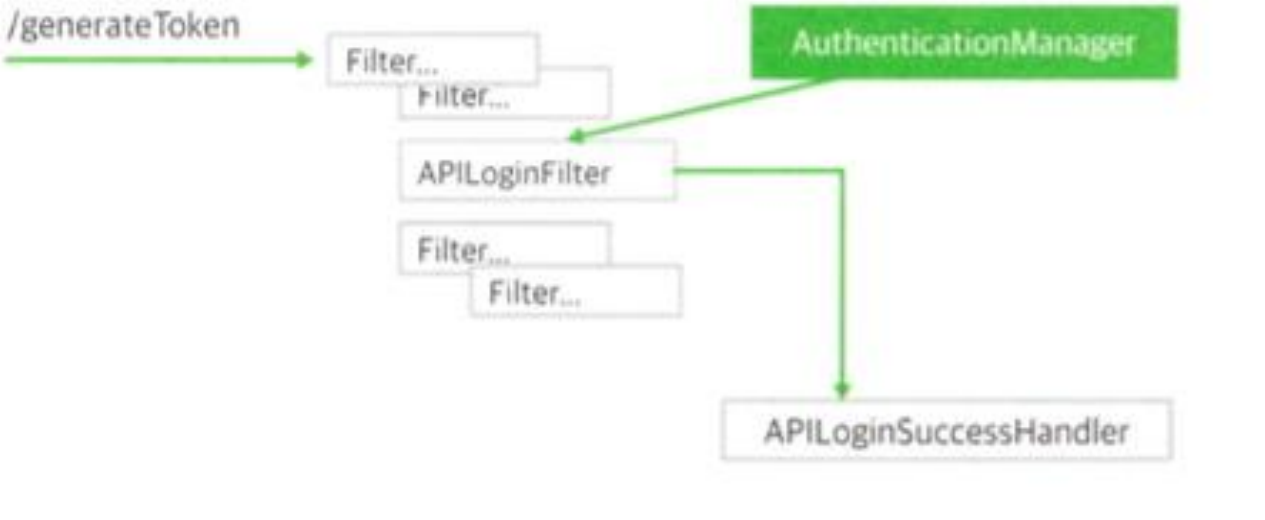
JWT 인증

- JWT 생성 과 인증
  - ✓ 테스트 클래스에서 확인 –JWTUtilTests 수정



JWT 인증

- Access Token 발행
  - ✓ 활용 방식
    - 사용자가 /generateToken 을 POST 방식으로 필요한 정보(mid, mpw)를 전달하면 APILoginFilter가 동작하고 인증 처리가 된 후에는 APILoginSuccessHandler 가 동작하는데 이 클래스의 내부에서는 인증된 사용자에게 Access Token / Refresh Token을 발행해 주기 위해서 JWTUtil을 이용



JWT 인증

❖ Access Token 발행

✓ APILoginSuccessHandler 클래스 수정 – JWTUtil을 주입하고 필요한 코드 작성

```
@Log4j2
@RequiredArgsConstructor
public class APILoginSuccessHandler implements AuthenticationSuccessHandler {
    private final JWTUtil jwtUtil;
```

JWT 인증

❖ Access Token 발행

✓ APILoginSuccessHandler 클래스 수정 – JWTUtil을 주입하고 필요한 코드 작성

```
@Override
public void onAuthenticationSuccess(HttpServletRequest request,
                                    HttpServletResponse response,
                                    Authentication authentication) throws IOException {

    log.info("Login Success Handler.....................");

    response.setContentType(MediaType.APPLICATION_JSON_VALUE);

    log.info(authentication);
    log.info(authentication.getName()); //username

    Map<String, Object> claim = Map.of("mid", authentication.getName());
    //Access Token 유효기간 1일
    String accessToken = jwtUtil.generateToken(claim, 1);
    //Refresh Token 유효기간 30일
    String refreshToken = jwtUtil.generateToken(claim, 10);
```

JWT 인증

- Access Token 발행
  - ✓ APILoginSuccessHandler 클래스 수정 – JWTUtil을 주입하고 필요한 코드 작성

```
        Gson gson = new Gson();

        Map<String,String> keyMap = Map.of(
                "accessToken", accessToken,
                "refreshToken", refreshToken);

        String jsonStr = gson.toJson(keyMap);

        response.getWriter().println(jsonStr);
    }
}
```

JWT 인증

- Access Token 발행
  - ✓ CustomSecurityConfig 클래스 수정

```
public class CustomSecurityConfig{
    private final APIUserDetailsService apiUserDetailsService;
    private final JWTUtil jwtUtil;

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

JWT 인증

- Access Token 발행
  - ✓ CustomSecurityConfig 클래스 수정

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    log.info("-------------------configure-----------------------------");

    //AuthenticationManager설정
    AuthenticationManagerBuilder authenticationManagerBuilder =
            http.getSharedObject(AuthenticationManagerBuilder.class);
    authenticationManagerBuilder.userDetailsService(apiUserDetailsService)
            .passwordEncoder(passwordEncoder());
    // Get AuthenticationManager
    AuthenticationManager authenticationManager =
authenticationManagerBuilder.build();

    //반드시 필요
    http.authenticationManager(authenticationManager);
    //APILoginFilter
    //스프링 Security에서 username 과 password를 처리하는
UsernamePasswordAuthenticationFilter 의 앞쪽에서 동작하도록 설정
    APILoginFilter apiLoginFilter = new APILoginFilter("/generateToken");
    apiLoginFilter.setAuthenticationManager(authenticationManager);
```

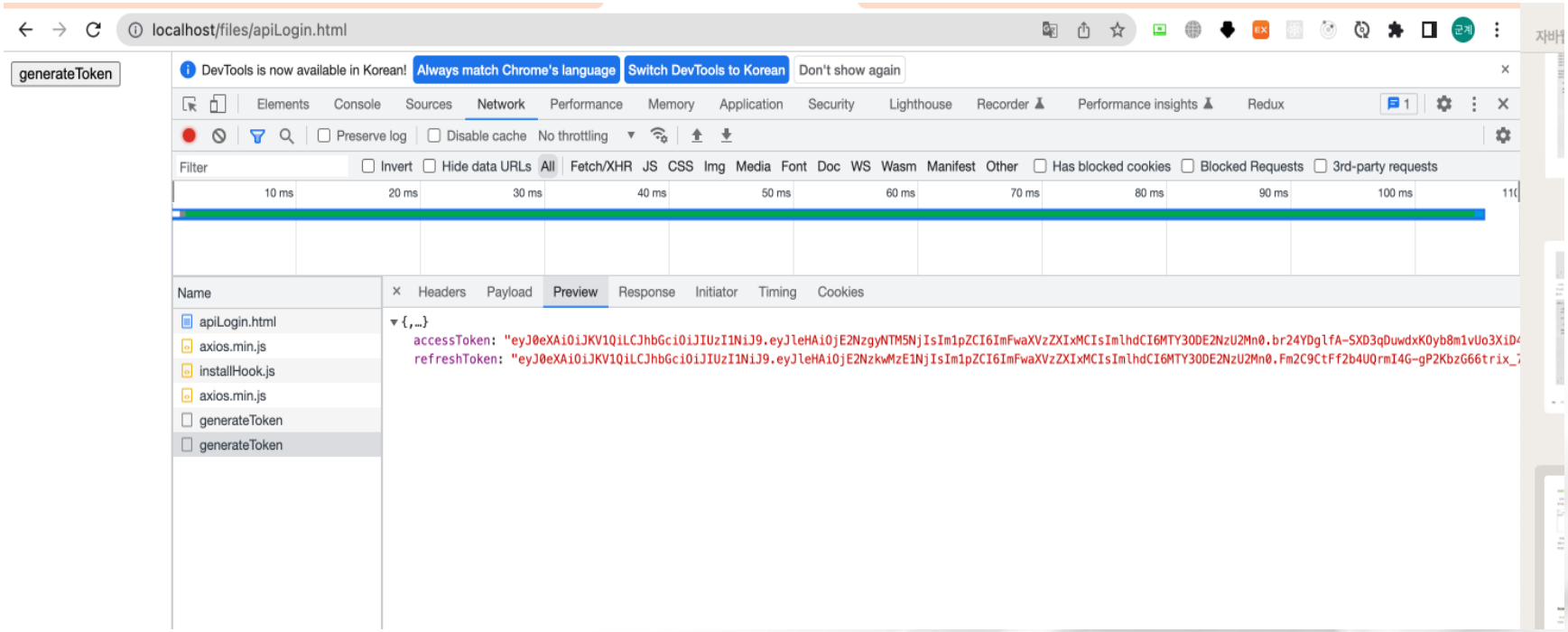
JWT 인증

- Access Token 발행
  - ✓ CustomSecurityConfig 클래스 수정

```
        APILoginSuccessHandler successHandler = new APILoginSuccessHandler(jwtUtil);
        apiLoginFilter.setAuthenticationSuccessHandler(successHandler);

        //APILoginFilter의 위치 조정
        http.addFilterBefore(apiLoginFilter, UsernamePasswordAuthenticationFilter.class);

        http.csrf().disable();
        //세션을 사용하지 않음

    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        return http.build();
    }

    @Bean
    public WebSecurityCustomizer webSecurityCustomizer() {
        log.info("------------web configure-------------------");
        return (web) ->
     web.ignoring().requestMatchers(PathRequest.toStaticResources().atCommonLocation
     s());
    }
}
```

JWT 인증

- ❖ Access Token 발행
  - ✓ 실행하고 토큰 생성 확인



JWT 인증

- Access Token 검증 필터
  - ✓ TokenCheckFilter 클래스(하나의 요청에 대해서 한 번씩 동작하는 필터 상속) 생성 – security.filter.TokenCheckFilter

```
@Log4j2
@RequiredArgsConstructor
public class TokenCheckFilter extends OncePerRequestFilter {
    private final JWTUtil jwtUtil;
    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws IOException, ServletException
    {
        String path = request.getRequestURI();
        if (!path.startsWith("/api/")) {
            filterChain.doFilter(request, response);
            return;
        }
        log.info("Token Check Filter.....");
        log.info("JWTUtil: " + jwtUtil);

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

JWT 인증

- Access Token 검증 필터
  - ✓ CustomSecurityConfig 클래스에 메서드 추가

```
private TokenCheckFilter tokenCheckFilter(JWTUtil jwtUtil){
        return new TokenCheckFilter(jwtUtil);
}
```

JWT 인증

- Access Token 검증 필터
  - ✓ CustomSecurityConfig 클래스의 filterChain 메서드 수정

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    log.info("-------------------configure-----------------------------");

    //AuthenticationManager설정
    AuthenticationManagerBuilder authenticationManagerBuilder =
            http.getSharedObject(AuthenticationManagerBuilder.class);
    authenticationManagerBuilder.userDetailsService(apiUserDetailsService)
            .passwordEncoder(passwordEncoder());
    // Get AuthenticationManager
    AuthenticationManager authenticationManager =
authenticationManagerBuilder.build();

    //반드시 필요
    http.authenticationManager(authenticationManager);
    //APILoginFilter
    //스프링 Security에서 username 과 password를 처리하는
UsernamePasswordAuthenticationFilter 의 앞쪽에서 동작하도록 설정
    APILoginFilter apiLoginFilter = new APILoginFilter("/generateToken");
    apiLoginFilter.setAuthenticationManager(authenticationManager);
```

JWT 인증

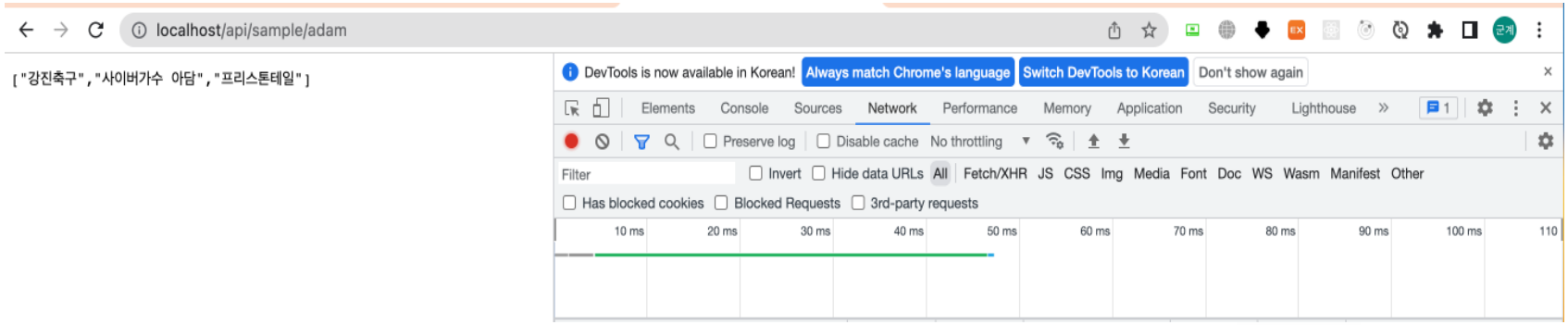
❖ Access Token 검증 필터

✓ CustomSecurityConfig 클래스의 filterChain 메서드 수정

```
        APILoginSuccessHandler successHandler = new APILoginSuccessHandler(jwtUtil);
        apiLoginFilter.setAuthenticationSuccessHandler(successHandler);

        //APILoginFilter의 위치 조정
        http.addFilterBefore(apiLoginFilter, UsernamePasswordAuthenticationFilter.class);

        http.addFilterBefore(tokenCheckFilter(jwtUtil),
    UsernamePasswordAuthenticationFilter.class);

        http.csrf().disable();
        //세션을 사용하지 않음

    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        return http.build();
    }
```

JWT 인증

- Access Token 검증 필터
  - ✓ 프로젝트 실행 후 확인




```
- [p-nio-80-exec-4] o.s.security.web.FilterChainProxy        : Invoking APILoginFilter (6/12)
- [p-nio-80-exec-4] o.s.security.web.FilterChainProxy        : Invoking TokenCheckFilter (7/12)
- [p-nio-80-exec-4] c.k.a.security.filter.TokenCheckFilter   : Token Check Filter.....................
- [p-nio-80-exec-4] c.k.a.security.filter.TokenCheckFilter   : JWTUtil: com.kakao.apiserver.util.JWTUtil@3579caff
- [p-nio-80-exec-4] o.s.security.web.FilterChainProxy        : Invoking RequestCacheAwareFilter (8/12)
- [p-nio-80-exec-4] o.s.security.web.FilterChainProxy        : Invoking SecurityContextHolderAwareRequestFilter (9/12)
```

