YouTube
NAVER 카페
구멍가게 코딩단

코드로 배우는 리액트

7. 시큐리티와 API 서버

7장. 시큐리티와 API 서버

- API 서버와 리액트 어플리케이션 분리로 인해
  인증 방식은 쿠키나 세션 대신 JWT(JSON Web Token) 토큰 기반 인증 사용
- API 서버 로그인 시 JWT 토큰 생성.
- API 경로 호출 시 JWT 토큰 검사하여 접근 제한 설정.
- 보안 강화와 제한된 사용자만 API 서버 접근 가능하도록 처리

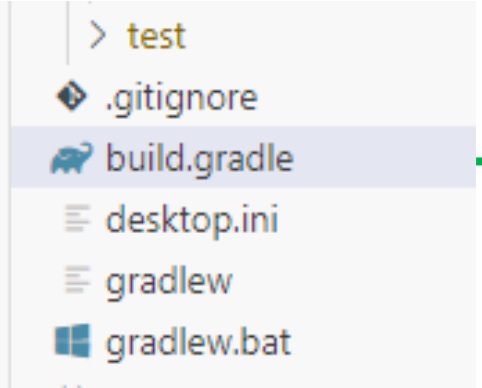
개발목표

1. API 서버를 위한 스프링 시큐리티의 설정
2. API 서버 호출과 JWT 문자열 생성
3. JWT 생성과 검증
4. Access Token과 Refresh Token을 이용한 토큰 갱신

Index

스프링 시큐리티 설정

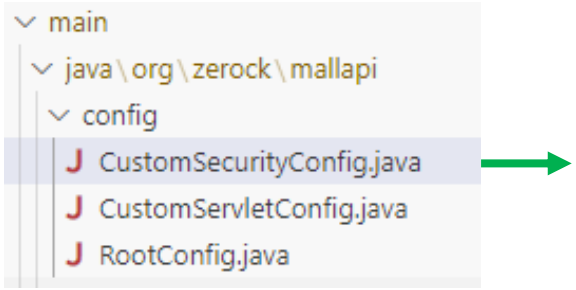
→ build.gradle파일에 스프링 시큐리티 관련 모듈 추가




```
dependencies {
  …생략

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

}
```

스프링 시큐리티 설정

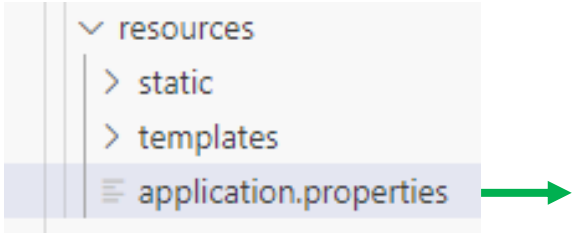
→ config 패키지에 CustomSecurityConfig.java파일 추가




```java
package org.zerock.mallapi.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;

@Configuration
@Log4j2
@RequiredArgsConstructor
public class CustomSecurityConfig {

  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    log.info("----------------security config-------------------");

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

스프링 시큐리티 설정

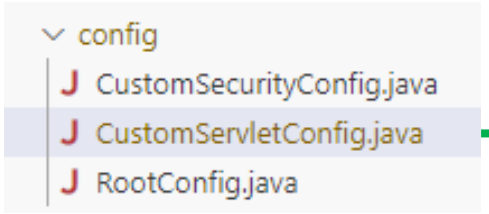
→ application.properties에서 로그 설정




```
… 생략

logging.level.org.springframework.security.web=trace

… 생략
```

API 서버를 위한 기본 설정

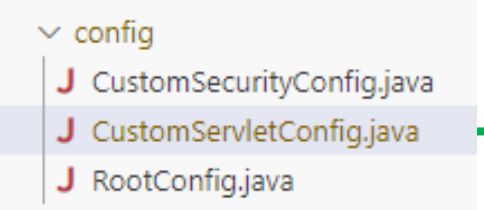
→ CORS, CSRF설정




```java
package org.zerock.mallapi.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.format.FormatterRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.zerock.mallapi.controller.formatter.LocalDateFormatter;

@Configuration
public class CustomServletConfig implements WebMvcConfigurer{

  @Override
  public void addFormatters(FormatterRegistry registry) {

    registry.addFormatter(new LocalDateFormatter());
  }

}
```

API 서버를 위한 기본 설정

→ CustomSecurityConfig.java 에서 CORS 설정




```
@Configuration
@Log4j2
@RequiredArgsConstructor
public class CustomSecurityConfig {

  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    log.info("---------------------security config---------------------------");

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

    return http.build();
  }
```

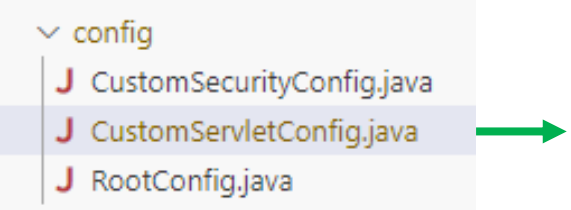
API 서버를 위한 기본 설정

→ CustomSecurityConfig.java 에서 CORS 설정

```
  @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;
  }
}
```

API 서버를 위한 기본 설정

→ CSRF설정




```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

  log.info("---------------security config---------------------");

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

  http.sessionManagement(sessionConfig ->
    sessionConfig.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

  http.csrf(config -> config.disable());

  return http.build();
}
```

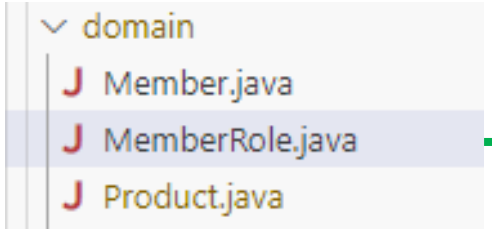
API 서버를 위한 기본 설정

→ PasswordEncoder설정


```java
@Bean
public PasswordEncoder passwordEncoder(){

  return new BCryptPasswordEncoder();

}
```

Member 엔티티 처리

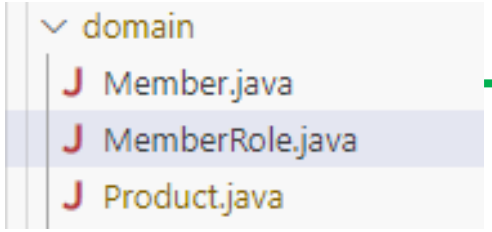
→ MemberRole.java




```java
package org.zerock.mallapi.domain;

public enum MemberRole {

    USER, MANAGER,ADMIN;

}
```

Member 엔티티 처리

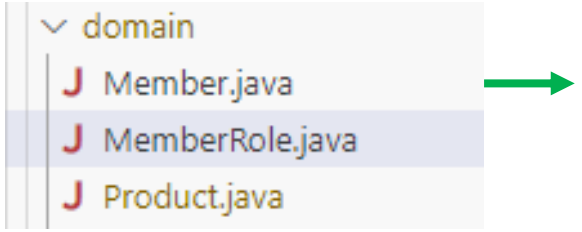
→ Member.java




```java
package org.zerock.mallapi.domain;

import jakarta.persistence.*;
import lombok.*;

import java.util.*;

