Rest API 인증

Spring Boot

```java
package com.edu.entity;


@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;

}
```

```java
package com.edu.entity;

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

```java
package com.edu.entity;

@Entity
@Getter
@ToString
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class ClubMember extends BaseEntity {

     @Id
     private String email;
     private String password;
     private String name;
     private boolean fromSocial;

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

     public void addMemberRole(ClubMemberRole clubMemberRole) {
          roleSet.add(clubMemberRole);
     }

}
```

```java
package com.edu.repository;


import java.util.Optional;

import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.edu.entity.ClubMember;


public interface ClubMemberRepository extends JpaRepository<ClubMember, String> {

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

```java
package com.edu.entity;

@Entity
@Builder @AllArgsConstructor @NoArgsConstructor
@Getter @ToString
public class Note extends BaseEntity {

     @Id
     @GeneratedValue(strategy = GenerationType.IDENTITY)
     private Long num;
     private String title;
     private String content;

     @ManyToOne(fetch = FetchType.LAZY)
     private ClubMember writer;

     public void changeTitle(String title) {
          this.title = title;
     }

     public void changeContent(String content) {
          this.content = content;
     }

}
```

```java
package com.edu.dto;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class NoteDTO {
     private Long num;
     private String title;
     private String content;
     private String writerEmail;
     private LocalDateTime regDate, modDate;
}
```

```java
package com.edu.repository;

import java.util.List;
import java.util.Optional;

import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.edu.entity.Note;

public interface NoteRepository extends JpaRepository<Note, Long> {
    @EntityGraph(attributePaths = "writer", type = EntityGraph.EntityGraphType.LOAD)
    @Query("select n from Note n where n.num = :num")
    Optional<Note> getWithWriter(@Param("num") Long num);

    @EntityGraph(attributePaths = { "writer" }, type = EntityGraph.EntityGraphType.LOAD)
    @Query("select n from Note n where n.writer.email = :email")
    List<Note> getList(@Param("eamil") String email);
}
```

```java
package com.edu.service;

import java.util.List;

import com.edu.dto.NoteDTO;
import com.edu.entity.ClubMember;
import com.edu.entity.Note;

public interface NoteService {

     Long register(NoteDTO noteDTO);

     NoteDTO get(Long num);

     void modify(NoteDTO noteDTO);

     void remove(Long num);

     List<NoteDTO> getAllWithWriter(String writerEmail);

     default Note dtoToEntity(NoteDTO noteDTO) {
          Note note = Note.builder().num(noteDTO.getNum()).content(noteDTO.getContent())
          .writer(ClubMember.builder().email(noteDTO.getWriterEmail()).build()).build();

          return note;
     }

     default NoteDTO entityToDTO(Note note) {
          NoteDTO noteDTO = NoteDTO.builder().num(note.getNum()).title(note.getTitle()).content(note.getContent())
          .writerEmail(note.getWriter().getEmail()).regDate(note.getRegDate()).modDate(note.getModDate()).build();

          return noteDTO;
     }
}
```

```java
@Service
@RequiredArgsConstructor
public class NoteServiceImpl implements NoteService {

    private final NoteRepository noteRepository;

    @Override
    public Long register(NoteDTO noteDTO) {
        Note note = dtoToEntity(noteDTO);
        noteRepository.save(note);
        return note.getNum();
    }

    @Override
    public NoteDTO get(Long num) {
        Optional<Note> result =