JWT 인증

- Access Token 검증 필터
  - ✓ TokenCheckFilter 내 토큰 추출
    - TokenCheckFilter 클래스는 /api/로 시작하는 모든 경로의 호출에 사용되고 사용자는 해당 경로에 다음과 같은 상황으로 접근
      - Access Token이 없는 경우 - 토큰이 없다는 메시지 전달
      - Access Token이 잘못된 경우(서명 혹은 구성, 기타 에러) - 잘못된 토큰이라는 메시지 전달
      - Access Token이 존재하지만 오래된(expired) 값인 경우 - 토큰을 갱신하라는 메시지 전달
    - Access Token의 추출과 검증
      - Access Token의 값은 HTTP Header 중에 Authorization을 이용해서 전달
      - Authorization 헤더는 type + 인증값으로 작성되는데 type 값들은 Basic, Bearer, Digest, HOBA, Mutual 등을 이용하는데 이중에서 OAuth나 JWT는 Bearer라는 타입을 이용
      - TokenCheckFilter에서는 별도의 메서드를 이용해서 Authorization 헤더를 추출하고 Access Token을 검사하도록 함

JWT 인증

- Access Token 검증 필터
  - ✓ AccessToken에 문제가 있는 경우 발생할 사용자 정의 예외 클래스 생성 – security.AccessTokenException

```
public class AccessTokenException extends RuntimeException {

    TOKEN_ERROR token_error;
```

JWT 인증

- Access Token 검증 필터
  - ✓ AccessToken에 문제가 있는 경우 발생할 사용자 정의 예외 클래스 생성 – security.AccessTokenException

```
public enum TOKEN_ERROR {
    UNACCEPT(401,"Token is null or too short"),
    BADTYPE(401, "Token type Bearer"),
    MALFORM(403, "Malformed Token"),
    BADSIGN(403, "BadSignatured Token"),
    EXPIRED(403, "Expired Token");
    private int status;
    private String msg;

    TOKEN_ERROR(int status, String msg){
        this.status = status;
        this.msg = msg;
    }
    public int getStatus() {
        return this.status;
    }
    public String getMsg() {
        return this.msg;
    }
}
```

JWT 인증

- Access Token 검증 필터
  - ✓ AccessToken에 문제가 있는 경우 발생할 사용자 정의 예외 클래스 생성 – security.AccessTokenException

```
public AccessTokenException(TOKEN_ERROR error){
    super(error.name());
    this.token_error = error;
}

public void sendResponseError(HttpServletResponse response){
    response.setStatus(token_error.getStatus());
    response.setContentType(MediaType.APPLICATION_JSON_VALUE);

    Gson gson = new Gson();

    String responseStr = gson.toJson(Map.of("msg", token_error.getMsg(), "time",
 new Date()));

    try {
        response.getWriter().println(responseStr);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
}
```

JWT 인증

- Access Token 검증 필터
  - ✓ TokenCheckFilter 클래스에 Access Token을 검증하는 메서드를 추가

```
@Log4j2
@RequiredArgsConstructor
public class TokenCheckFilter extends OncePerRequestFilter {

    private final JWTUtil jwtUtil;
```

JWT 인증

- Access Token 검증 필터
  - ✓ TokenCheckFilter 클래스에 Access Token을 검증하는 메서드를 추가

```
private Map<String, Object> validateAccessToken(HttpServletRequest request)
 throws AccessTokenException {

    String headerStr = request.getHeader("Authorization");

    if(headerStr == null  || headerStr.length() < 8){
        throw new
AccessTokenException(AccessTokenException.TOKEN_ERROR.UNACCEPT);
    }

    //Bearer 생략
    String tokenType = headerStr.substring(0,6);
    String tokenStr =  headerStr.substring(7);

    if(tokenType.equalsIgnoreCase("Bearer") == false){
        throw new
AccessTokenException(AccessTokenException.TOKEN_ERROR.BADTYPE);
    }
```

JWT 인증

❖ Access Token 검증 필터

✓ TokenCheckFilter 클래스에 Access Token을 검증하는 메서드를 추가

```
        try{
            Map<String, Object> values = jwtUtil.validateToken(tokenStr);

            return values;
        }catch(MalformedJwtException malformedJwtException){
            log.error("MalformedJwtException----------------------");
            throw new
    AccessTokenException(AccessTokenException.TOKEN_ERROR.MALFORM);
        }catch(SignatureException signatureException){
            log.error("SignatureException----------------------");
            throw new
    AccessTokenException(AccessTokenException.TOKEN_ERROR.BADSIGN);
        }catch(ExpiredJwtException expiredJwtException){
            log.error("ExpiredJwtException----------------------");
            throw new
    AccessTokenException(AccessTokenException.TOKEN_ERROR.EXPIRED);
        }
    }
```

JWT 인증

- Access Token 검증 필터
  - ✓ TokenCheckFilter 클래스에 Access Token을 검증하는 메서드를 추가

```
@Override
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain filterChain) throws IOException, ServletException
{

    String path = request.getRequestURI();


    if (!path.startsWith("/api/")) {
        filterChain.doFilter(request, response);
        return;
    }

    log.info("Token Check Filter.....................");
    log.info("JWTUtil: " + jwtUtil);
```

JWT 인증

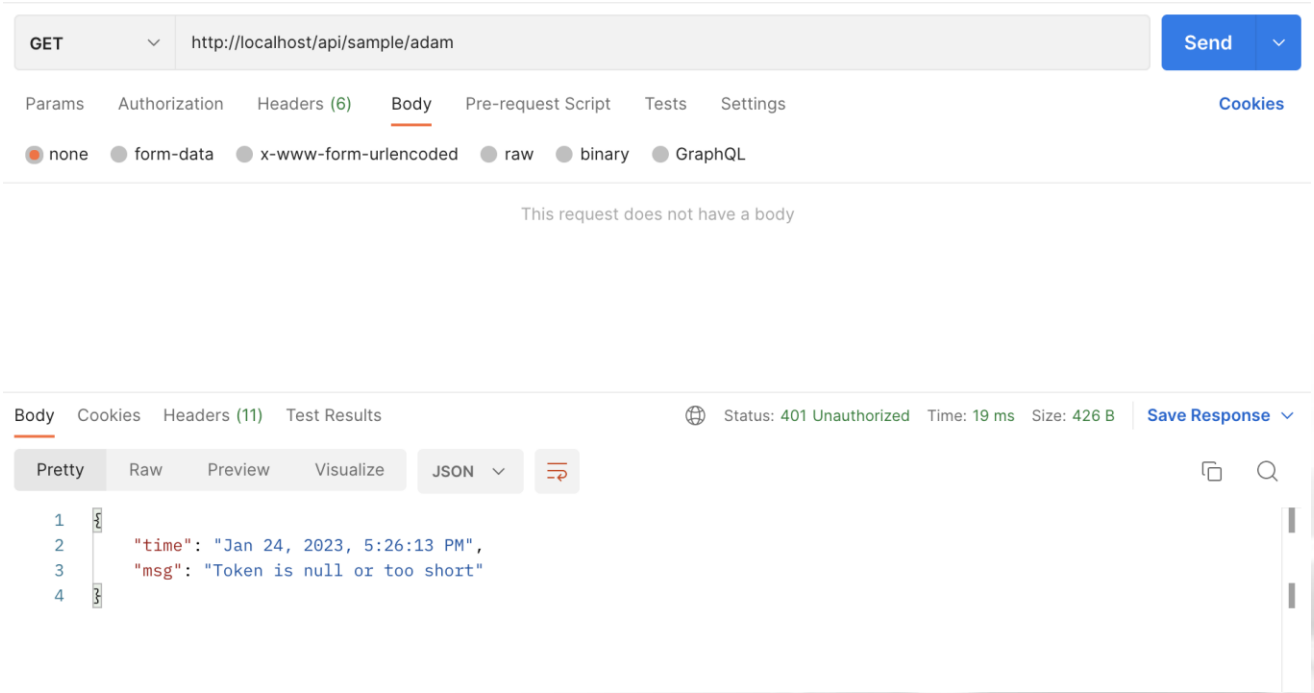
- Access Token 검증 필터
  - ✓ TokenCheckFilter 클래스에 Access Token을 검증하는 메서드를 추가

```
            try {
                validateAccessToken(request);
                filterChain.doFilter(request, response);
            }catch(AccessTokenException accessTokenException){
                accessTokenException.sendResponseError(response);
            }
        }
    }
```

JWT 인증

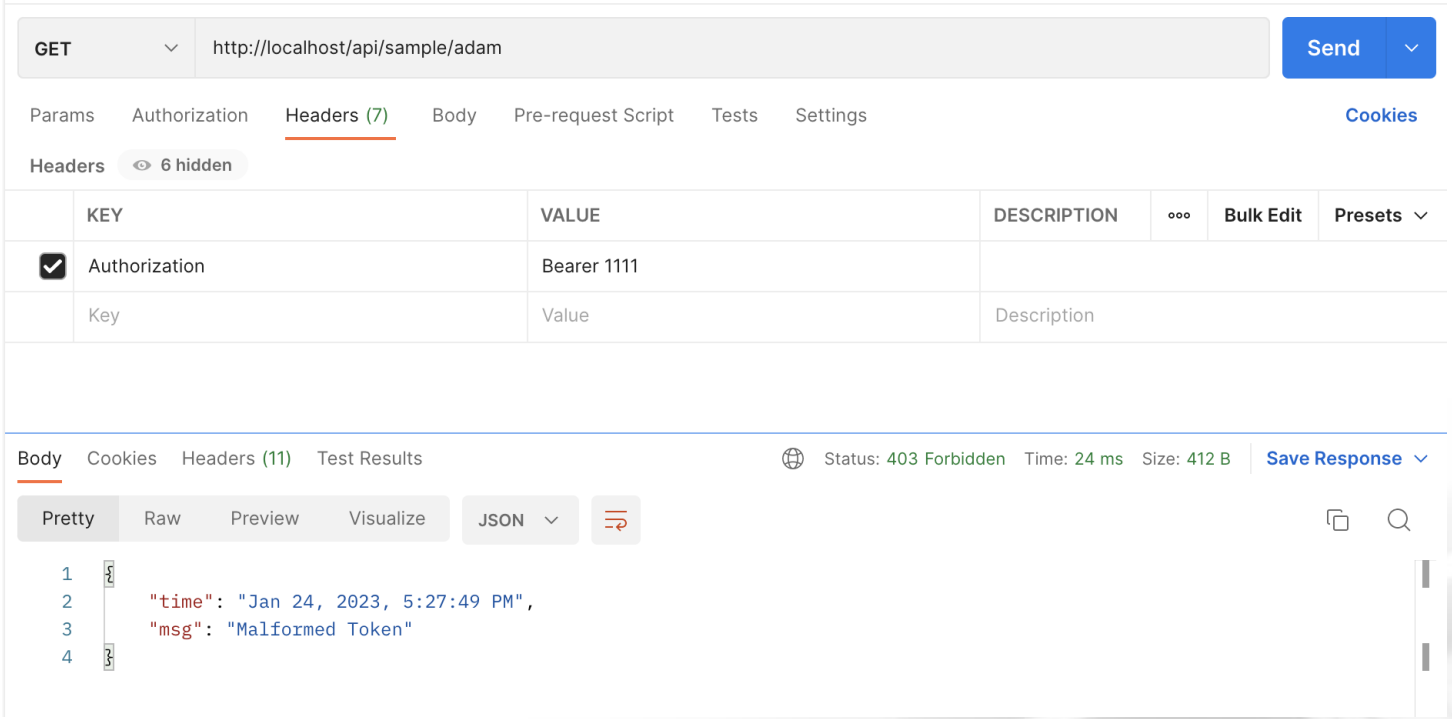
❖ Access Token 검증 필터

✓ 실행 – Authorization이 설정되지 않은 경우



JWT 인증

- Access Token 검증 필터
  - ✓ 실행 – 잘못된 Token 전달



JWT 인증

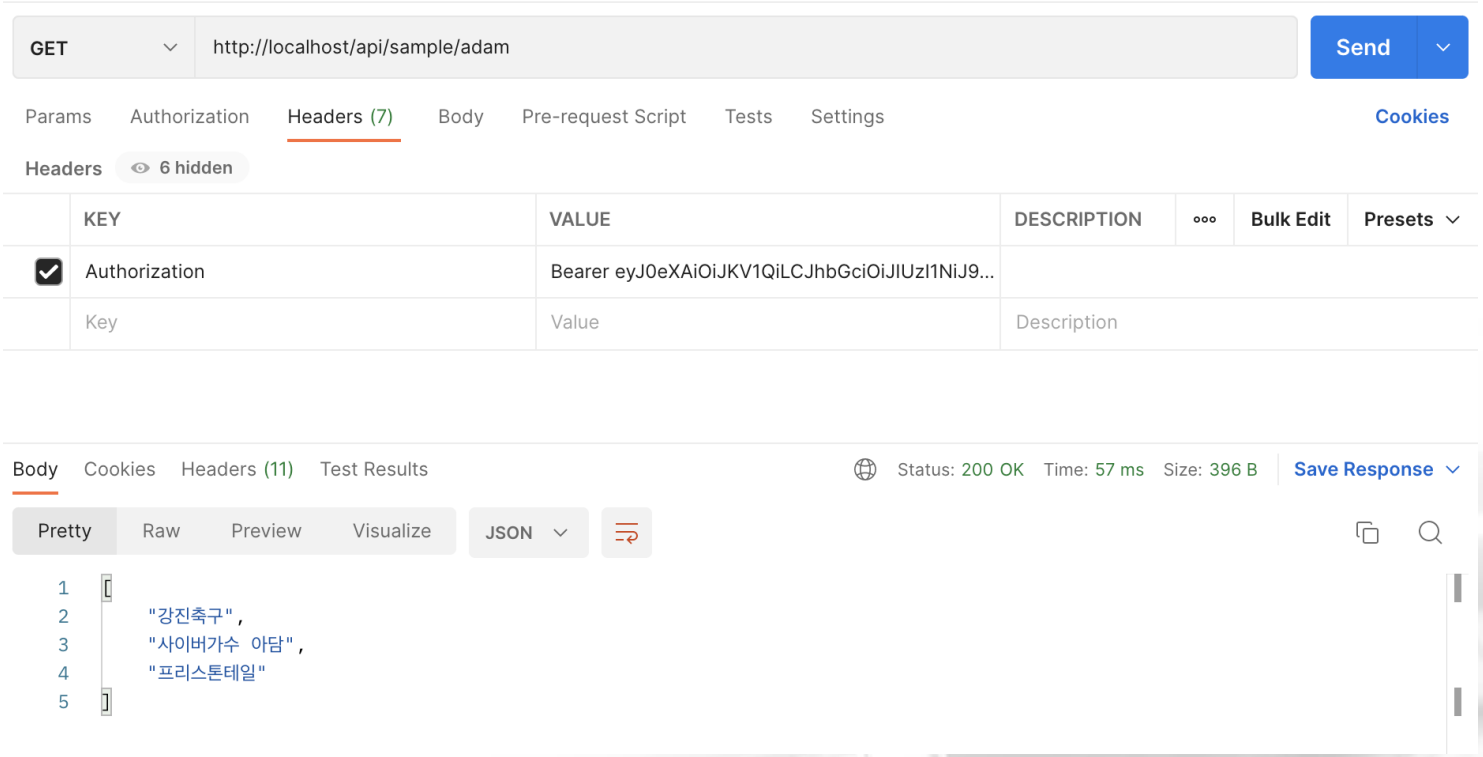
- Access Token 검증 필터
  - ✓ apiLogin.html 파일의 스크립트 수정해서 Token을 확인한 후 실행