@Entity
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
@ToString (exclude = "memberRoleList")
public class Member {
```

Member 엔티티 처리

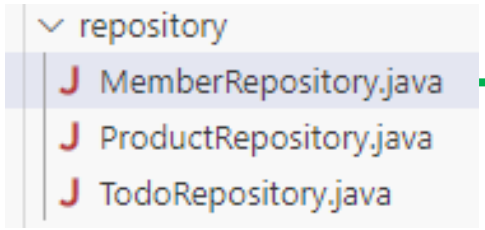
→ Member.java




```
    @Id
    private String email;
    private String pw;
    private String nickname;
    private boolean social;

    @ElementCollection(fetch = FetchType.LAZY)
    @Builder.Default
    private List<MemberRole> memberRoleList = new ArrayList<>();

    public void addRole(MemberRole memberRole){ memberRoleList.add(memberRole); }
    public void clearRole(){ memberRoleList.clear(); }
    public void changeNickname(String nickname) { this.nickname = nickname; }
    public void changePw(String pw){ this.pw = pw; }
    public void changeSocial(boolean social) { this.social = social; }

}
```

Member 엔티티 처리

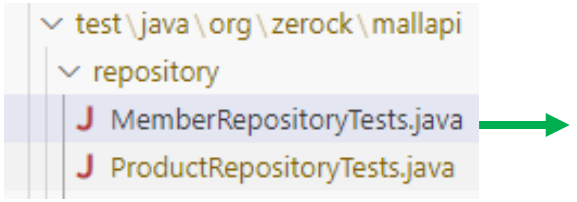
→ repository 패키지 내에 MemberRepository 인터페이스 추가




```java
package org.zerock.mallapi.repository;

import org.springframework.data.jpa.repository.*;
import org.springframework.data.repository.query.Param;
import org.zerock.mallapi.domain.Member;

public interface MemberRepository extends JpaRepository<Member, String> {

    @EntityGraph(attributePaths = {"memberRoleList"})
    @Query("select m from Member m where m.email = :email")
    Member getWithRoles(@Param("email") String email);

}
```

테스트 코드를 이용한 등록/조회 확인

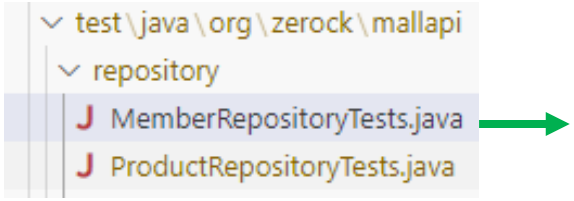
→ 테스트 폴더의 repository 패키지에 MemberRepositoryTests.java 파일을 추가




```java
package org.zerock.mallapi.repository;

import lombok.extern.log4j.Log4j2;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.zerock.mallapi.domain.Member;
import org.zerock.mallapi.domain.MemberRole;

@SpringBootTest
@Log4j2
public class MemberRepositoryTests {
    @Autowired
    private MemberRepository memberRepository;
    @Autowired
    private PasswordEncoder passwordEncoder;
}
```

테스트 코드를 이용한 등록/조회 확인

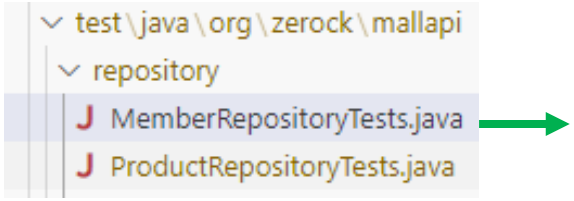
→ 엔티티의 등록




```java
@Test
public void testInsertMember(){

    for (int i = 0; i < 10 ; i++) {
        Member member = Member.builder()
                .email("user"+i+"@aaa.com")
                .pw(passwordEncoder.encode("1111"))
                .nickname("USER"+i)
                .build();

        member.addRole(MemberRole.USER);
        if(i >= 5)
            member.addRole(MemberRole.MANAGER);
        if(i >=8)
            member.addRole(MemberRole.ADMIN);
        memberRepository.save(member);
    }
}
```

테스트 코드를 이용한 등록/조회 확인

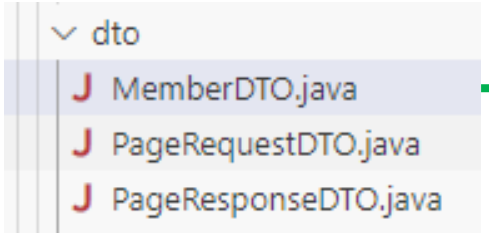
→ 엔티티의 등록




```java
@Test
public void testRead() {

    String email = "user9@aaa.com";

    Member member = memberRepository.getWithRoles(email);

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

}
```

DTO와 인증 처리 서비스

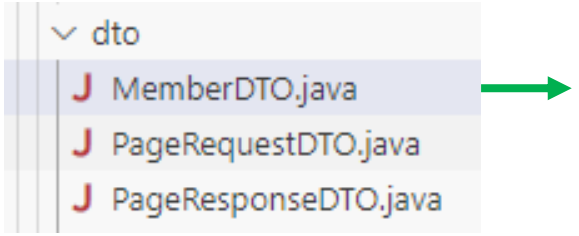
→ dto 패키지에 MemberDTO 클래스 추가




```java
package org.zerock.mallapi.dto;

import java.util.*;
import java.util.stream.Collectors;

import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@ToString
public class MemberDTO extends User {

}
```

DTO와 인증 처리 서비스

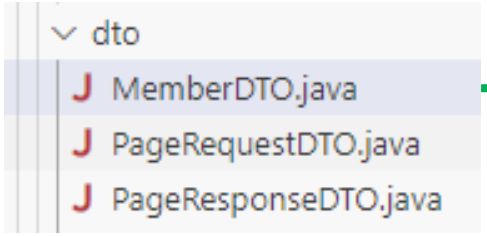
→ dto 패키지에 MemberDTO 클래스 추가




```java
private String email;

private String pw;

private String nickname;

private boolean social;

private List<String> roleNames = new ArrayList<>();
```

DTO와 인증 처리 서비스

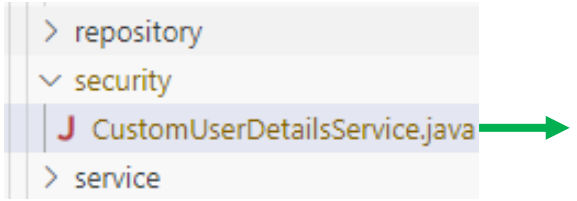
→ dto 패키지에 MemberDTO 클래스 추가




```java
  public MemberDTO( String email, String pw, String nickname, boolean social,
                    List<String> roleNames) {
    super(email,pw, roleNames.stream().map(str ->
          new SimpleGrantedAuthority("ROLE_"+str)).collect(Collectors.toList()));
    this.email = email;
    this.pw = pw;
    this.nickname = nickname;
    this.social = social;
    this.roleNames = roleNames;
  }

  public Map<String, Object> getClaims() {
    Map<String, Object> dataMap = new HashMap<>();
    dataMap.put("email", email);
    dataMap.put("pw",pw);
    dataMap.put("nickname", nickname);
    dataMap.put("social", social);
    dataMap.put("roleNames", roleNames);
    return dataMap;
  }
}
```

UserDetailsService 구현

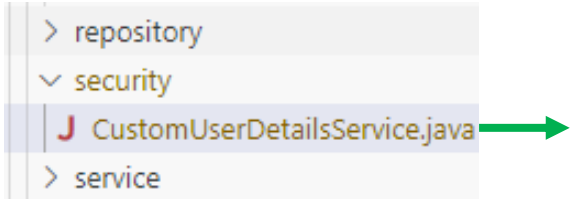
→ security패키지를 생성하고 CustomUserDetailsService클래스 추가