                    noteRepository.getWithWriter(num);
        if (result.isPresent()) {
                return entityToDTO(result.get());
        }
        return null;
    }

    @Override
    public void modify(NoteDTO noteDTO) {
        Long num = noteDTO.getNum();
        Optional<Note> result = noteRepository.findById(num);
        if (result.isPresent()) {
            Note note = result.get();
            note.changeTitle(noteDTO.getTitle());
            note.changeContent(noteDTO.getContent());
            noteRepository.save(note);
        }

    }

    @Override
    public void remove(Long num) {
        noteRepository.deleteById(num);
    }


    @Override
    public List<NoteDTO> getAllWithWriter(String writerEmail) {

        List<Note> noteList = noteRepository.getList(writerEmail);

        return noteList.stream()
                       .map(note -> entityToDTO(note))
                       .collect(Collectors.toList());
    }

}
```

```java
package com.edu.controller;

@RestController  @RequiredArgsConstructor  @RequestMapping("/notes")
public class NoteController {

	private final NoteService service;

	@PostMapping("/add")
	public ResponseEntity<Long> register(@RequestBody NoteDTO noteDTO) {
		Long num = service.register(noteDTO);
		return new ResponseEntity<Long>(num, HttpStatus.OK);
	}
	@GetMapping("/{num}") // 특정 번호의 Note 확인하기
	public ResponseEntity<NoteDTO> read(@PathVariable("num") Long num) {
		NoteDTO noteDTO = service.get(num);
		return new ResponseEntity<NoteDTO>(noteDTO, HttpStatus.OK);
	}

	@GetMapping("/all")  // 특정 회원의 모든 Note 확인하기
	public ResponseEntity<List<NoteDTO>> getList(String email) {
		List<NoteDTO> list = service.getAllWithWriter(email);
		return new ResponseEntity<>(list, HttpStatus.OK);
	}
	@DeleteMapping("/{num}") // Note 삭제

	public ResponseEntity<String> remove(@PathVariable("num") Long num){
		service.remove(num);
		return new ResponseEntity<String>("remove", HttpStatus.OK);
	}

	@PutMapping("/{num}") // Note 수정
	public ResponseEntity<String> modify(@RequestBody NoteDTO noteDTO){
		service.modify(noteDTO);
		return new ResponseEntity<String>("modified", HttpStatus.OK);
	}
}
```

```java
package com.edu.dto;

import java.util.Collection;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.User;

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

@Getter
@Setter
@ToString(callSuper = true)
public class ClubAuthMemberDTO extends User  {

    private static final long serialVersionUID = 1L;
    private String email;
    private String name;
    private boolean fromSocial;

    public ClubAuthMemberDTO(String username, String password,
                                            Collection<? extends GrantedAuthority> authorities) {
        super(username, password, authorities);
        this.email = username;
    }

}
```

```java
package com.edu.service;

@Service
public class ClubUserDetaileService implements UserDetailsService {

    @Autowired
    private ClubMemberRepository repository;

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

        Optional<ClubMember> result = repository.findByEmail(username, false);

        if(!result.isPresent()) {
                throw new UsernameNotFoundException(username+" 사용자 없음");
        }
        ClubMember clubMember = result.get();

        ClubAuthMemberDTO clubAuthMember = new ClubAuthMemberDTO(
                                clubMember.getEmail(),
                                clubMember.getPassword(),
                                clubMember.getRoleSet().stream()
                                .map(role->new SimpleGrantedAuthority("ROLE_"+role.name()))
                                .collect(Collectors.toSet())
                                );
        return clubAuthMember;
    }
}
```