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  document.getElementById("btn1").addEventListener("click",()=> {
    const data = {mid: "apiuser10", mpw: "1111"}
    axios.post("http://localhost/generateToken", data)
          .then(res => {
            console.log(res.data.accessToken)
          });
  });
</script>
```

JWT 인증

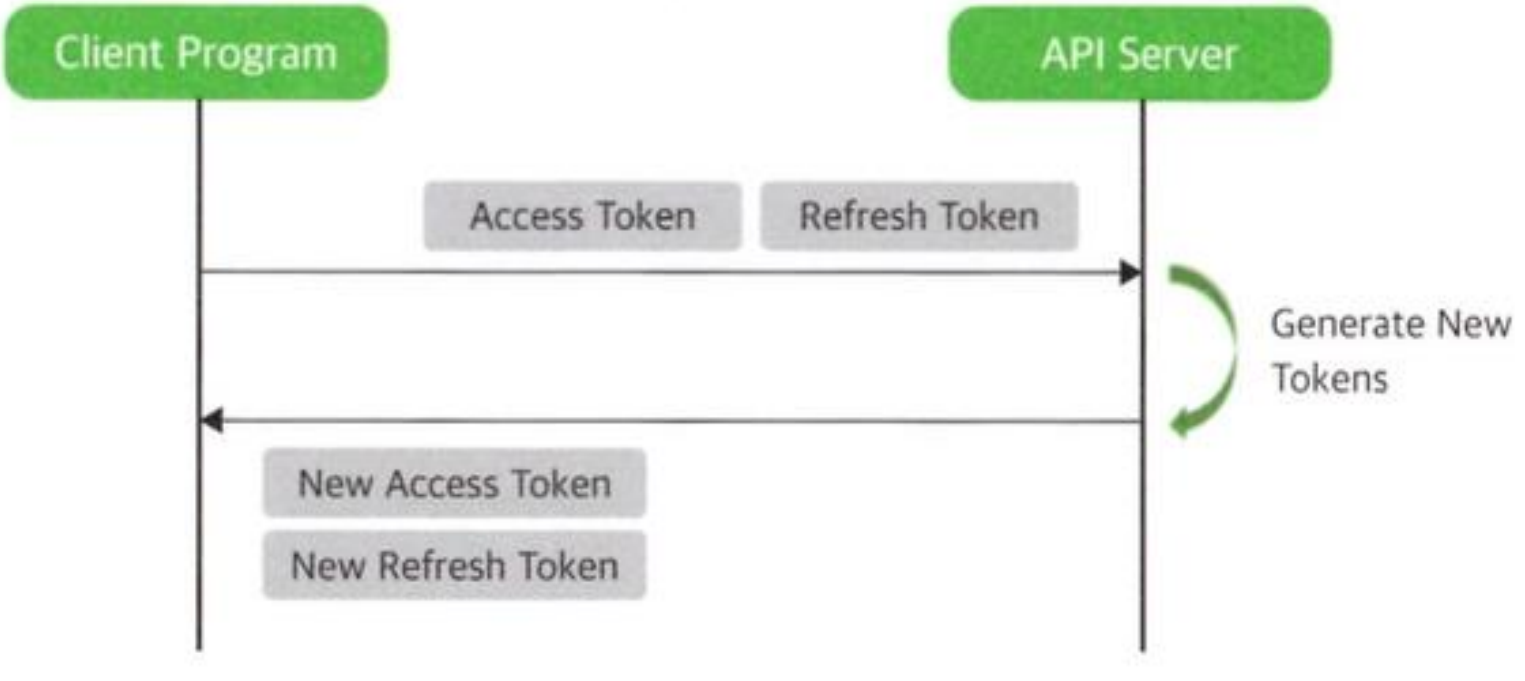
❖ Access Token 검증 필터

✓ 실행 – 토큰 정보를 복사한 후 앞에 Baerer: 을 추가하고 확인



JWT 인증

- 만료된 토큰이 전송되는 경우
  - ✓ 만료된 토큰이 전송되는 경우에 사용자는 다시 서버에 Access Token을 갱신해 달라고 요 구
  - ✓ /refreshToken이라는 경로를 이용해서 사용자가 다시 현재의 Access Token과 Refresh Token을 전송해 주면 이를 처리하도록 구성



JWT 인증

- 만료된 토큰이 전송되는 경우
  - ✓ /refreshToken에는 주어진 토큰이 다음과 같은 검증 과정으로 동작하도록 작성
    - Access Token이 존재하는지 확인
    - Refresh Token의 만료 여부 확인
    - Refresh Token의 만료 기간이 지났다면 다시 인증을 통해서 토큰들을 발급받아야 함을 전달
  - ✓ Refresh Token을 이용하는 과정에서는 발생하는 상황
    - Refresh Token의 만료 기간이 충분히 남아있으므로 AccessToken만 새로 만들어지는 경우
    - Refresh Token 자체도 만료 기간이 얼마 안 남아서 Access Token과 Refresh Token 모두 새로 만들어야 하는 경우

JWT 인증

- Refresh Token 검증 필터
  - ✓ Security.filter.RefreshTokenFilter 클래스 생성 - /refreshToken 경로와 JWTUtil 인스턴스를 주입받아서 해당 경로가 아닌 경우에는 다음 순서의 필터가 실행되도록 작성

```
@Log4j2
@RequiredArgsConstructor
public class RefreshTokenFilter  extends OncePerRequestFilter {
    private final String refreshPath;
    private final JWTUtil jwtUtil;
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
     response, FilterChain filterChain) throws ServletException, IOException {

        String path = request.getRequestURI();

        if (!path.equals(refreshPath)) {
            log.info("skip refresh token filter.....");
            filterChain.doFilter(request, response);
            return;
        }
        log.info("Refresh Token Filter...run.....1");
    }
}
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFilter 필터 설정 – CustomSecurityFilter 클래스의 메서드 수정

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("-------------------configure-----------------------------");

        //AuthenticationManager설정
        AuthenticationManagerBuilder authenticationManagerBuilder =
                http.getSharedObject(AuthenticationManagerBuilder.class);
        authenticationManagerBuilder.userDetailsService(apiUserDetailsService)
                .passwordEncoder(passwordEncoder());
        // Get AuthenticationManager
        AuthenticationManager authenticationManager =
    authenticationManagerBuilder.build();

        //반드시 필요
        http.authenticationManager(authenticationManager);
        //APILoginFilter
        //스프링 Security에서 username 과 password를 처리하는
    UsernamePasswordAuthenticationFilter 의 앞쪽에서 동작하도록 설정
        APILoginFilter apiLoginFilter = new APILoginFilter("/generateToken");
        apiLoginFilter.setAuthenticationManager(authenticationManager);
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFilter 필터 설정 – CustomSecurityFilter 클래스의 메서드 수정

```
        APILoginSuccessHandler successHandler = new APILoginSuccessHandler(jwtUtil);
        apiLoginFilter.setAuthenticationSuccessHandler(successHandler);

        //APILoginFilter의 위치 조정
        http.addFilterBefore(apiLoginFilter, UsernamePasswordAuthenticationFilter.class);

        http.addFilterBefore(tokenCheckFilter(jwtUtil),
    UsernamePasswordAuthenticationFilter.class);

        http.addFilterBefore(new RefreshTokenFilter("/refreshToken", jwtUtil),
                TokenCheckFilter.class);

        http.csrf().disable();
        //세션을 사용하지 않음

    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        return http.build();
    }
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFilter 필터 동작 확인


```
2023-01-24T17:42:32.639+09:00 TRACE 2522 --- [p-nio-80-exec-1] o.s.security.web.FilterChainProxy        : Invoking WebAsyncManagerIntegrationFilter (2/13)
2023-01-24T17:42:32.641+09:00 TRACE 2522 --- [p-nio-80-exec-1] o.s.security.web.FilterChainProxy        : Invoking SecurityContextHolderFilter (3/13)
2023-01-24T17:42:32.642+09:00 TRACE 2522 --- [p-nio-80-exec-1] o.s.security.web.FilterChainProxy        : Invoking HeaderWriterFilter (4/13)
2023-01-24T17:42:32.643+09:00 TRACE 2522 --- [p-nio-80-exec-1] o.s.security.web.FilterChainProxy        : Invoking LogoutFilter (5/13)
2023-01-24T17:42:32.643+09:00 TRACE 2522 --- [p-nio-80-exec-1] o.s.s.w.a.logout.LogoutFilter            : Did not match request to Or [Ant [pattern='/logout', GE
2023-01-24T17:42:32.644+09:00 TRACE 2522 --- [p-nio-80-exec-1] o.s.security.web.FilterChainProxy        : Invoking RefreshTokenFilter (6/13)
2023-01-24T17:42:32.644+09:00  INFO 2522 --- [p-nio-80-exec-1] c.k.a.s.filter.RefreshTokenFilter        : Refresh Token Filter...run.............1
2023-01-24T17:42:32.645+09:00 TRACE 2522 --- [p-nio-80-exec-1] o.s.s.w.header.writers.HstsHeaderWriter  : Not injecting HSTS header since it did not match reques
```

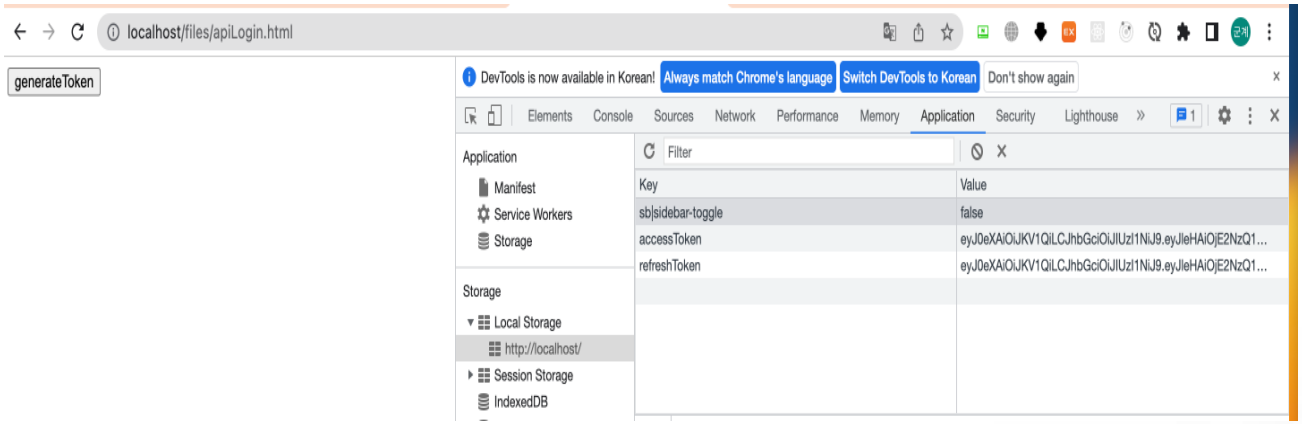
JWT 인증

- Refresh Token 검증 필터
  - ✓ apiLogin.html 파일의 스크립트를 수정해서 화면에서 테스트

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  document.querySelector(".btn1").addEventListener("click",()=> {
    const data = {mid:"apiuser10", mpw:"1111"}
    axios.post("http://localhost/generateToken", data).then(res => {
      const accessToken = res.data.accessToken
      const refreshToken = res.data.refreshToken
      localStorage.setItem("accessToken", accessToken)
      localStorage.setItem("refreshToken", refreshToken)
    })
  }, false)
</script>
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ apiLogin.html 파일을 수정



JWT 인증

- Refresh Token 검증 필터
  - ✓ refreshTest.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

<h1>ACCESS TOKEN</h1>
<h3 class="accessOld"></h3>
<h3 class="accessResult"></h3>
<hr/>

<h1>REFRESH TOKEN</h1>
<h3 class="refreshOld"></h3>
<h3 class="refreshResult"></h3>

<button class="btn1">Refresh</button>
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ refreshTest.html 파일을 생성하고 작성

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>
  const oldAccessToken = localStorage.getItem("accessToken")
  const oldRefreshToken = localStorage.getItem("refreshToken")
  document.querySelector(".accessOld").innerHTML = oldAccessToken
  document.querySelector(".refreshOld").innerHTML = oldRefreshToken
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ refreshTest.html 파일을 생성하고 작성