```java
package org.zerock.mallapi.security;

import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.zerock.mallapi.repository.MemberRepository;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;

@Service
@Log4j2
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {

}
```

UserDetailsService 구현

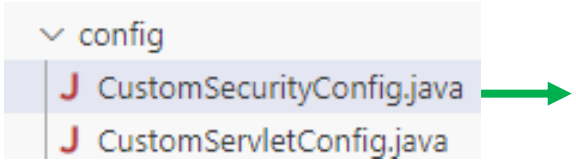
→ security패키지를 생성하고 CustomUserDetailsService클래스 추가




```
  private final MemberRepository memberRepository;

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

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

    return null;

  }
```

UserDetailsService 구현

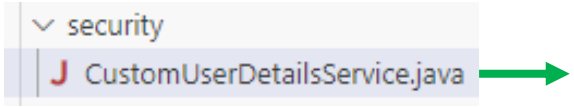
→ API 서버로 로그인을 할 수 있도록 CustomSecurityConfig의 설정




```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

  log.info("---------------security config---------------------");

  http.cors(httpSecurityCorsConfigurer -> {
    httpSecurityCorsConfigurer.configurationSource(corsConfigurationSource());
  });
  http.sessionManagement(sessionConfig ->
    sessionConfig.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

  http.csrf(config -> config.disable());
  http.formLogin(config -> {
    config.loginPage("/api/member/login");
  });
  return http.build();
}
```

UserDetailsService 구현

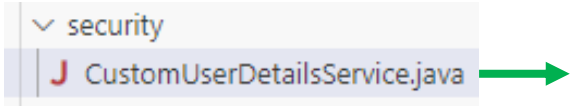
→ MemberRepository에서 MemberDTO타입으로 반환하도록 구현




```java
package org.zerock.mallapi.security;

import java.util.stream.Collectors;

import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.zerock.mallapi.domain.Member;
import org.zerock.mallapi.dto.MemberDTO;
import org.zerock.mallapi.repository.MemberRepository;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;

@Service
@Log4j2
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {

}
```

UserDetailsService 구현

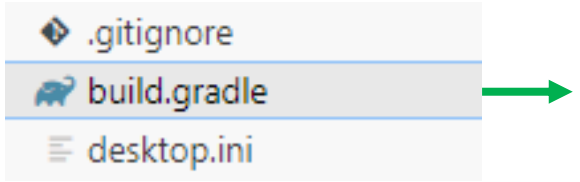
→ MemberRepository에서 MemberDTO타입으로 반환하도록 구현




```
  private final MemberRepository memberRepository;

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

    Member member = memberRepository.getWithRoles(username);
    if(member == null){
      throw new UsernameNotFoundException("Not Found");
    }
    MemberDTO memberDTO = new MemberDTO(
            member.getEmail(),
            member.getPw(),
            member.getNickname(),
            member.isSocial(),
            member.getMemberRoleList().stream()
                  .map(memberRole -> memberRole.name()).collect(Collectors.toList()));
    log.info(memberDTO);
    return memberDTO;
  }
}
```

로그인 성공 후 JSON데이터 생성

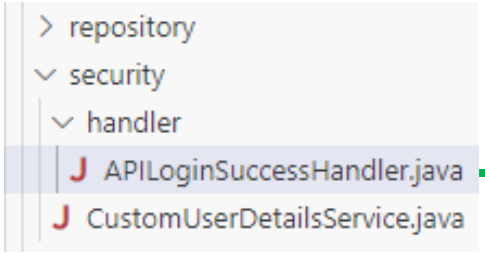
→ build.gradle파일에 Gson라이브러리 추가




```
dependencies {

  …생략

  implementation 'com.google.code.gson:gson:2.10.1'

}
```

로그인 성공 후 JSON데이터 생성

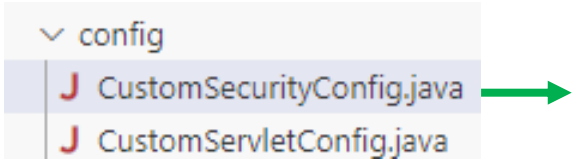
→ security패키지내에 handler 패키지 추가, APILoginSuccessHandler클래스 추가




```java
package org.zerock.mallapi.security.handler;

import java.io.IOException;

import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.extern.log4j.Log4j2;

@Log4j2
public class APILoginSuccessHandler implements AuthenticationSuccessHandler{
  @Override
  public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
      Authentication authentication) throws IOException, ServletException {
    log.info("------------------------------------");
    log.info(authentication);
  }
}
```

로그인 성공 후 JSON데이터 생성

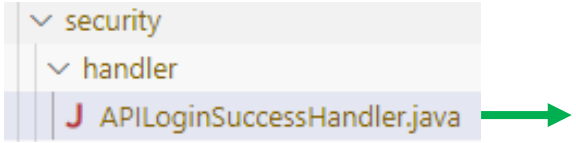
→ CustomSecurityConfig에 로그인 후 처리를 APILoginSuccessHandler로 설정




```
http.formLogin(config -> {
  config.loginPage("/api/member/login");
  config.successHandler(new APILoginSuccessHandler());
});
```

로그인 성공 후 JSON데이터 생성

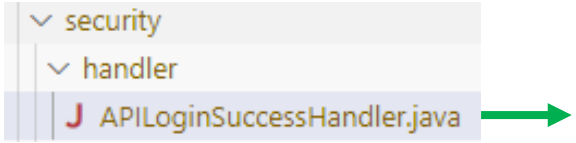
→ JSON 결과의 전송




```java
package org.zerock.mallapi.security.handler;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Map;

import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
import org.zerock.mallapi.dto.MemberDTO;

import com.google.gson.Gson;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.extern.log4j.Log4j2;

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

로그인 성공 후 JSON데이터 생성

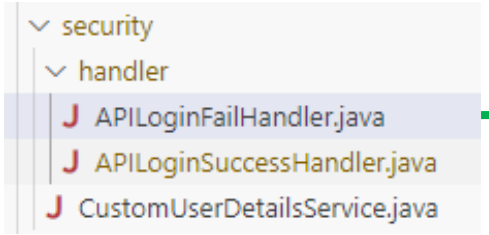
→ JSON 결과의 전송




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

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

  MemberDTO memberDTO = (MemberDTO)authentication.getPrincipal();
  Map<String, Object> claims = memberDTO.getClaims();
  claims.put("accessToken", "");
  claims.put("refreshToken", "");

  Gson gson = new Gson();
  String jsonStr = gson.toJson(claims);
  response.setContentType("application/json");