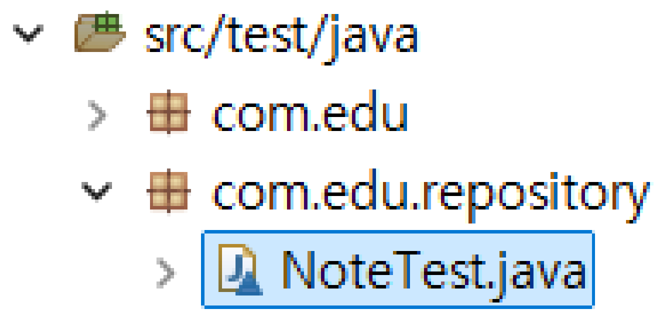
API 서버를 위한 필터

```
package com.edu.security;

import org.springframework.util.AntPathMatcher;
import org.springframework.web.filter.OncePerRequestFilter;

public class ApiCheckFilter extends OncePerRequestFilter {

     private AntPathMatcher antPathMatcher;
     private String pattern;

     public ApiCheckFilter(String pattern) {
          this.antPathMatcher = new AntPathMatcher();
          this.pattern = pattern;
     }

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

          if (antPathMatcher.match(pattern, request.getRequestURI())) {
               System.out.println("================ ApiCheckFilter ====================");
               return;
          }
          filterChain.doFilter(request, response);
     }
}
```

```java
package com.edu.config;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public ApiCheckFilter apiCheckFilter() {
        return new ApiCheckFilter("/notes/**/*");
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception{
       http
            .csrf(csrf -> csrf.disable())
            .formLogin(auth->auth.disable())
            .authorizeHttpRequests(authz -> authz
                .requestMatchers("/", "/notes/**").permitAll()
                .anyRequest().authenticated()
            )
            .sessionManagement(session->session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

        http
            .addFilterBefore(apiCheckFilter(), UsernamePasswordAuthenticationFilter.class);

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

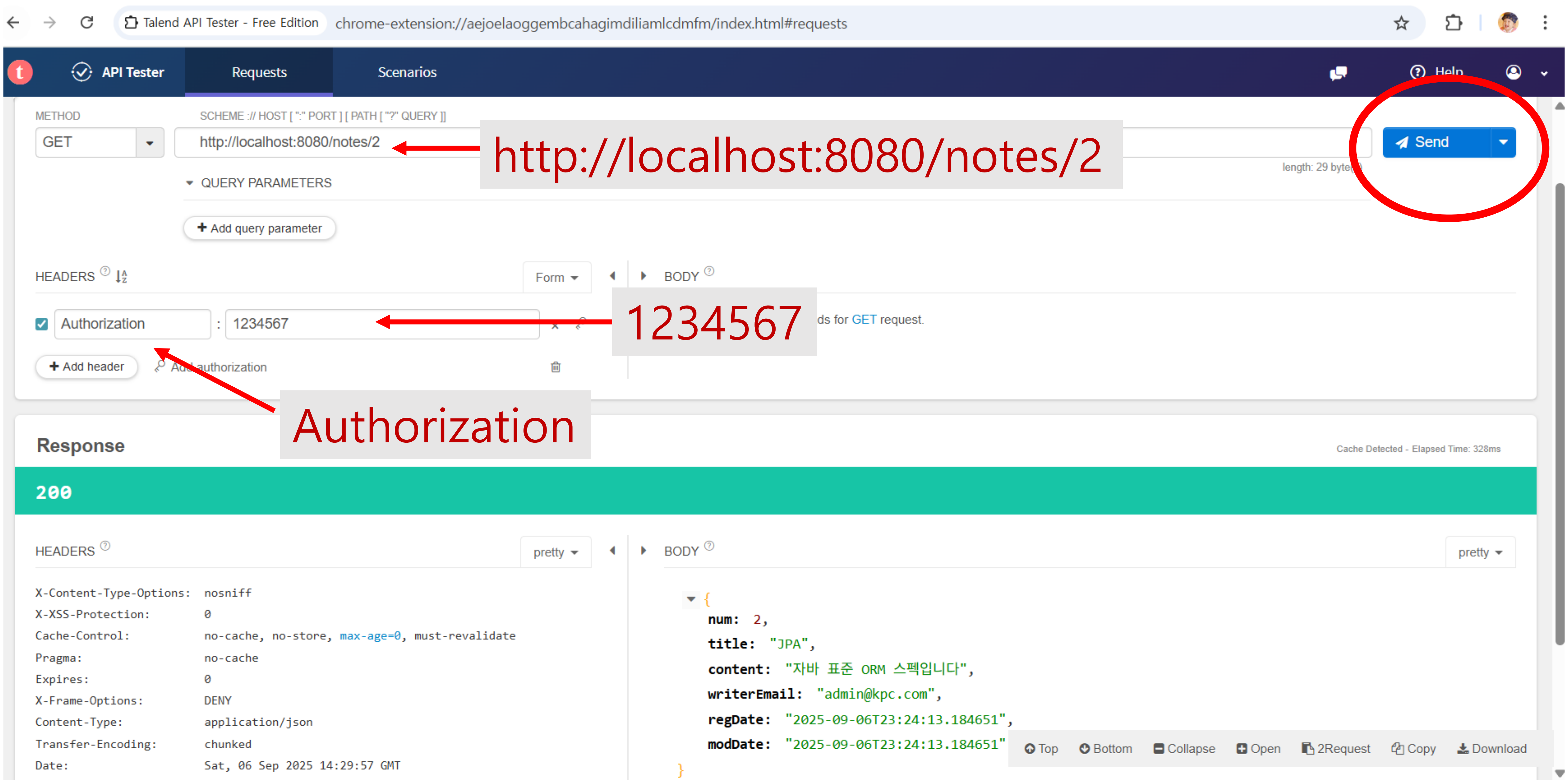
테스트 데이터 생성