```
document.querySelector(".btn1").addEventListener("click", () => {
  axios.post('http://localhost/refreshToken',
        {accessToken: oldAccessToken, refreshToken: oldRefreshToken})
        .then(res => {
          console.log(res.data)
          const newAccessToken = res.data.accessToken
          const newRefreshToken = res.data.refreshToken
          document.querySelector(".accessResult").innerHTML =
                oldAccessToken !== newAccessToken?newAccessToken:'OLD'
          document.querySelector(".refreshResult").innerHTML =
                oldRefreshToken !== newRefreshToken?newRefreshToken:'OLD'
        })
        .catch(error => {
                console.error(error)
              }
        )
  },false)
</script>
</body>
</html>
```

JWT 인증

- ❖ Refresh Token 검증 필터
  - ✓ RefreshTokenFilter 내부 구현
    - 전송된 JSON 데이터에서 accessToken과 refreshToken을 추출
    - accessToken을 검사해서 토큰이 없거나 잘못된 토큰인 경우 에러 메시지 전송
    - refreshToken을 검사해서 토큰이 없거나 잘못된 토큰 혹은 만료된 토큰인 경우 에러 메시지 전송
    - 새로운 accessToken 생성
    - 만료 기한이 얼마 남지 않은 경우 새로운 refreshToken 생성
    - accessToken 과 refreshToken 전송

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenException 클래스 생성

```
public class RefreshTokenException extends RuntimeException{

    private ErrorCase errorCase;

    public enum ErrorCase {
        NO_ACCESS, NO_REFRESH, OLD_REFRESH
    }

    public RefreshTokenException(ErrorCase errorCase){
        super(errorCase.name());
        this.errorCase = errorCase;
    }
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenException 클래스 생성

```
public void sendResponseError(HttpServletResponse response){

    response.setStatus(HttpStatus.UNAUTHORIZED.value());
    response.setContentType(MediaType.APPLICATION_JSON_VALUE);

    Gson gson = new Gson();

    String responseStr = gson.toJson(Map.of("msg", errorCase.name(), "time", new
 Date()));

    try {
        response.getWriter().println(responseStr);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
  }
}
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스 수정

```
@Log4j2
@RequiredArgsConstructor
public class RefreshTokenFilter  extends OncePerRequestFilter {

    private final String refreshPath;

    private final JWTUtil jwtUtil;

    private Map<String,String> parseRequestJSON(HttpServletRequest request) {

        //JSON 데이터를 분석해서 mid, mpw 전달 값을 Map으로 처리
        try(Reader reader = new InputStreamReader(request.getInputStream())){
            Gson gson = new Gson();
            return gson.fromJson(reader, Map.class);
        }catch(Exception e){
            log.error(e.getMessage());
        }
        return null;
    }
```

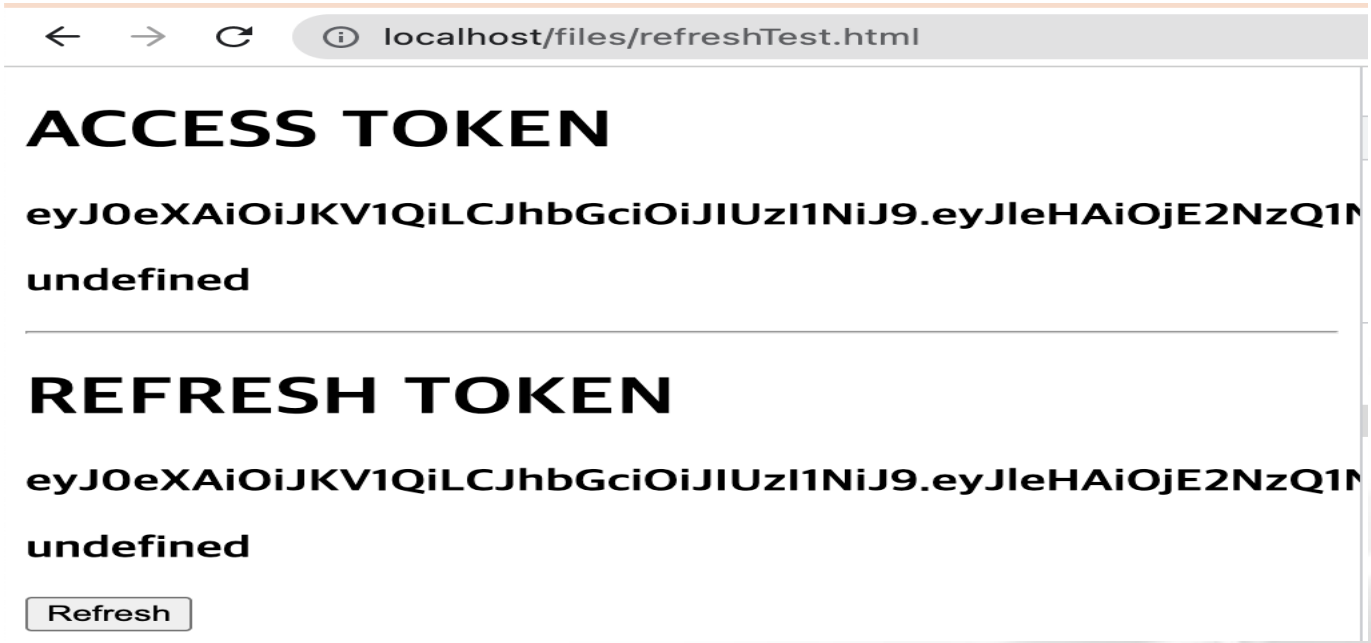
JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스 수정

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
 response, FilterChain filterChain) throws ServletException, IOException {
    String path = request.getRequestURI();

    if (!path.equals(refreshPath)) {
        log.info("skip refresh token filter.....");
        filterChain.doFilter(request, response);
        return;
    }

    log.info("Refresh Token Filter...run.....1");
    //전송된 JSON에서 accessToken과 refreshToken을 얻어온다.
    Map<String, String> tokens = parseRequestJSON(request);

    String accessToken = tokens.get("accessToken");
    String refreshToken = tokens.get("refreshToken");

    log.info("accessToken: " + accessToken);
    log.info("refreshToken: " + refreshToken);
  }
}
```

JWT 인증

- ❖ Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 확인




```
: accessToken: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NzQ1NTA3NzQsIm1pZCI6ImFwaXVzZXIxMCIsImlhdCI6MTY3NDU1MDE3NH0.Vyxim
: refreshToken: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2NzQ1NTYxNzQsIm1pZCI6ImFwaXVzZXIxMCIsImlhdCI6MTY3NDU1MDE3NH0.uE_S
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스에 AccessToken을 확인하는 메서드 추가

```
private void checkAccessToken(String accessToken)throws RefreshTokenException {

        try{
            jwtUtil.validateToken(accessToken);
        }catch (ExpiredJwtException expiredJwtException){
            log.info("Access Token has expired");
        }catch(Exception exception){
            throw new
    RefreshTokenException(RefreshTokenException.ErrorCase.NO_ACCESS);
        }
}
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스에 RefreshToken을 확인하는 메서드 추가

```
private Map<String, Object> checkRefreshToken(String refreshToken)throws
    RefreshTokenException{
        try {
            Map<String, Object> values = jwtUtil.validateToken(refreshToken);
            return values;
        }catch(ExpiredJwtException expiredJwtException){
            throw new
    RefreshTokenException(RefreshTokenException.ErrorCase.OLD_REFRESH);
        }catch(MalformedJwtException malformedJwtException){
            throw new
    RefreshTokenException(RefreshTokenException.ErrorCase.NO_REFRESH);
        }catch(Exception exception){
            new RefreshTokenException(RefreshTokenException.ErrorCase.NO_REFRESH);
        }
        return null;
}
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스의 doFilterInternal 메서드 수정 – AccessToken 과 RefreshToken 확인

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
 response, FilterChain filterChain) throws ServletException, IOException {
    String path = request.getRequestURI();

    if (!path.equals(refreshPath)) {
        log.info("skip refresh token filter.....");
        filterChain.doFilter(request, response);
        return;
    }

    log.info("Refresh Token Filter...run.....1");
    //전송된 JSON에서 accessToken과 refreshToken을 얻어온다.
    Map<String, String> tokens = parseRequestJSON(request);

    String accessToken = tokens.get("accessToken");
    String refreshToken = tokens.get("refreshToken");

    log.info("accessToken: " + accessToken);
    log.info("refreshToken: " + refreshToken);
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스의 doFilterInternal 메서드 수정 – AccessToken 과 RefreshToken 확인

```
        try{
            checkAccessToken(accessToken);
        }catch(RefreshTokenException refreshTokenException){
            refreshTokenException.sendResponseError(response);
            return;
        }

        Map<String, Object> refreshClaims = null;

        try {

            refreshClaims = checkRefreshToken(refreshToken);
            log.info(refreshClaims);

        }catch(RefreshTokenException refreshTokenException){
            refreshTokenException.sendResponseError(response);
            return;
        }
    }
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ 오래된 토큰을 전송하면 OLD_REFRESH 라는 메시지가 전송

JWT 인증

- 새로운 AccessToken 발생
  - ✓ Access Token은 무조건 새로 발행
  - ✓ Refresh Token은 만료일이 얼마 남지 않은 경우에 새로 발행

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스에 토큰을 전송하는 메서드 추가

```
private void sendTokens(String accessTokenValue, String refreshTokenValue,
    HttpServletResponse response) {
        response.setContentType(MediaType.APPLICATION_JSON_VALUE);

        Gson gson = new Gson();

        String jsonStr = gson.toJson(Map.of("accessToken", accessTokenValue,
                "refreshToken", refreshTokenValue));

        try {
            response.getWriter().println(jsonStr);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
}
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스의 doFilterInternal 메서드 수정

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
    response, FilterChain filterChain) throws ServletException, IOException {
    String path = request.getRequestURI();

    if (!path.equals(refreshPath)) {
        log.info("skip refresh token filter.....");
        filterChain.doFilter(request, response);
        return;
    }

    log.info("Refresh Token Filter...run.....1");
    //전송된 JSON에서 accessToken과 refreshToken을 얻어온다.
    Map<String, String> tokens = parseRequestJSON(request);

    String accessToken = tokens.get("accessToken");
    String refreshToken = tokens.get("refreshToken");

    log.info("accessToken: " + accessToken);
    log.info("refreshToken: " + refreshToken);
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스의 doFilterInternal 메서드 수정

```
try{
    checkAccessToken(accessToken);
}catch(RefreshTokenException refreshTokenException){
    refreshTokenException.sendResponseError(response);
    return;
}

Map<String, Object> refreshClaims = null;

try {

    refreshClaims = checkRefreshToken(refreshToken);
    log.info(refreshClaims);

}catch(RefreshTokenException refreshTokenException){
    refreshTokenException.sendResponseError(response);
    return;
}
```

JWT 인증

- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스의 doFilterInternal 메서드 수정

```
//Refresh Token의 유효시간이 얼마 남지 않은 경우
Integer exp = (Integer)refreshClaims.get("exp");

Date expTime = new Date(Instant.ofEpochMilli(exp).toEpochMilli() * 1000);

Date current = new Date(System.currentTimeMillis());

//만료 시간과 현재 시간의 간격 계산
//만일 3일 미만인 경우에는 Refresh Token도 다시 생성
long gapTime = (expTime.getTime() - current.getTime());

log.info("-------------------------------------------");
log.info("current: " + current);
log.info("expTime: " + expTime);
log.info("gap: " + gapTime );

String mid = (String)refreshClaims.get("mid");

//이상태까지 오면 무조건 AccessToken은 새로 생성
String accessTokenValue = jwtUtil.generateToken(Map.of("mid", mid), 1);
```

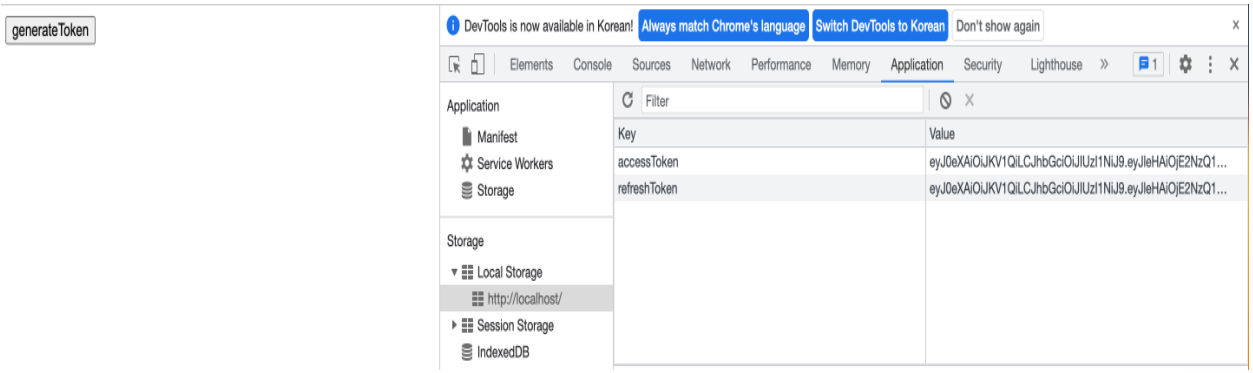
JWT 인증

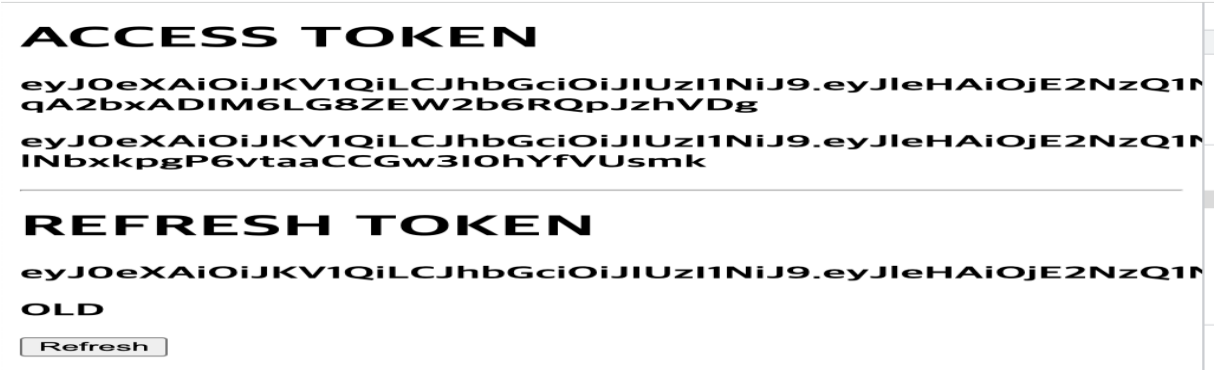
- Refresh Token 검증 필터
  - ✓ RefreshTokenFIlter 클래스 수정 – 새로운 AccessToken 발행 및 전송

```
        String refreshTokenValue = tokens.get("refreshToken");