  PrintWriter printWriter = response.getWriter();
  printWriter.println(jsonStr);
  printWriter.close();
}
```

로그인 성공 후 JSON데이터 생성

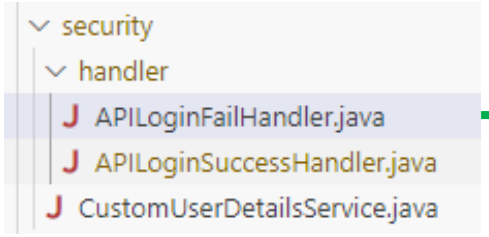
→ 인증 실패 처리




```java
package org.zerock.mallapi.security.handler;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Map;

import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.authentication.AuthenticationFailureHandler;

import com.google.gson.Gson;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.extern.log4j.Log4j2;

@Log4j2
public class APILoginFailHandler implements AuthenticationFailureHandler{

}
```

로그인 성공 후 JSON데이터 생성

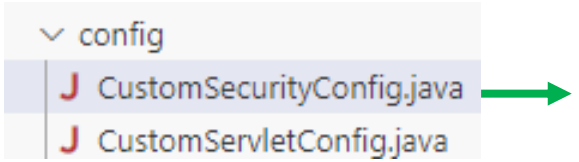
→ 인증 실패 처리




```java
@Override
public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response,
    AuthenticationException exception) throws IOException, ServletException {

  log.info("Login fail....." + exception);

  Gson gson = new Gson();

  String jsonStr = gson.toJson(Map.of("error", "ERROR_LOGIN"));

  response.setContentType("application/json");
  PrintWriter printWriter = response.getWriter();
  printWriter.println(jsonStr);
  printWriter.close();

}
```

로그인 성공 후 JSON데이터 생성

→ 인증 실패 처리 : CustomSecurityConfig에 로그인 실패 처리시에 대한 설정




```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

  log.info("----------------security config----------------------");

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

  http.sessionManagement(sessionConfig ->
    sessionConfig.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

  http.csrf(config -> config.disable());

  http.formLogin(config -> {
    config.loginPage("/api/member/login");
    config.successHandler(new APILoginSuccessHandler());
    config.failureHandler(new APILoginFailHandler());
  });
  return http.build();
}
```

API 서버의 특징과 JWT 기반 인증 방식

→ 상태 유지 없음

- API 서버는 상태를 유지하지 않고 세션 및 쿠키를 활용하지 않음.

→ JWT 문자열 토큰

- API 호출에 사용되는 JWT 문자열 토큰 (Access Token)으로 사용자 인증 수행.
  API 호출 시마다 **JWT Access Token**을 전달하여 사용자 인증을 구현.

→ 유효시간 제한

- 보안을 강화하기 위해 노출된 토큰을 방지하기 위해 Access Token의 유효시간을 짧게 설정.
  자주 발급받는 번거로움을 피하기 위해 **Refresh Token**을 사용하여 새로운 Access Token 발급 가능.

→ Refresh Token 특징

- Refresh Token은 긴 유효시간을 가지며, 새로운 Access Token 발급의 장치 역할 수행.
  Refresh Token을 활용하여 사용자 인증과 보안 유지하면서 Access Token 재발급 편의성 제공.

JWT구성

→ https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api

Home » io.jsonwebtoken » jjwt-api

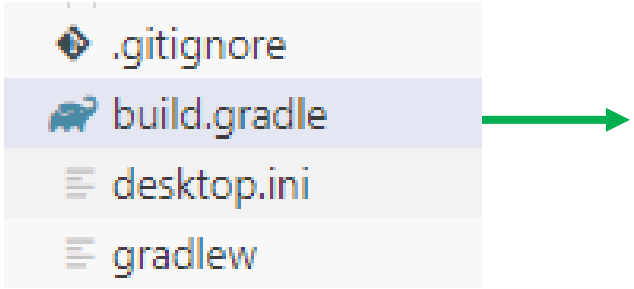
JJWT :: API

JJWT :: API

License	Apache 2.0
Categories	JWT Libraries
Tags	api jwt json security
Ranking	#978 in MvnRepository (See Top Artifacts) #3 in JWT Libraries

JWT구성

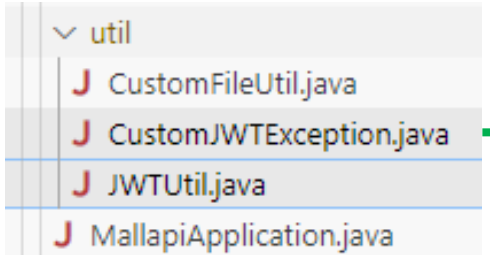
→ build.gradle 파일에 라이브러리 추가




```
dependencies {

  ...

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

}
```

JWT 문자열의 생성과 검증

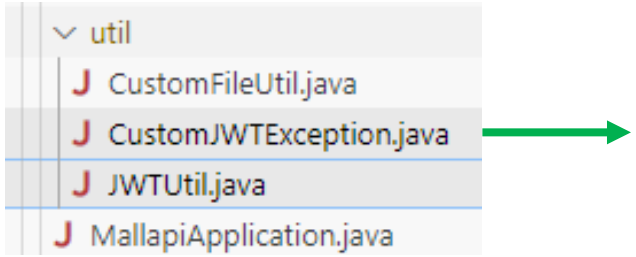
→ util패키지 생성, JWTUtil 클래스와 CustomJWTException클래스 추가




```java
package org.zerock.mallapi.util;

public class CustomJWTException extends RuntimeException{

  public CustomJWTException(String msg){

      super(msg);

  }

}
```

JWT 문자열의 생성과 검증

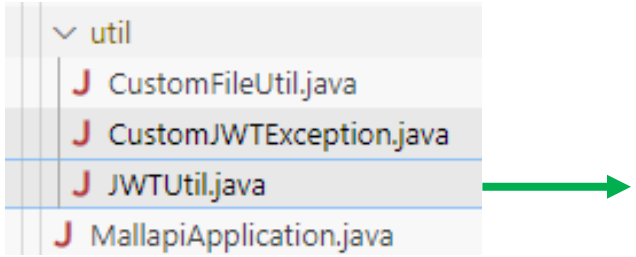
→ util패키지 생성, CustomJWTException클래스 추가




```java
package org.zerock.mallapi.util;

public class CustomJWTException extends RuntimeException{

  public CustomJWTException(String msg){

      super(msg);

  }

}
```

JWT 문자열의 생성과 검증

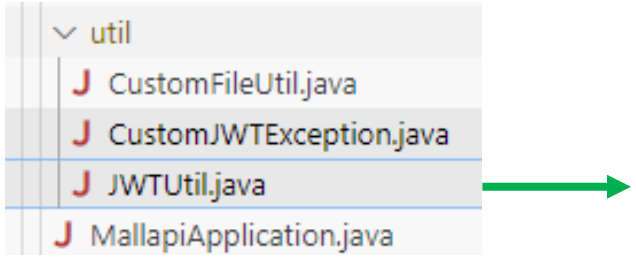
→ util패키지 생성, JWTUtil 클래스 추가




```java
package org.zerock.mallapi.util;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import lombok.extern.log4j.Log4j2;

import java.time.ZonedDateTime;
import java.util.*;

import javax.crypto.SecretKey;