```java
package com.edu.repository;

@SpringBootTest
class NoteTest {

	@Autowired
	private NoteRepository noteRepository;
	@Autowired
	private ClubMemberRepository memberRepository;

	@Test
	@DisplayName("note 등록이 정상 동작한다")
	public void insertNoteTest() {
		ClubMember member = ClubMember.builder()
				.email("admin@kpc.com")
				.password("123456789")
				.name("오정임")
				.fromSocial(false)
				.build();
		member.addMemberRole(ClubMemberRole.ADMIN);

		memberRepository.save(member);

		Note note = Note.builder()
				.title("JPA")
				.content("자바 표준 ORM 스펙입니다")
				.writer(member)
				.build();
		Note saveNote = noteRepository.save(note);

		assertThat(saveNote.getTitle()).isEqualTo(note.getTitle());
	}
}
```



Authorization 헤더 처리

```
public class ApiCheckFilter extends OncePerRequestFilter {
     ~ 생략 ~
     @Override
     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
                                                                                    throws ServletException, IOException {
          if(antPathMatcher.match(pattern, request.getRequestURI())){
               boolean checkHeader = checkAuthHeader(request);
               if(checkHeader) {
                    filterChain.doFilter(request, response);
                    return;
               }
               return;
          }
          filterChain.doFilter(request, response);
     }

     private boolean checkAuthHeader(HttpServletRequest request) {
          boolean checkResult = false;
          String authHeader = request.getHeader("Authorization");
          if(StringUtils.hasText(authHeader)) {
               if(authHeader.equals("1234567")) {
                    checkResult = true;
               }
          }