        //RefrshToken이 3일도 안남았다면..
        if(gapTime < (1000 * 60  * 3  ) ){
            //if(gapTime < (1000 * 60 * 60 * 24 * 3  ) ){
            log.info("new Refresh Token required...  ");
            refreshTokenValue = jwtUtil.generateToken(Map.of("mid", mid), 30);
        }

        log.info("Refresh Token result.....................");
        log.info("accessToken: " + accessTokenValue);
        log.info("refreshToken: " + refreshTokenValue);

        sendTokens(accessTokenValue, refreshTokenValue, response);
    }
```

JWT 인증

- ❖ Refresh Token 검증 필터
  - ✓ 확인



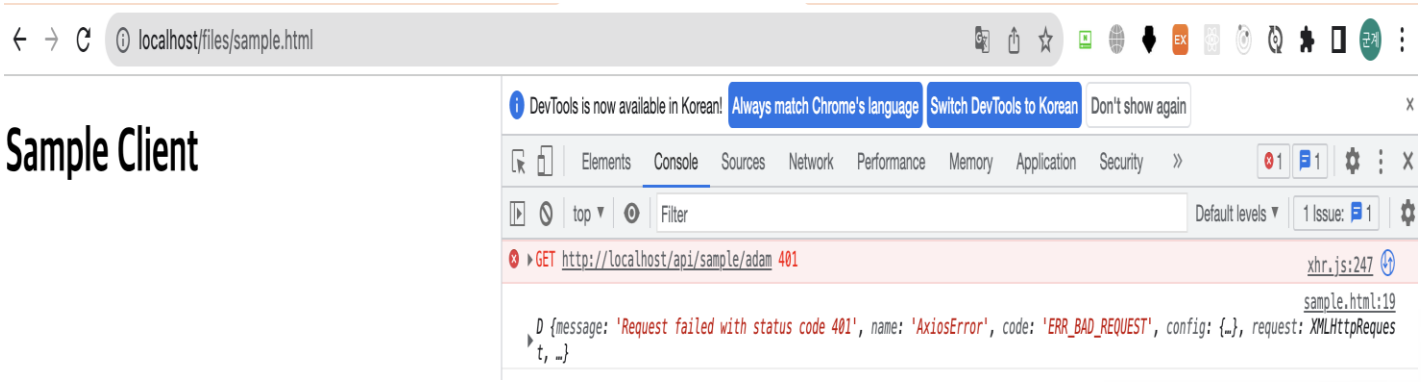


JWT 인증

- JWT 한계
  - ✓ JWT를 이용해서 자원을 보호하는 방식은 태생적으로 문자열이라는 한계가 존재
  - ✓ Refresh Token을 이용하는 부분만 생각해 보아도 외부의 공격자가 Refresh Token을 탈취한 상황이라면 얼마든지 새로운 Access Token을 생성할 수 있기 때문에 안전하지 않음
  - ✓ 이런 상황을 조금이라도 보완하기 위해서 Access Token과 Refresh Token을 DBMS에 보관하고 토큰을 갱신할 때 DBMS의 값과 비교하는 방법을 이용할 수 있는데 이 경우 정상적인 사용자가 Refresh Token을 갱신하게 되면 공격자가 탈취한 Refresh Token이 쓸모없게 됨
  - ✓ 이런 방식과 유사하게 새로운 Access Token이 발행될 때 아예 Refresh Token도 새로 발급하고 데이터베이스에 이를 저장하는 방법도 있는데 이렇게 되면 탈취된 Refresh Token의 유효 기간이 짧아지기 때문에 조금 더 안전해지긴 하지만 Access Token과 Refresh Token의 유효 시간이 같아지게 되므로 원래의 취지와는 조금 다른 형태가 됨
  - ✓ JWT를 안전하게 하기 위한 대부분의 방법들이 근본적으로 공격자가 Access Token과 Refresh Token을 탈취한 경우 최소한 1회 이상은 작업이 가능하다는 점에서 모든 보완책이 완벽할 수는 없음(인터넷에서 공격자가 사용자의 아이디와 패스워드를 탈취하는 상황과 큰 차이가 없음)

JWT 인증

- 브라우저에서 JWT 확인
  - ✓ 브라우저에서 sample.html 파일을 호출하면 에러 발생



JWT 인증

- ❖ 브라우저에서 JWT 확인
  - ✓ JWT를 사용하는 시나리오
    - /generateToken 을 호출해서 서버에서 발행한 Access Token과 Refresh Token을 받는 단계로 브라우저는 받은 토큰들을 저장해 두고 필요할 때마다 토큰들을 찾아서 사용하도록 구성
    - 브라우저에서 /api/sample/adam 를 호출할 때 가지고 있는 Access Token을 같이 전달했을 때 정상적인 결과가 나오는지를 확인
    - Access Token의 유효 기간이 만료되는 상황에 대한 처리fh Access Token의 유효 기간이 만료되면 서버에서는 에러 메시지를 전송하게 되는데 이 메시지를 판단해서 브라우저는 Refresh Token으로 다시 새로운 Access Token을 받고 원래 의도했던 작업을 수행(이 과정은 사용자가 참여하지 않고 자동으로 처리되어야 하기 때문에 slient refreshing 이라고 하기도 함)
    - Refresh Token 마저도 만료된 상황에 대한 처리로 Refresh Token이 만료되면 새로운 Access Token을 발행할 수 없기 때문에 사용자에게 1단계부터 다시 시작해야 함을 알려주어야 함

JWT 인증

- ❖ 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일을 생성하고 확인 – Local Storage에 보관된 Access Token을 이용하도록 하고 토큰이 없다면 경고창을 이용해서 출력

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>

<div class="result">

</div>

<button class="btn1">CALL SERVER</button>
```

JWT 인증

- ❖ 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일을 생성하고 확인 – Local Storage에 보관된 Access Token을 이용하도록 하고 토큰이 없다면 경고창을 이용해서 출력

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script>

  const callServer = async() => {
    console.log("call server 1...")

    const accessToken = localStorage.getItem("accessToken")

    if(!accessToken) {
      throw 'Cannot Find Access Token'
    }
  }
  const resultDiv = document.querySelector(".result")
```

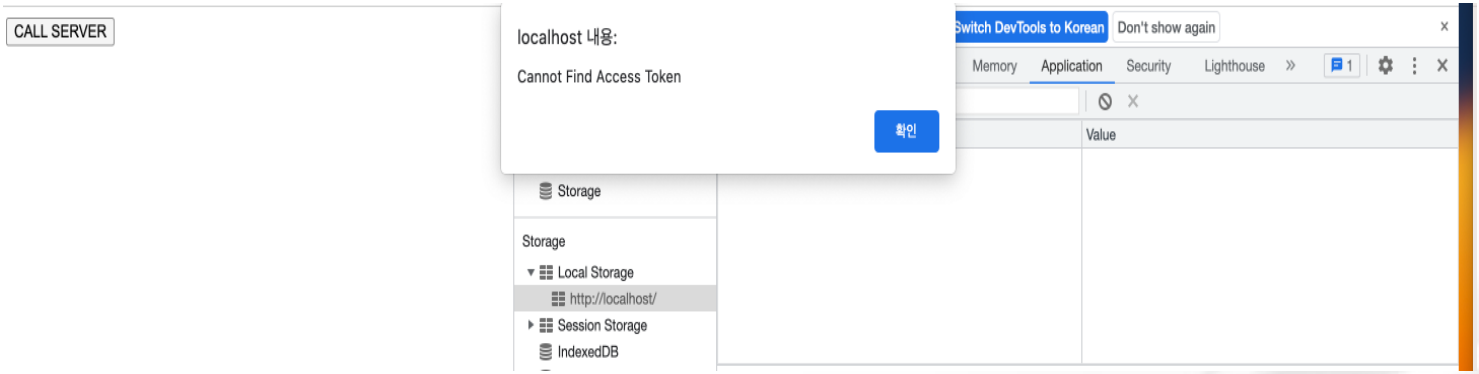
JWT 인증

- ❖ 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일을 생성하고 확인 – Local Storage에 보관된 Access Token을 이용하도록 하고 토큰이 없다면 경고창을 이용해서 출력

```
document.querySelector(".btn1").addEventListener("click", () => {

  callServer().then(result => {
    console.log(result)
  }).catch(error => {
    alert(error)
  })
},false)

</script>

</body>
</html>
```

JWT 인증

- ❖ 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일을 생성하고 확인 – Local Storage에 보관된 Access Token을 이용하도록 하고 토큰이 없다면 경고창을 이용해서 출력



JWT 인증

❖ 브라우저에서 JWT 확인

✓ Access Token을 이용한 접근

- sendJWT.html 파일의 스크립트 수정 – Local Storage에 보관된 Access Token이 존재하면 데이터를 호출하고 토큰이 없다면 경고창을 이용해서 출력

```
<script>
  const callServer = async() => {
    console.log("call server 1...")

    const accessToken = localStorage.getItem("accessToken")

    if(!accessToken) {
      throw 'Cannot Find Access Token'
    }

    const authHeader = {"Authorization": `Bearer ${accessToken}`}

    const res = await axios.get("http://localhost/api/sample/adam",{headers:
     authHeader})
    return res.data
  }
```

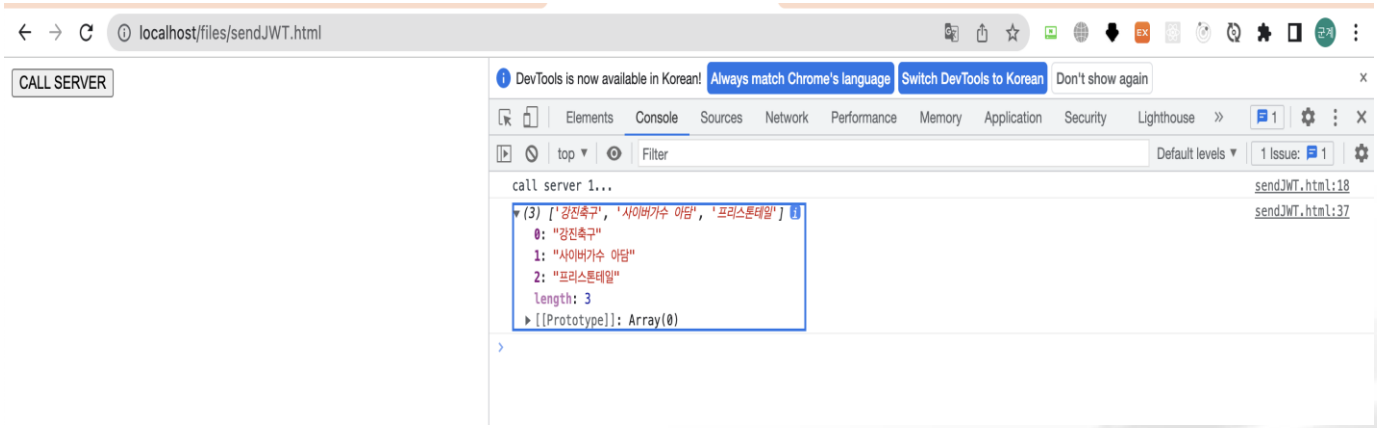
JWT 인증

- ❖ 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일의 스크립트 수정 – Local Storage에 보관된 Access Token이 존재하면 데이터를 호출하고 토큰이 없다면 경고창을 이용해서 출력

```
const resultDiv = document.querySelector(".result")

document.querySelector(".btn1").addEventListener("click", () => {

  callServer().then(result => {
    console.log(result)
  }).catch(error => {
    alert(error)
  })
},false)

</script>
```

JWT 인증

- ❖ 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일의 스크립트 수정 – Local Storage에 보관된 Access Token이 존재하면 데이터를 호출하고 토큰이 없다면 경고창을 이용해서 출력



JWT 인증

- 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일의 스크립트 수정 – ExpiredToken 메시지가 발생하는 경우는 반드시 한번은 /refreshToken을 호출하는 함수를 추가하고 예외 발생 시에 호출하도록 수정

```
<script>

  const callServer = async() => {
    console.log("call server 1...")

    const accessToken = localStorage.getItem("accessToken")

    if(!accessToken) {
      throw 'Cannot Find Access Token'
    }
```

JWT 인증

- ❖ 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일의 스크립트 수정 – ExpiredToken 메시지가 발생하는 경우는 반드시 한번은 /refreshToken을 호출하는 함수를 추가하고 예외 발생 시에 호출하도록 수정

```
const authHeader = {"Authorization": `Bearer ${accessToken}`}
try {
  const res = await axios.get("http://localhost/api/sample/adam",
          {headers: authHeader})
  return res.data
}catch(err) {
  if(err.response.data.msg === 'Expired Token'){
    console.log("Refresh Your Token")
    try{
      await callRefresh()
      console.log("new tokens....saved..")
      return callServer()
    }catch(refreshErr){
      throw refreshErr.response.data.msg
    }
  }//end if
}
}
```

JWT 인증

- ❖ 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일의 스크립트 수정 – ExpiredToken 메시지가 발생하는 경우는 반드시 한번은 /refreshToken을 호출하는 함수를 추가하고 예외 발생 시에 호출하도록 수정

```
const callRefresh = async () => {

  const accessToken = localStorage.getItem("accessToken")
  const refreshToken = localStorage.getItem("refreshToken")

  const tokens = {accessToken, refreshToken}
  const res = await axios.post("http://localhost/refreshToken", tokens)
  localStorage.setItem("accessToken", res.data.accessToken)
  localStorage.setItem("refreshToken", res.data.refreshToken)
}
```

JWT 인증

- 브라우저에서 JWT 확인
  - ✓ Access Token을 이용한 접근
    - sendJWT.html 파일의 스크립트 수정 – ExpiredToken 메시지가 발생하는 경우는 반드시 한번은 /refreshToken을 호출하는 함수를 추가하고 예외 발생 시에 호출하도록 수정

```
const resultDiv = document.querySelector(".result")

document.querySelector(".btn1").addEventListener("click", () => {

  callServer().then(result => {
    console.log(result)
  }).catch(error => {
    alert(error)
  })
},false)