@Log4j2
public class JWTUtil {

    private static String key = "1234567890123456789012345678901234567890";

}
```

JWT 문자열의 생성과 검증

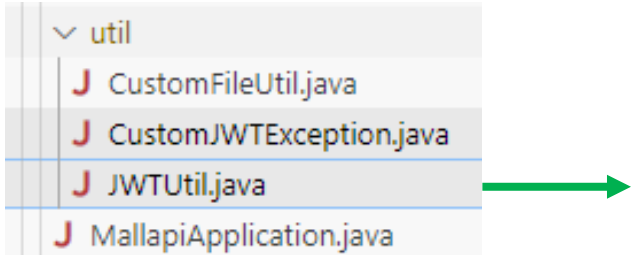
→ JWTUtil 클래스 : JWT문자열 생성을 위한 generateToken( )




```java
public static String generateToken(Map<String, Object> valueMap, int min) {

    SecretKey key = null;

    try{
        key = Keys.hmacShaKeyFor(JWTUtil.key.getBytes("UTF-8"));
    }catch(Exception e){
        throw new RuntimeException(e.getMessage());
    }

    String jwtStr = Jwts.builder()
        .setHeader(Map.of("typ","JWT"))
        .setClaims(valueMap)
        .setIssuedAt(Date.from(ZonedDateTime.now().toInstant()))
        .setExpiration(Date.from(ZonedDateTime.now().plusMinutes(min).toInstant()))
        .signWith(key)
        .compact();

    return jwtStr;
}
```

JWT 문자열의 생성과 검증

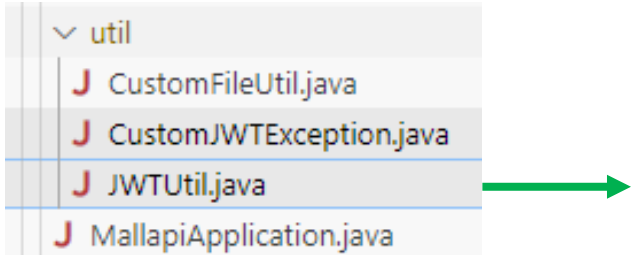
→ JWTUtil 클래스 : 검증을 위한 validateToken( )




```java
public static Map<String, Object> validateToken(String token) {
    Map<String, Object> claim = null;
    try{
        SecretKey key = Keys.hmacShaKeyFor(JWTUtil.key.getBytes("UTF-8"));
        claim = Jwts.parserBuilder()
                .setSigningKey(key)
                .build()
                .parseClaimsJws(token) // 파싱 및 검증, 실패 시 에러
                .getBody();
    }catch(MalformedJwtException malformedJwtException){
        throw new CustomJWTException("MalFormed");
    }catch(ExpiredJwtException expiredJwtException){
        throw new CustomJWTException("Expired");
    }catch(InvalidClaimException invalidClaimException){
        throw new CustomJWTException("Invalid");
    }catch(JwtException jwtException){
        throw new CustomJWTException("JWTError");
    }catch(Exception e){
        throw new CustomJWTException("Error");
    }
    return claim;
}
```

JWT 문자열의 생성과 검증

→ JWTUtil 클래스 : 검증을 위한 validateToken( )




```java
public static Map<String, Object> validateToken(String token) {
    Map<String, Object> claim = null;
    try{
        SecretKey key = Keys.hmacShaKeyFor(JWTUtil.key.getBytes("UTF-8"));
        claim = Jwts.parserBuilder()
                .setSigningKey(key)
                .build()
                .parseClaimsJws(token) // 파싱 및 검증, 실패 시 에러
                .getBody();
    }catch(MalformedJwtException malformedJwtException){
        throw new CustomJWTException("MalFormed");
    }catch(ExpiredJwtException expiredJwtException){
        throw new CustomJWTException("Expired");
    }catch(InvalidClaimException invalidClaimException){
        throw new CustomJWTException("Invalid");
    }catch(JwtException jwtException){
        throw new CustomJWTException("JWTError");
    }catch(Exception e){
        throw new CustomJWTException("Error");
    }
    return claim;
}
```

JWT 문자열의 생성과 검증

→ 로그인 성공시에 동작하는

APILoginSuccessHandler.java


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

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

    MemberDTO memberDTO = (MemberDTO)authentication.getPrincipal();

    Map<String, Object> claims = memberDTO.getClaims();

    String accessToken = JWTUtil.generateToken(claims, 10);
    String refreshToken = JWTUtil.generateToken(claims,60*24);

    claims.put("accessToken", accessToken);
    claims.put("refreshToken", refreshToken);

    Gson gson = new Gson();
    String jsonStr = gson.toJson(claims);

    response.setContentType("application/json");
    PrintWriter printWriter = response.getWriter();
    printWriter.println(jsonStr);
    printWriter.close();
  }
```

Access Token과 JWT 기반 인증 방식

→ Access Token 역할

→ API 서버의 특정 경로에 접근하기 위한 용도로 사용되는 토큰.
HTTP 요청의 Authorization 헤더에 값으로 포함되어 전달되며, 서버는 이를 확인하여 접근 여부 결정.

→ Authorization 헤더 구조

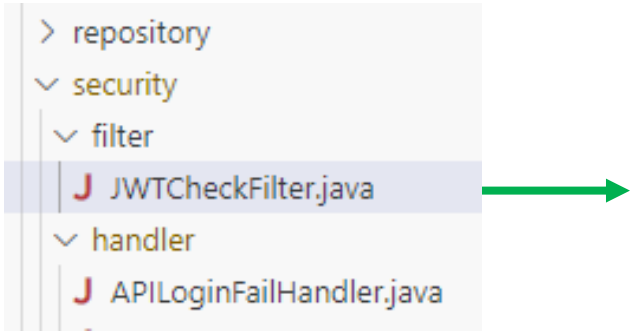
→ 'Bearer <토큰>' 형식으로 타입과 토큰이 공백으로 구분되어 포함됨.
주로 JWT를 활용하는 경우 **'Bearer' 타입**을 사용하여 **Authorization 헤더에 Access Token을 담아 전송**.

→ 접근 제어 구현

→ API 서버는 보호하고자 하는 자원에 대해 Access Token을 확인하여 유효성 여부 판단 후 접근 판단
필터, 스프링 MVC의 인터셉터, 스프링 시큐리티의 필터 등을 활용하여 서버 내에서 Access Token 유효성 체크 및 접근 제어 구현.

Access Token 체크 필터

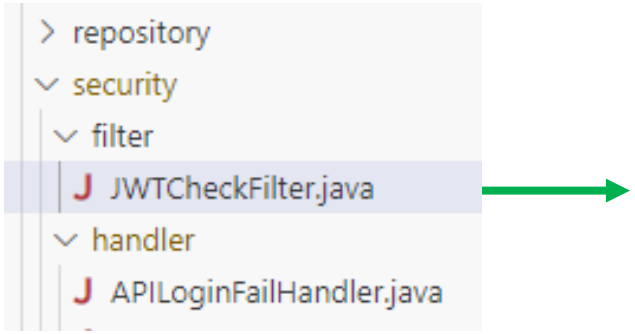
→ security패키지에 filter패키지를 추가, JWTCheckFilter클래스 추가




```java
package org.zerock.mallapi.security.filter;

import java.io.IOException;

import org.springframework.web.filter.OncePerRequestFilter;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.extern.log4j.Log4j2;

@Log4j2
public class JWTCheckFilter extends OncePerRequestFilter {

}
```

Access Token 체크 필터

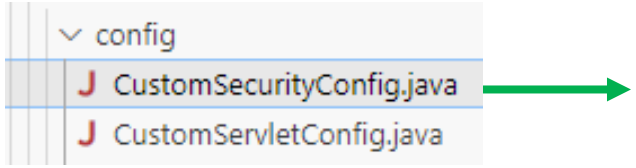
→ security패키지에 filter패키지를 추가, JWTCheckFilter클래스 추가




```java
  @Override
  protected boolean shouldNotFilter(HttpServletRequest request)
                                                  throws ServletException {
    String path = request.getRequestURI();
    log.info("check uri.............." + path);
    return false;
  }

  @Override
  protected void doFilterInternal(HttpServletRequest request,
              HttpServletResponse response, FilterChain filterChain)
                                  throws ServletException, IOException {
    log.info("--------------------------------");
    log.info("--------------------------------");
    log.info("--------------------------------");
    filterChain.doFilter(request, response); //통과
  }
}
```

Access Token 체크 필터

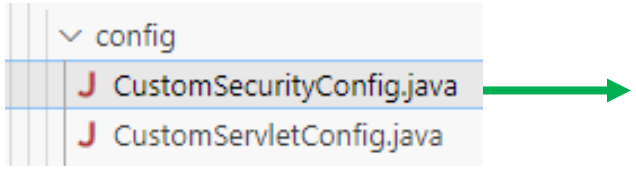
→ CustomSecurityConfig에 JWTCheckFilter를 추가




```
  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    log.info("----------------security config---------------");
    http.cors(httpSecurityCorsConfigurer -> {
      httpSecurityCorsConfigurer.configurationSource(corsConfigurationSource());
    });
    http.sessionManagement(sessionConfig ->
sessionConfig.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
    http.csrf(config -> config.disable());

    http.formLogin(config -> {
      config.loginPage("/api/member/login");
      config.successHandler(new APILoginSuccessHandler());
      config.failureHandler(new APILoginFailHandler());
    });

    http.addFilterBefore(new JWTCheckFilter(),
            UsernamePasswordAuthenticationFilter.class); //JWT체크

    return http.build();
  }
```

필터를 통한 검증/예외 처리

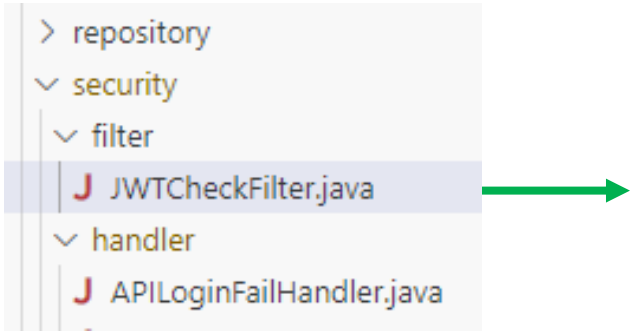
→ JWTCheckFilter.java




```java
@Override
protected boolean shouldNotFilter(HttpServletRequest request) throws ServletException {

  // Preflight요청은 체크하지 않음
  if(request.getMethod().equals("OPTIONS")){
    return true;
  }

  String path = request.getRequestURI();
  log.info("check uri............." + path);

  //api/member/ 경로의 호출은 체크하지 않음
  if(path.startsWith("/api/member/")) {
    return true;
  }

  //이미지 조회 경로는 체크하지 않음
  if(path.startsWith("/api/products/view/")) {
    return true;
  }
  return false;
}
```

필터를 통한 검증/예외 처리

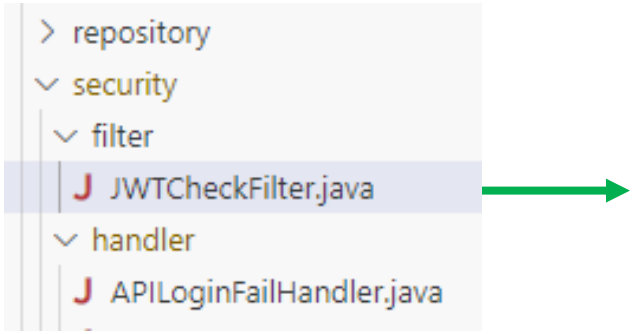
→ JWTCheckFilter.java




```java
@Log4j2
public class JWTCheckFilter extends OncePerRequestFilter {

  @Override
  protected boolean shouldNotFilter(HttpServletRequest request)
                                                  throws ServletException {

    …생략

  }
```

필터를 통한 검증/예외 처리

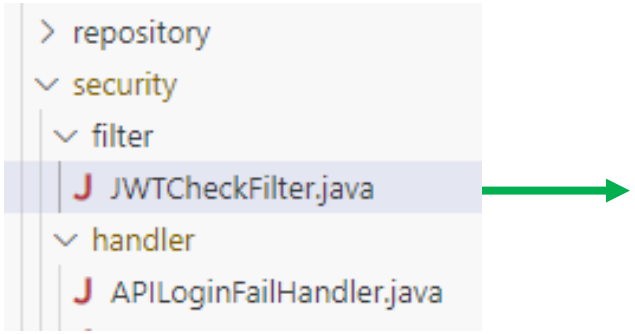
→ JWTCheckFilter.java




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

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

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

  try {
    //Bearer accestoken...
    String accessToken = authHeaderStr.substring(7);
    Map<String, Object> claims = JWTUtil.validateToken(accessToken);

    log.info("JWT claims: " + claims);

    filterChain.doFilter(request, response);

  }catch(Exception e){

    log.error("JWT Check Error..............");
    log.error(e.getMessage());
```

필터를 통한 검증/예외 처리

→ JWTCheckFilter.java

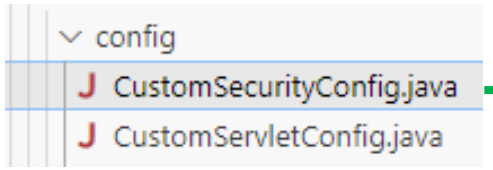



```
        Gson gson = new Gson();
        String msg = gson.toJson(Map.of("error", "ERROR_ACCESS_TOKEN"));

        response.setContentType("application/json");
        PrintWriter printWriter = response.getWriter();
        printWriter.println(msg);
        printWriter.close();
    }
  }
}
```

@PreAuthorize를 통한 접근 권한 처리

→ 시큐리티 설정 :

org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity 를 추가

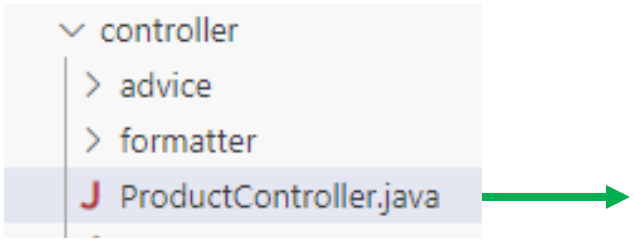