</script>
```

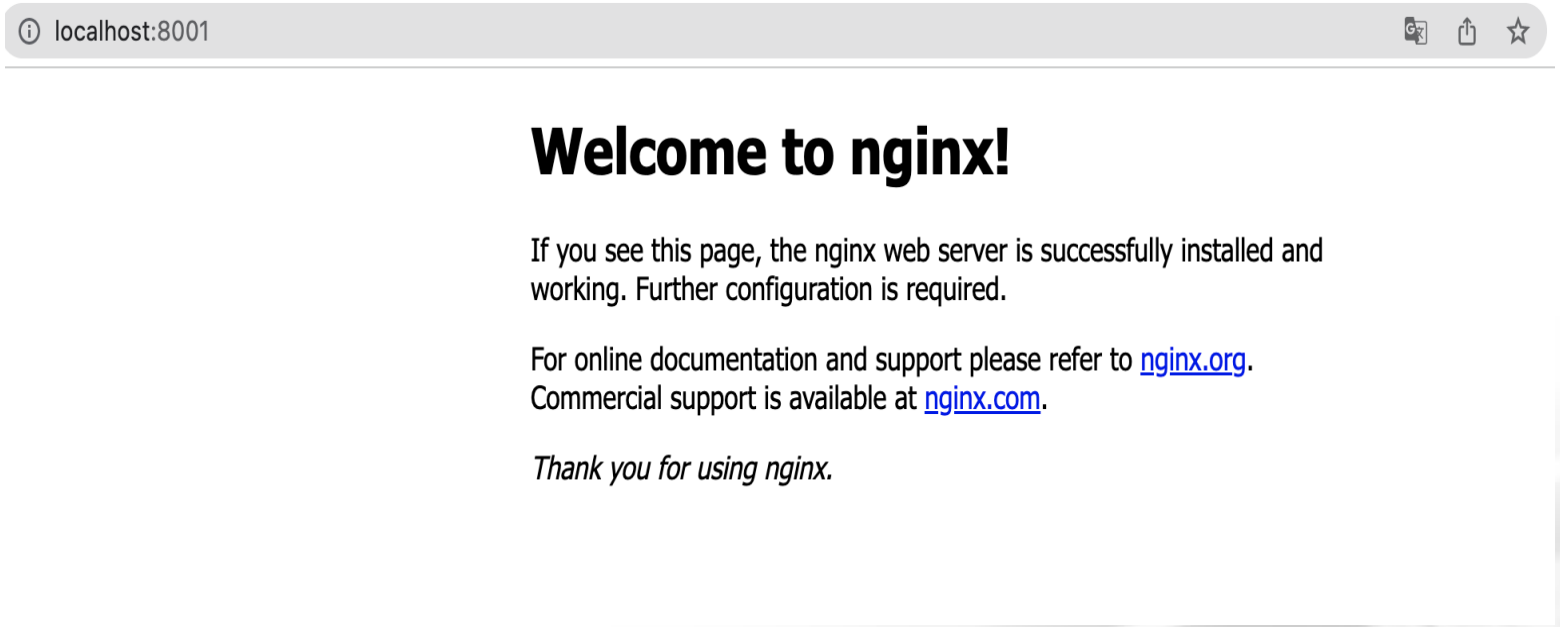
CORS 설정

- CORS
  - ✓ 별도의 WebServer 구동을 위해서 Docker에서 Nginx 컨테이너 실행

    docker run --name webserver -d -p 8001:80 nginx

CORS 설정

- CORS
  - ✓ 브라우저에서 구동 확인 - localhost:8001

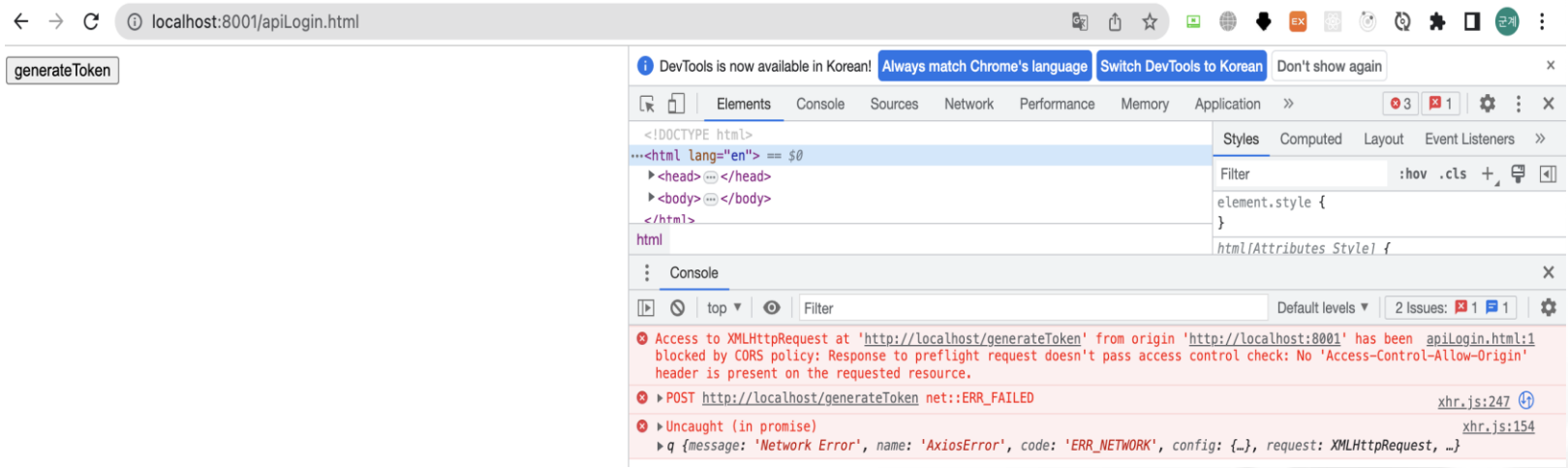


CORS 설정

- CORS
  - ✓ apiLogin.html 파일을 nginx 에 복사: docker cp apiLogin.html webserver:/usr/share/nginx/html/apiLogin.html

CORS 설정

- CORS
  - ✓ 브라우저에서 구동 확인 - localhost:8001/apiLogin.html



CORS 설정

- CORS
  - ✓ APIServer 프로젝트의 CustomSecurityConfig 클래스에 메서드를 추가

```
@Bean
public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOriginPatterns(Arrays.asList("*"));
        configuration.setAllowedMethods(Arrays.asList("HEAD", "GET", "POST", "PUT",
    "DELETE"));
        configuration.setAllowedHeaders(Arrays.asList("Authorization", "Cache-Control",
    "Content-Type"));
        configuration.setAllowCredentials(true);
        UrlBasedCorsConfigurationSource source = new
    UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
}
```

CORS 설정

- CORS
  - ✓ APIServer 프로젝트의 CustomSecurityConfig 클래스의 filterChain 메서드를 수정

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        log.info("-------------------configure-----------------------------");
        //AuthenticationManager설정
        AuthenticationManagerBuilder authenticationManagerBuilder =
                http.getSharedObject(AuthenticationManagerBuilder.class);
        authenticationManagerBuilder.userDetailsService(apiUserDetailsService)
                .passwordEncoder(passwordEncoder());
        // Get AuthenticationManager
        AuthenticationManager authenticationManager =
    authenticationManagerBuilder.build();

        //반드시 필요
        http.authenticationManager(authenticationManager);

        //APILoginFilter
        //스프링 Security에서 username 과 password를 처리하는
    UsernamePasswordAuthenticationFilter 의 앞쪽에서 동작하도록 설정
        APILoginFilter apiLoginFilter = new APILoginFilter("/generateToken");
        apiLoginFilter.setAuthenticationManager(authenticationManager);
```

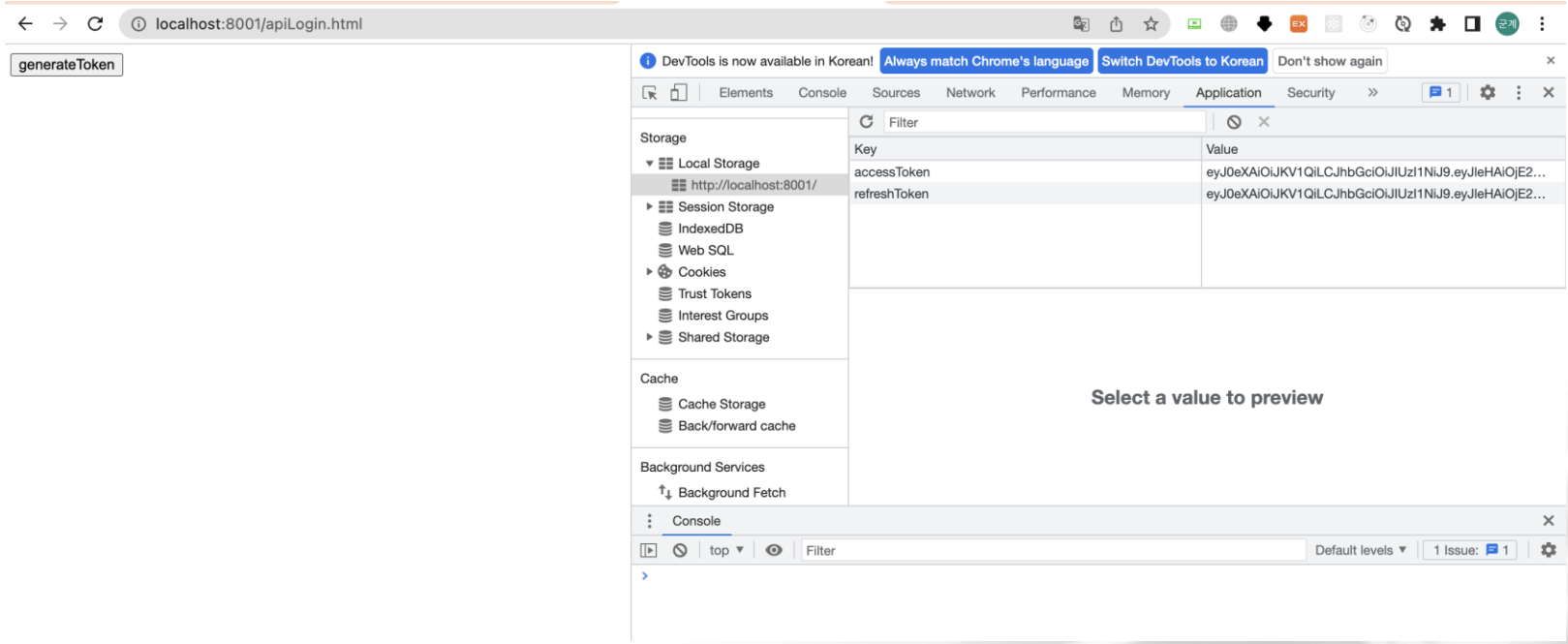
CORS 설정

- CORS
  - ✓ APIServer 프로젝트의 CustomSecurityConfig 클래스의 filterChain 메서드를 수정

```
        APILoginSuccessHandler successHandler = new APILoginSuccessHandler(jwtUtil);
        apiLoginFilter.setAuthenticationSuccessHandler(successHandler);
        //APILoginFilter의 위치 조정
        http.addFilterBefore(apiLoginFilter, UsernamePasswordAuthenticationFilter.class);

        http.addFilterBefore(tokenCheckFilter(jwtUtil),
    UsernamePasswordAuthenticationFilter.class);

        http.addFilterBefore(new RefreshTokenFilter("/refreshToken", jwtUtil),
    TokenCheckFilter.class);

        http.csrf().disable();
        //세션을 사용하지 않음
       http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        http.cors(httpSecurityCorsConfigurer -> {
            httpSecurityCorsConfigurer.configurationSource(corsConfigurationSource());
        });
        return http.build();
    }
```

CORS 설정

- CORS
  - ✓ 브라우저에서 구동 확인 - localhost:8001/apiLogin.html



ToDo API Server

❖ 요청 처리

경로	메소드	파라미터	설명
/api/todo/	POST	JSON	신규 Todo 입력
/api/todo/list	GET	size,page,from,to,keyword	PageResponseDTO를 JSON으로 만든 결과
/api/todo/{tno}	GET		특정 Todo 조회
/api/todo/{tno}	PUT	JSON	특정 Todo 수정
/api/todo/{tno}	DELETE		특정 Todo 삭제

ToDo API Server

- Model 계층
  - ✓ build.gradle 에 querydsl 사용을 위한 의존성 설정

```
// == 스프링 부트 3.0 이상 ==
implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'
annotationProcessor "com.querydsl:querydsl-
    apt:${dependencyManagement.importedProperties['querydsl.version']}:jakarta"
annotationProcessor "jakarta.annotation:jakarta.annotation-api"
annotationProcessor "jakarta.persistence:jakarta.persistence-api"
```

ToDo API Server

- Model 계층
  - ✓ Todo Entity – domain.Todo

```
@Entity
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Getter
@ToString
@Table(name = "tbl_todo_api")
```

ToDo API Server

- Model 계층
  - ✓ Todo Entity – domain.Todo

```
public class Todo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long tno;
    private String title;
    private LocalDate dueDate;
    private String writer;
    private boolean complete;

    public void changeComplete(boolean complete){
        this.complete = complete;
    }
    public void changeDueDate(LocalDate dueDate){
        this.dueDate = dueDate;
    }
    public void changeTitle(String title){
        this.title = title;
    }
}
```

ToDo API Server

- Model 계층
  - ✓ Todo Entity CRUD 작업을 위한 Repository 인터페이스 생성 – persistence.TodoRepository

```
public interface TodoRepository extends JpaRepository<Todo, Long> {
}
```

ToDo API Server

- Model 계층
  - ✓ Repository 테스트 – TodoRepositoryTests

```
@SpringBootTest
@Log4j2
public class TodoRepositoryTests {
    @Autowired
    private TodoRepository todoRepository;
    @Test
    public void testInsert() {
        IntStream.rangeClosed(1,100).forEach(i -> {
            Todo todo = Todo.builder()
                    .title("Todo..."+i)
                    .dueDate(LocalDate.of(2022, (i%12)+1, (i%30)+1 ))
                    .writer("user"+(i % 10))
                    .complete(false)
                    .build();

            todoRepository.save(todo);

        });
    }
}
```

ToDo API Server

- Service 계층
  - ✓ DTO 클래스 – dto.TodoDTO

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class TodoDTO {

    private Long tno;
    private String title;

    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd",
     timezone = "Asia/Seoul")
    private LocalDate dueDate;

    private String writer;
    private boolean complete;
}
```

ToDo API Server

- Service 계층
  - ✓ Service 인터페이스 – service.TodoService