```
@Configuration
@Log4j2
@RequiredArgsConstructor
@EnableMethodSecurity
public class CustomSecurityConfig {
…생략
```

@PreAuthorize를 통한 접근 권한 처리

→ 특정 권한을 가진 사용자만이 접근할 수 있도록 제한

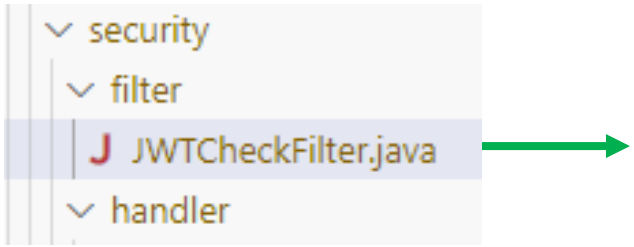
각 메소드에 org.springframework.security.access.prepost.PreAuthorize를 이용




```java
import org.springframework.security.access.prepost.PreAuthorize;

…생략

@PreAuthorize("hasAnyRole('ROLE_USER','ROLE_ADMIN')") //임시로 권한 설정
@GetMapping("/list")
public PageResponseDTO<ProductDTO> list(PageRequestDTO pageRequestDTO) {

  log.info("list............." + pageRequestDTO);

  return productService.getList(pageRequestDTO);

}
```

@PreAuthorize를 통한 접근 권한 처리

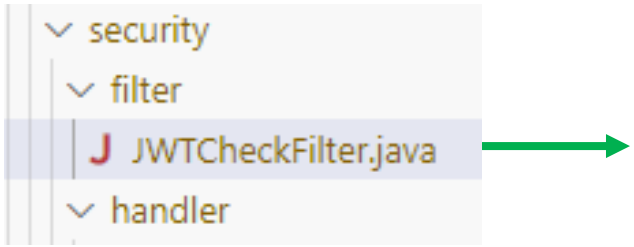
→ JWTCheckFilter에서 JWT인증 정보를 이용해서 사용자를 구성




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

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

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

  try {
    //Bearer accestoken...
    String accessToken = authHeaderStr.substring(7);
    Map<String, Object> claims = JWTUtil.validateToken(accessToken);

    log.info("JWT claims: " + claims);

    String email = (String) claims.get("email");
    String pw = (String) claims.get("pw");
    String nickname = (String) claims.get("nickname");
    Boolean social = (Boolean) claims.get("social");
    List<String> roleNames = (List<String>) claims.get("roleNames");
```

@PreAuthorize를 통한 접근 권한 처리

→ JWTCheckFilter에서 JWT인증 정보를 이용해서 사용자를 구성




```
        MemberDTO memberDTO = new MemberDTO( email, pw, nickname, social.booleanValue(),
                                             roleNames);
        log.info("-----------------------------------");
        log.info(memberDTO);
        log.info(memberDTO.getAuthorities());

        UsernamePasswordAuthenticationToken authenticationToken
        = new UsernamePasswordAuthenticationToken(memberDTO,pw,memberDTO.getAuthorities());

        SecurityContextHolder.getContext().setAuthentication(authenticationToken);

        filterChain.doFilter(request, response);

    } catch(Exception e) {
```

@PreAuthorize를 통한 접근 권한 처리

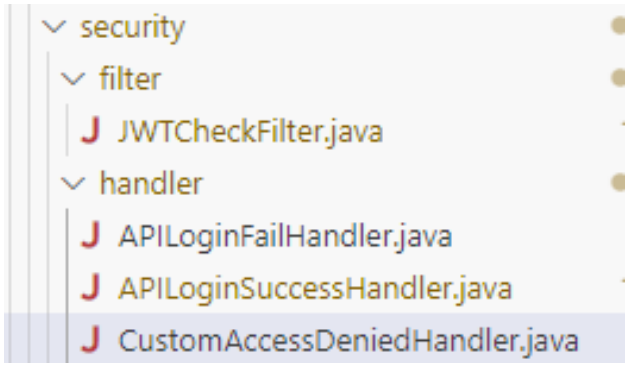
→ JWTCheckFilter에서 JWT인증 정보를 이용해서 사용자를 구성


```
      log.error("JWT Check Error..............");
      log.error(e.getMessage());

      Gson gson = new Gson();
      String msg = gson.toJson(Map.of("error", "ERROR_ACCESS_TOKEN"));

      response.setContentType("application/json");
      PrintWriter printWriter = response.getWriter();
      printWriter.println(msg);
      printWriter.close();

    }
  }
... 생략
```

@PreAuthorize를 통한 접근 권한 처리

→ 메소드 접근 제한 예외 처리

security/handler 패키지에 CustomAccessDeniedHandler클래스를 추가

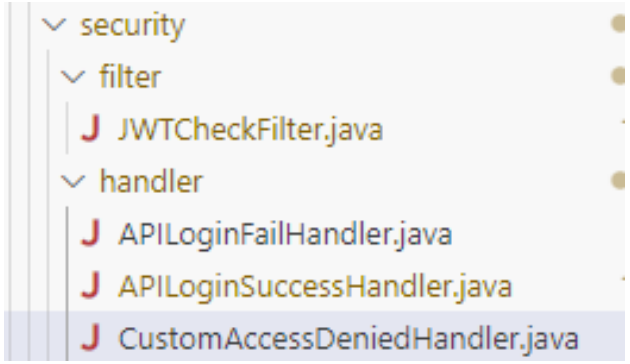



```java
package org.zerock.mallapi.security.handler;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Map;

import org.springframework.http.HttpStatus;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.web.access.AccessDeniedHandler;

import com.google.gson.Gson;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class CustomAccessDeniedHandler implements AccessDeniedHandler{

}
```

@PreAuthorize를 통한 접근 권한 처리

→ 메소드 접근 제한 예외 처리

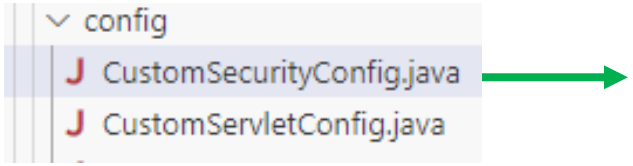
security/handler 패키지에 CustomAccessDeniedHandler클래스를 추가




```java
public class CustomAccessDeniedHandler implements AccessDeniedHandler{

  @Override
  public void handle(HttpServletRequest request,
                     HttpServletResponse response,
                     AccessDeniedException accessDeniedException)
                                    throws IOException, ServletException {
    Gson gson = new Gson();

    String jsonStr = gson.toJson(Map.of("error", "ERROR_ACCESSDENIED"));
    response.setContentType("application/json");
    response.setStatus(HttpStatus.FORBIDDEN.value());
    PrintWriter printWriter = response.getWriter();
    printWriter.println(jsonStr);
    printWriter.close();
  }
```

@PreAuthorize를 통한 접근 권한 처리

→ CustomSecurityConfig에 접근제한시 CustomAccessDeniedHandler를 이용하도록 설정




```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception{

  …생략

  http.exceptionHandling(config -> {
    config.accessDeniedHandler(new CustomAccessDeniedHandler());
  });

  return http.build();
}
```