```
public interface TodoService {
    Long register(TodoDTO todoDTO);
}
```

ToDo API Server

- Service 계층
  - ✓ Service 클래스 – service.TodoServiceImpl

```
@Service
@RequiredArgsConstructor
@Log4j2
public class TodoServiceImpl implements TodoService{

    private final TodoRepository todoRepository;
    private final ModelMapper modelMapper;

    @Override
    public Long register(TodoDTO todoDTO) {
        Todo todo = modelMapper.map(todoDTO, Todo.class);
        Long tno = todoRepository.save(todo).getTno();
        return tno;
    }
}
```

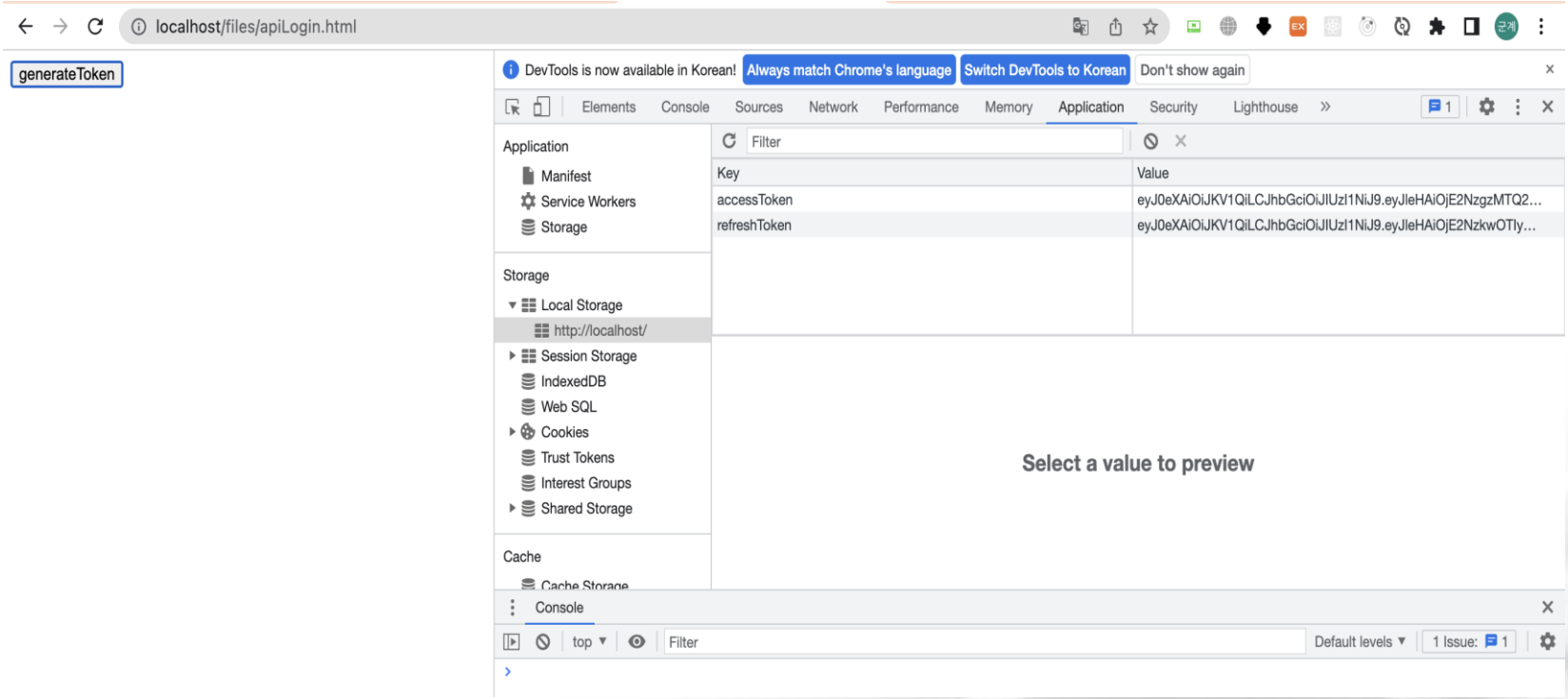
ToDo API Server

- Controller 계층
  - ✓ Controller 클래스 – controller.TodoController

```
@RestController
@RequestMapping("/api/todo")
@Log4j2
@RequiredArgsConstructor
public class TodoController {

    private final TodoService todoService;

    @PostMapping(value = "/", consumes = MediaType.APPLICATION_JSON_VALUE)
    public Map<String, Long> register(@RequestBody TodoDTO todoDTO){
        log.info(todoDTO);
        Long tno = todoService.register(todoDTO);
        return Map.of("tno", tno);
    }
}
```

ToDo API Server

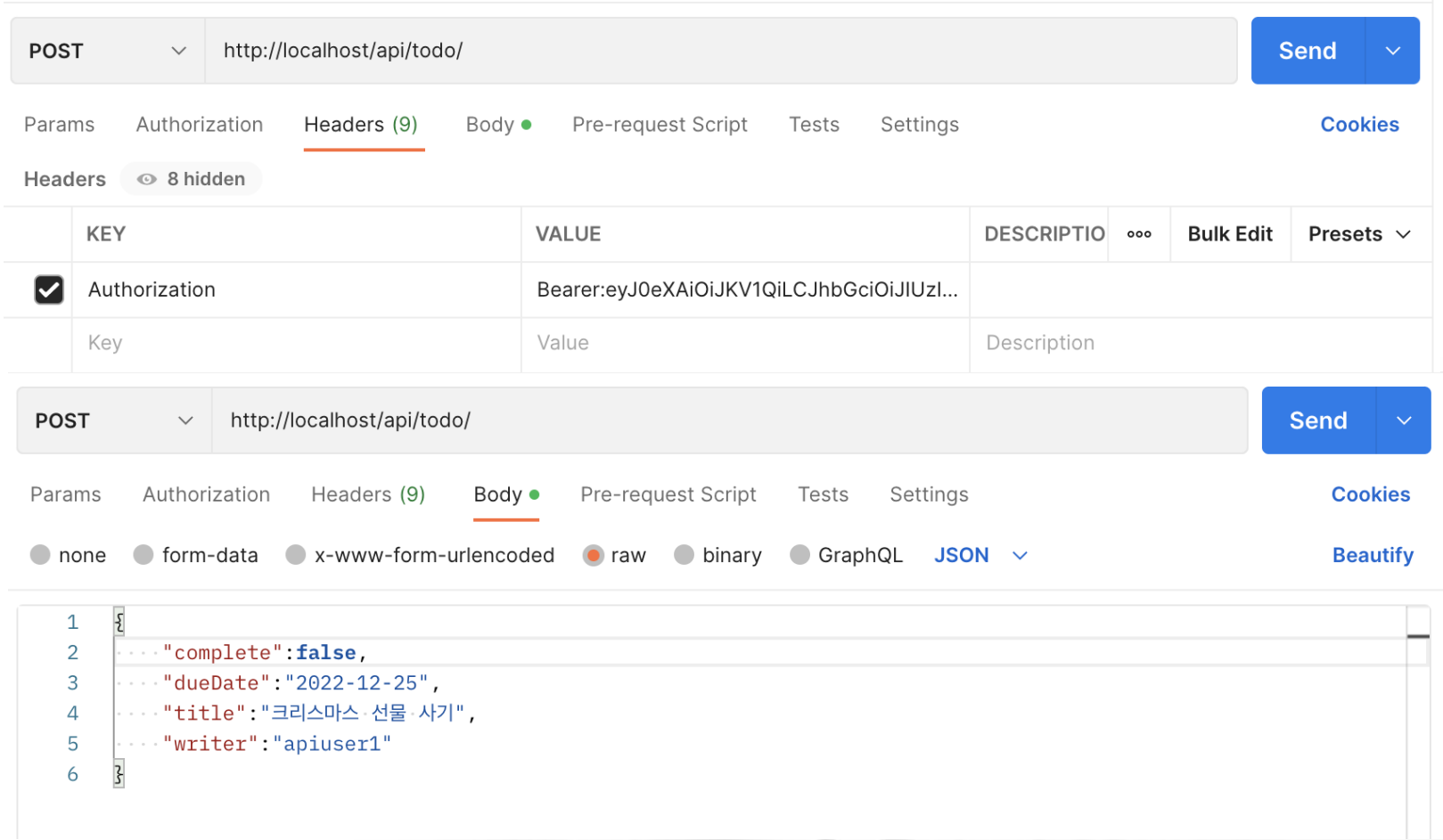
- 데이터 삽입 테스트
  - ✓ Access Token 생성



ToDo API Server

❖ 데이터 삽입 테스트

✓ 데이터 삽입



ToDo API Server

- 조회 와 목록 보기
  - ✓ TodoService 인터페이스에 조회 메서드 선언

    TodoDTO read(Long tno);

ToDo API Server

- 조회 와 목록 보기
  - ✓ TodoServiceImpl 클래스에 조회 메서드 구현

```
@Override
public TodoDTO read(Long tno) {
        Optional<Todo> result = todoRepository.findById(tno);
        Todo todo = result.orElseThrow();
        return modelMapper.map(todo, TodoDTO.class);
}
```

ToDo API Server

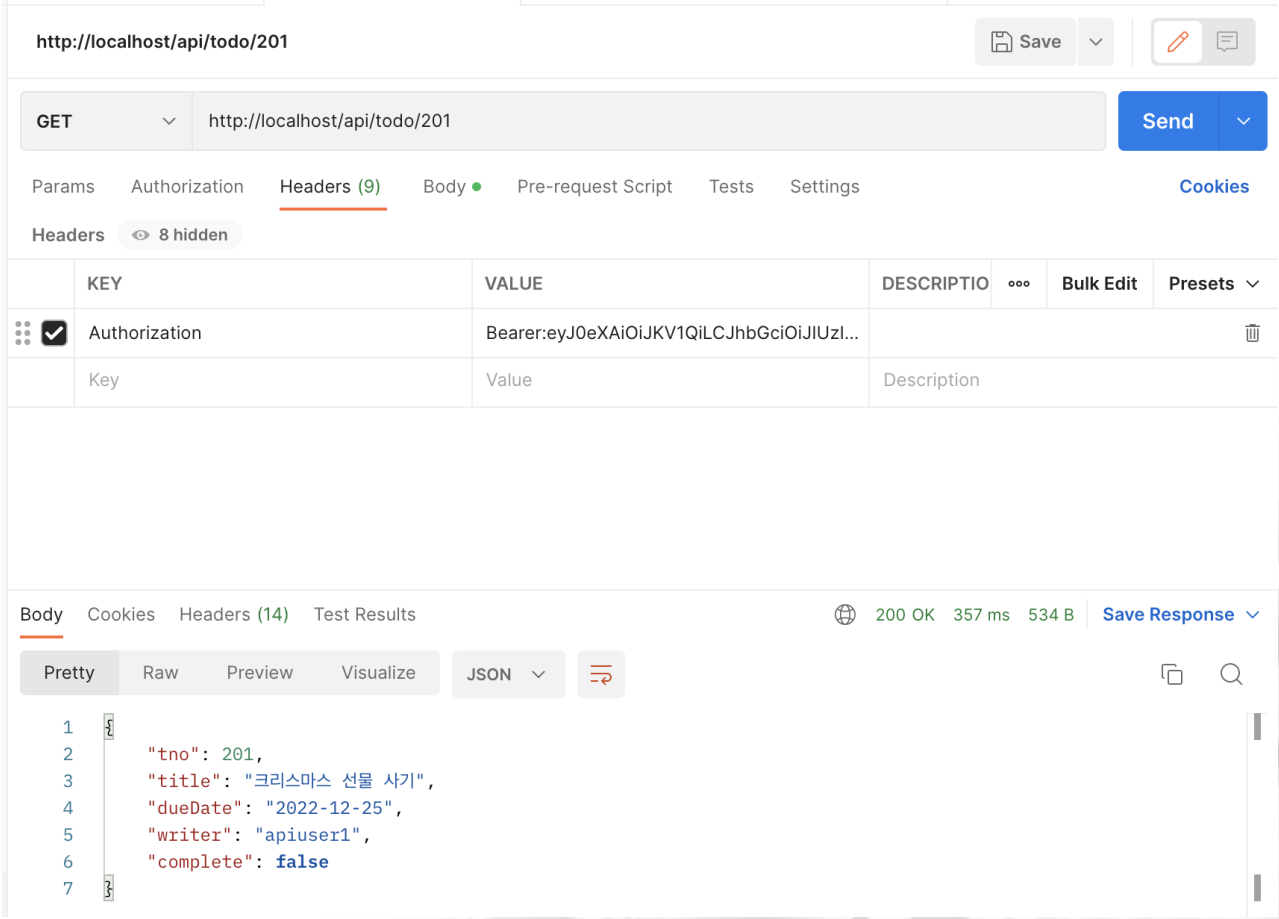
- 조회 와 목록 보기
  - ✓ TodoController 클래스에 조회 요청 처리 메서드 추가

```
@GetMapping("/{tno}")
public TodoDTO read(@PathVariable("tno") Long tno){
        log.info("read tno: " + tno);
        return todoService.read(tno);
}
```

ToDo API Server

❖ 조회 와 목록 보기

✓ 조회 테스트



ToDo API Server

- 조회 와 목록 보기
  - ✓ 목록 보기 요청을 위한 DTO 클래스 생성 – dto.PageRequestDTO

```
@Builder
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PageRequestDTO {
    @Builder.Default
    private int page = 1;
    @Builder.Default
    private int size = 10;
    private String type; // 검색의 종류 t,c, w, tc,tw, twc
    private String keyword;

    //추가된 내용들
    private LocalDate from;
    private LocalDate to;

    private Boolean completed;
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ 목록 보기 요청을 위한 DTO 클래스 생성 – dto.PageRequestDTO

```
public String[] getTypes(){
    if(type == null || type.isEmpty()){
        return null;
    }
    return type.split("");
}

public Pageable getPageable(String...props) {
    return PageRequest.of(this.page -1, this.size, Sort.by(props).descending());
}
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ 목록 보기 요청을 위한 DTO 클래스 생성 – dto.PageRequestDTO

```
private String link;
public String getLink() {
    if(link == null){
        StringBuilder builder = new StringBuilder();
        builder.append("page=" + this.page);
        builder.append("&size=" + this.size);
        if(type != null && type.length() > 0){
            builder.append("&type=" + type);
        }

        if(keyword != null){
            try {
                builder.append("&keyword=" + URLEncoder.encode(keyword,"UTF-8"));
            } catch (UnsupportedEncodingException e) {
            }
        }
        link = builder.toString();
    }
    return link;
}
}
```

ToDo API Server

❖ 조회 와 목록 보기

✓ 목록 보기 응답을 위한 DTO 클래스 생성 – dto.PageResponseDTO

```
@Getter
@ToString
public class PageResponseDTO<E> {

    private int page;
    private int size;
    private int total;

    //시작 페이지 번호
    private int start;
    //끝 페이지 번호
    private int end;

    //이전 페이지의 존재 여부
    private boolean prev;
    //다음 페이지의 존재 여부
    private boolean next;

    private List<E> dtoList;
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ 목록 보기 응답을 위한 DTO 클래스 생성 – dto.PageResponseDTO

```
@Builder(builderMethodName = "withAll")
public PageResponseDTO(PageRequestDTO pageRequestDTO, List<E> dtoList, int
 total){
    if(total <= 0){
        return;
    }
    this.page = pageRequestDTO.getPage();
    this.size = pageRequestDTO.getSize();
    this.total = total;
    this.dtoList = dtoList;
    this.end =   (int)(Math.ceil(this.page / 10.0 )) *  10;
    this.start = this.end - 9;
    int last =  (int)(Math.ceil((total/(double)size)));
    this.end =  end > last ? last: end;
    this.prev = this.start > 1;
    this.next =  total > this.end * this.size;
  }
}
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ Querydsl을 이용한 검색 조건 처리를 위한 인터페이스 생성 – persistence.search.TodoSearch

```
public interface TodoSearch {
    Page<TodoDTO> list(PageRequestDTO pageRequestDTO);
}
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ Querydsl을 이용한 검색 조건 처리를 위한 클래스 생성 – persistence.search.TodoSearchImpl

```
@Log4j2
public class TodoSearchImpl extends QuerydslRepositorySupport implements
    TodoSearch {

    public TodoSearchImpl() {
        super(Todo.class);
    }
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ Querydsl을 이용한 검색 조건 처리를 위한 클래스 생성 – persistence.search.TodoSearchImpl

```
@Override
public Page<TodoDTO> list(PageRequestDTO pageRequestDTO) {

    QTodo todo = QTodo.todo;

    JPQLQuery<Todo> query = from(todo);

    if(pageRequestDTO.getFrom() != null && pageRequestDTO.getTo() != null){

        BooleanBuilder fromToBuilder = new BooleanBuilder();
        fromToBuilder.and(todo.dueDate.goe(pageRequestDTO.getFrom()));
        fromToBuilder.and(todo.dueDate.loe(pageRequestDTO.getTo()));
        query.where(fromToBuilder);
    }

    if(pageRequestDTO.getCompleted() != null){
        query.where(todo.complete.eq(pageRequestDTO.getCompleted()));
    }
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ Querydsl을 이용한 검색 조건 처리를 위한 클래스 생성 – persistence.search.TodoSearchImpl

```
        if(pageRequestDTO.getKeyword() != null){
            query.where(todo.title.contains(pageRequestDTO.getKeyword()));
        }
        this.getQuerydsl().applyPagination(pageRequestDTO.getPageable("tno"), query);
        JPQLQuery<TodoDTO> dtoQuery = query.select(Projections.bean(TodoDTO.class,
                todo.tno,
                todo.title,
                todo.dueDate,
                todo.complete,
                todo.writer
        ));
        List<TodoDTO> list = dtoQuery.fetch();
        long count = dtoQuery.fetchCount();
        return new PageImpl<>(list, pageRequestDTO.getPageable("tno"), count);
    }
}
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ TodoRepository 인터페이스에 TodoSearch 인터페이스를 상속

```
public interface TodoRepository extends JpaRepository<Todo, Long>, TodoSearch {
}
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ 목록 보기 메서드 테스트

```
@Test
public void testSearch(){
        PageRequestDTO pageRequestDTO = PageRequestDTO.builder()
                .from(LocalDate.of(2022,10,01))
                .to(LocalDate.of(2022,12,31))
                .build();
        Page<TodoDTO> result = todoRepository.list(pageRequestDTO);
        result.forEach(todoDTO -> System.out.println(todoDTO));
}
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ TodoService 인터페이스에 목록 보기 메서드 선언

```
PageResponseDTO<TodoDTO> list(PageRequestDTO pageRequestDTO);
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ TodoServiceImpl 클래스에 목록 보기 메서드 구현

```
@Override
public PageResponseDTO<TodoDTO> list(PageRequestDTO pageRequestDTO) {

        Page<TodoDTO> result = todoRepository.list(pageRequestDTO);

        return PageResponseDTO.<TodoDTO>withAll()
                .pageRequestDTO(pageRequestDTO)
                .dtoList(result.toList())
                .total((int)result.getTotalElements())
                .build();
}
```

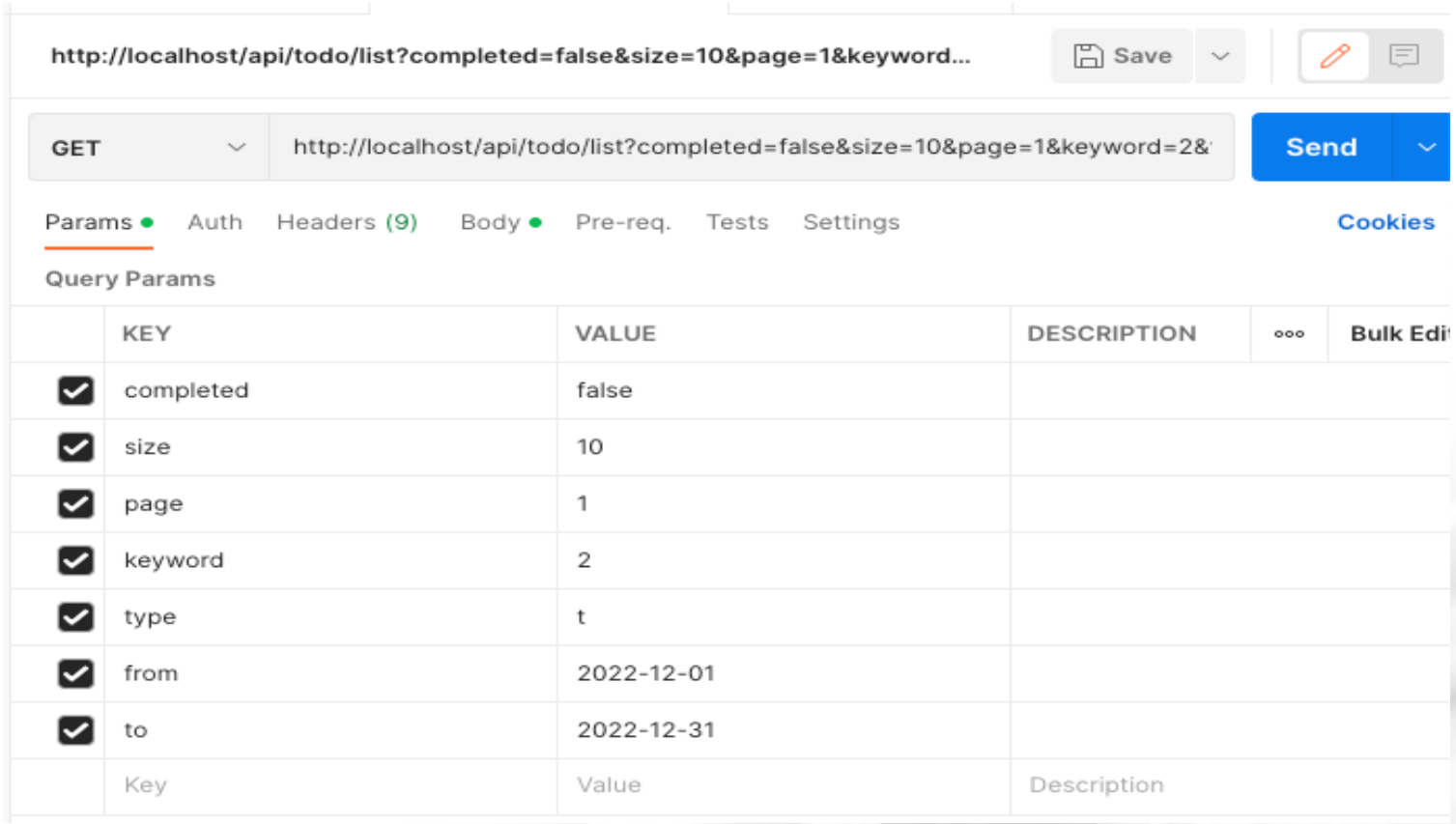
ToDo API Server

- 조회 와 목록 보기
  - ✓ TodoController 클래스에 목록 보기 요청 처리 메서드 구현

```
@GetMapping(value = "/list", produces = MediaType.APPLICATION_JSON_VALUE)
public PageResponseDTO<TodoDTO> list(PageRequestDTO pageRequestDTO){
        return todoService.list(pageRequestDTO);
}
```

ToDo API Server

- 조회 와 목록 보기
  - ✓ 목록 보기 요청 확인



ToDo API Server

❖ 수정 과 삭제

✓ TodoService 인터페이스에 수정 과 삭제 메서드 선언

```
void remove(Long tno);

void modify(TodoDTO todoDTO);
```

ToDo API Server

❖ 수정 과 삭제

✓ TodoServiceImpl 클래스에 수정 과 삭제 메서드 구현

```
@Override
public void remove(Long tno) {
        todoRepository.deleteById(tno);
}


@Override
public void modify(TodoDTO todoDTO) {
        Optional<Todo> result = todoRepository.findById(todoDTO.getTno());

        Todo todo = result.orElseThrow();

        todo.changeTitle(todoDTO.getTitle());
        todo.changeDueDate(todoDTO.getDueDate());
        todo.changeComplete(todoDTO.isComplete());

        todoRepository.save(todo);
}
```

ToDo API Server

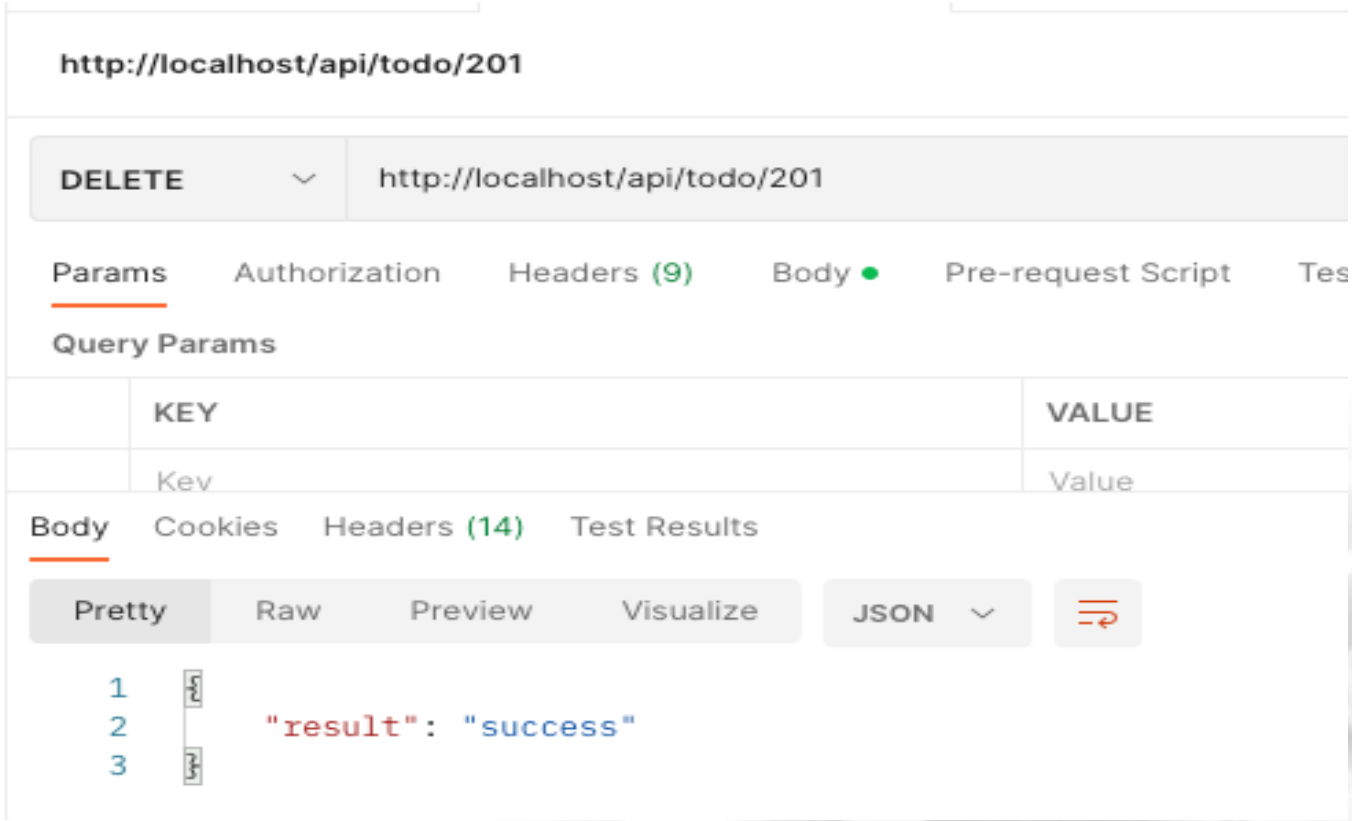
❖ 수정 과 삭제

✓ TodoController 클래스에 수정 과 삭제 요청 처리 메서드 구현

```
@DeleteMapping(value= "/{tno}")
public Map<String, String> delete(@PathVariable Long tno){

        todoService.remove(tno);
        return Map.of("result", "success");
}

@PutMapping(value="/{tno}", consumes = MediaType.APPLICATION_JSON_VALUE)
public Map<String, String> modify(@PathVariable("tno") Long tno,@RequestBody
    TodoDTO todoDTO ){

        //잘못된 tno가 발생하지 못하도록
        todoDTO.setTno(tno);

        todoService.modify(todoDTO);

        return Map.of("result", "success");
}
```

ToDo API Server

- 수정 과 삭제
  - ✓ 수정 과 삭제 테스트



ToDo API Server

❖ 수정 과 삭제

✓ 수정 과 삭제 테스트