Refresh Token

- Access Token이 없거나 잘못된 JWT인 경우 -> 예외 메시지 발생
- Access Token의 유효기간이 남아있는 경우 -> 전달된 토큰들을 그대로 전송
- Access Token은 만료, Refresh Token은 만료되지 않은 경우 -> 새로운 Access Token
  - Refresh Token의 유효 기한이 얼만 남지 않은 경우 -> 새로운 Refresh Token
  - Refresh Token의 유효 기한이 충분히 남은 경우 -> 기존의 Refresh Token

Refresh Token의 발행

→ controller패키지에 APIRefreshController 추가하고 Access Token과 Refresh Token을 파라미터 처리


```java
package org.zerock.mallapi.controller;

import java.util.Map;

import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.zerock.mallapi.dto.MemberDTO;
import org.zerock.mallapi.service.MemberService;
import org.zerock.mallapi.util.CustomJWTException;
import org.zerock.mallapi.util.JWTUtil;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;

@RestController
@RequiredArgsConstructor
@Log4j2
public class APIRefreshController {

}
```

Refresh Token의 발행

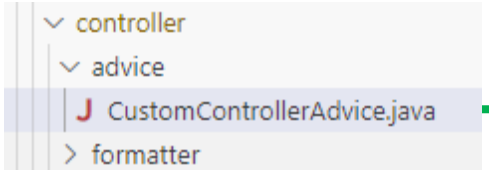

```java
@RequestMapping("/api/member/refresh")
 public Map<String, Object> refresh(@RequestHeader("Authorization") String authHeader,
                                    String refreshToken){

   if(refreshToken == null) {
     throw new CustomJWTException("NULL_REFRASH");
   }

   if(authHeader == null || authHeader.length() < 7) {
     throw new CustomJWTException("INVALID_STRING");
   }

   String accessToken = authHeader.substring(7);
   //Access 토큰이 만료되지 않았다면
   if(checkExpiredToken(accessToken) == false ) {
     return Map.of("accessToken", accessToken, "refreshToken", refreshToken);
   }

   //Refresh토큰 검증
   Map<String, Object> claims = JWTUtil.validateToken(refreshToken);
   log.info("refresh ... claims: " + claims);

   String newAccessToken = JWTUtil.generateToken(claims, 10);

   String newRefreshToken =  checkTime((Integer)claims.get("exp")) == true ?
                                  JWTUtil.generateToken(claims, 60*24) : refreshToken;
   return Map.of("accessToken", newAccessToken, "refreshToken", newRefreshToken);

 }
```

Refresh Token의 발행


```java
  //시간이 1시간 미만으로 남았다면
  private boolean checkTime(Integer exp) {

    //JWT exp를 날짜로 변환
    java.util.Date expDate = new java.util.Date( (long)exp * (1000 ));
    //현재 시간과의 차이 계산 - 밀리세컨즈
    long gap   = expDate.getTime() - System.currentTimeMillis();
    //분단위 계산
    long leftMin = gap / (1000 * 60);
    //1시간도 안남았는지..
    return leftMin < 60;
  }

  private boolean checkExpiredToken(String token) {
    try{
      JWTUtil.validateToken(token);
    }catch(CustomJWTException ex) {
      if(ex.getMessage().equals("Expired")){
          return true;
      }
    }
    return false;
  }
}
```

Refresh Token의 발행

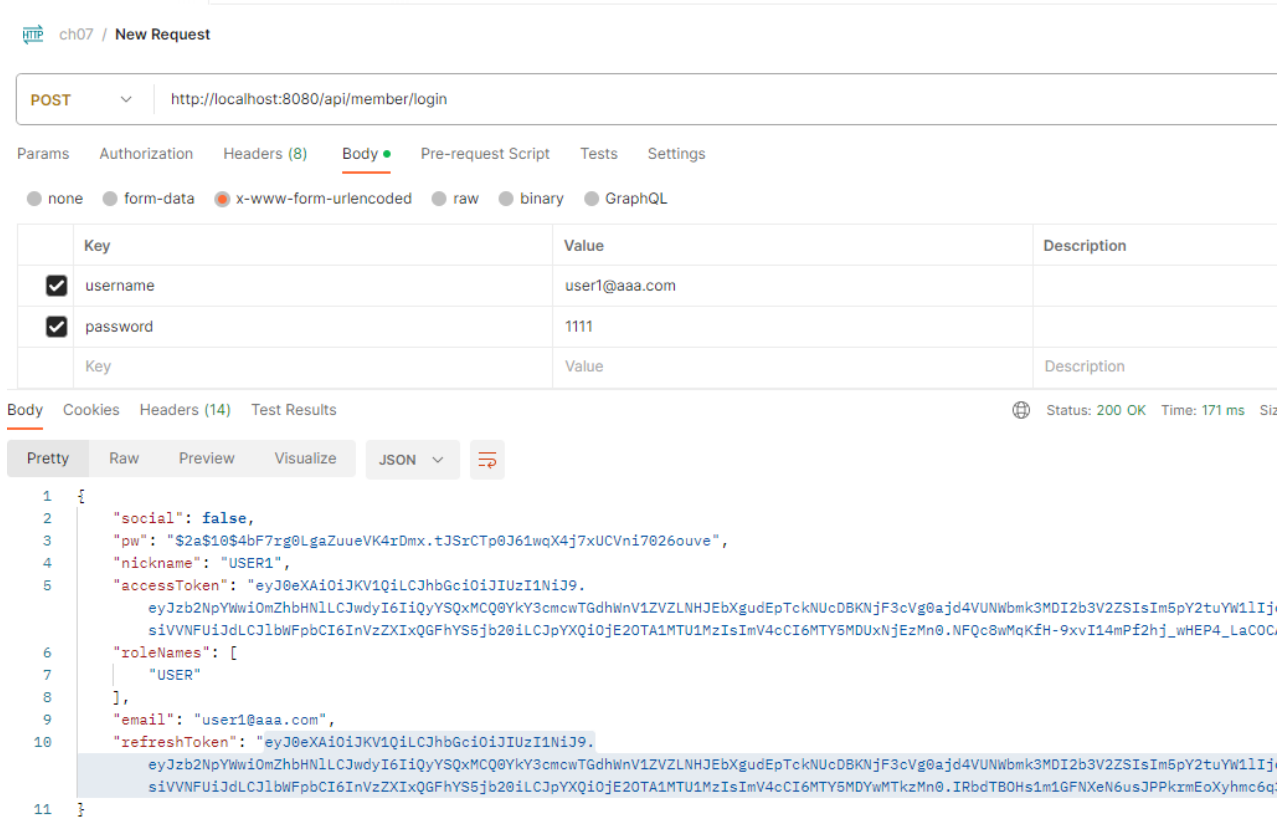
→ CustomControllerAdvice를 이용해서 예외 발생시 JSON 문자열을 전송하도록 구성




```java
@ExceptionHandler(CustomJWTException.class)
protected ResponseEntity<?> handleJWTException(CustomJWTException e) {

    String msg = e.getMessage();

    return ResponseEntity.ok().body(Map.of("error", msg));

}
```

Refresh Token 발행 테스트

→ '/api/member/login'을 통해서 'user1@aaa.com/1111'로 로그인 테스트



감사합니다.