          return checkResult;
     }
}
```


Talend API Tester - Free Edition chrome-extension://aejoelaoggembcahagimdiliamlcdmfm/index.html#requests

METHOD: GET

http://localhost:8080/notes/2

length: 29 byte(s)

HEADERS

Authorization : 1234567

BODY

ds for GET request.

http://localhost:8080/notes/2

1234567

Authorization

Response

Cache Detected - Elapsed Time: 328ms

200

HEADERS pretty

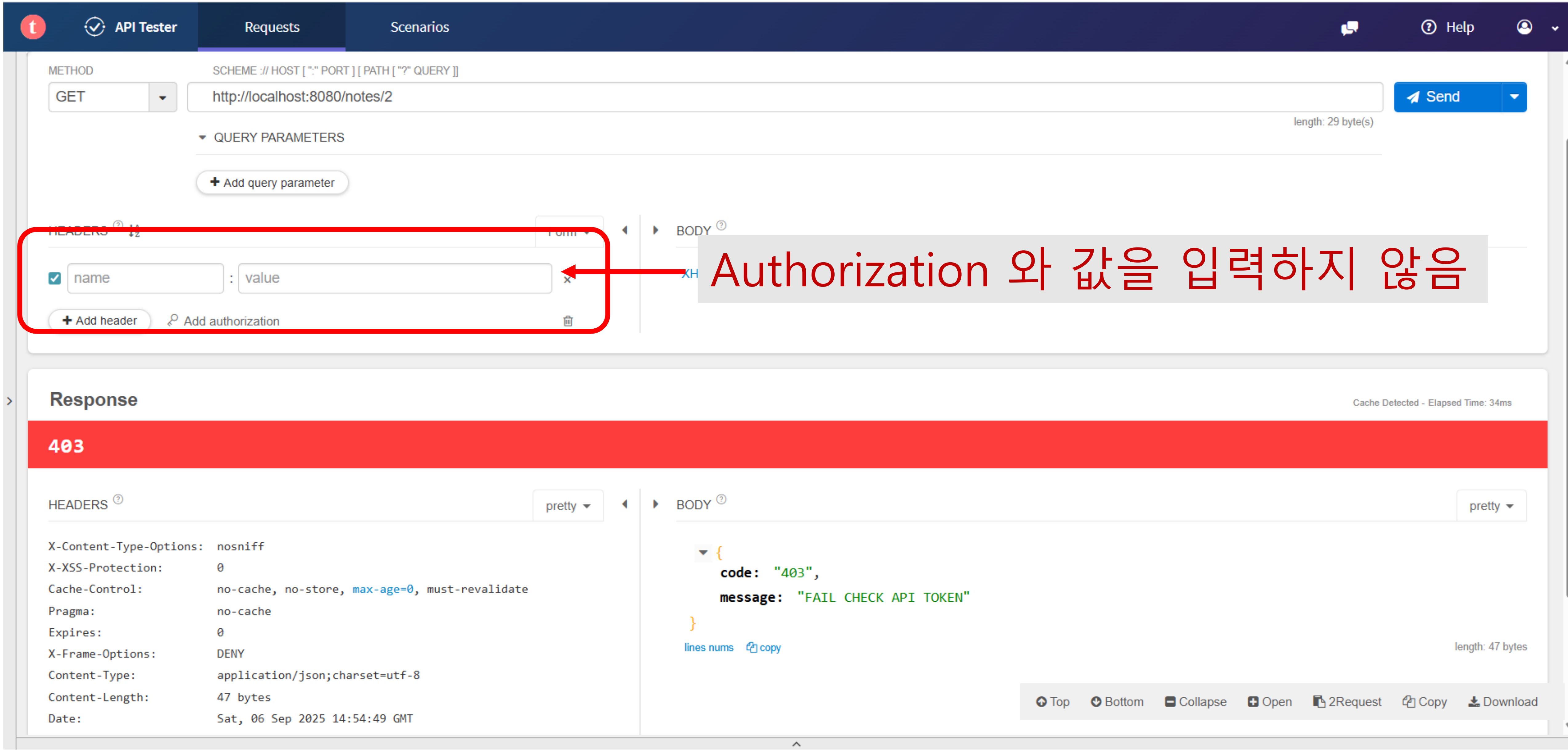
```
X-Content-Type-Options:   nosniff
X-XSS-Protection:         0
Cache-Control:            no-cache, no-store, max-age=0, must-revalidate
Pragma:                   no-cache
Expires:                  0
X-Frame-Options:          DENY
Content-Type:             application/json
Transfer-Encoding:        chunked
Date:                     Sat, 06 Sep 2025 14:29:57 GMT
```

BODY pretty

```
{
  num: 2,
  title: "JPA",
  content: "자바 표준 ORM 스펙입니다",
  writerEmail: "admin@kpc.com",
  regDate: "2025-09-06T23:24:13.184651",
  modDate: "2025-09-06T23:24:13.184651"
}
```

헤더 검증 실패 처리 – JSON

```java
public class ApiCheckFilter extends OncePerRequestFilter {

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

          if (antPathMatcher.match(pattern, request.getRequestURI())) {
               boolean checkHeader = checkAuthHeader(request);
               if (checkHeader) {
                    filterChain.doFilter(request, response);
                    return;
               }else {
                    response.setStatus(HttpServletResponse.SC_FORBIDDEN);
                    response.setContentType("application/json;charset=utf-8");
                    JSONObject json = new JSONObject();
                    String message = "FAIL CHECK API TOKEN";
                    try {
                         json.put("code","403");
                         json.put("message",message);
                    } catch (JSONException e) {   e.printStackTrace();   }

                    PrintWriter out = response.getWriter();
                    out.print(json);
                    return;
               }

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

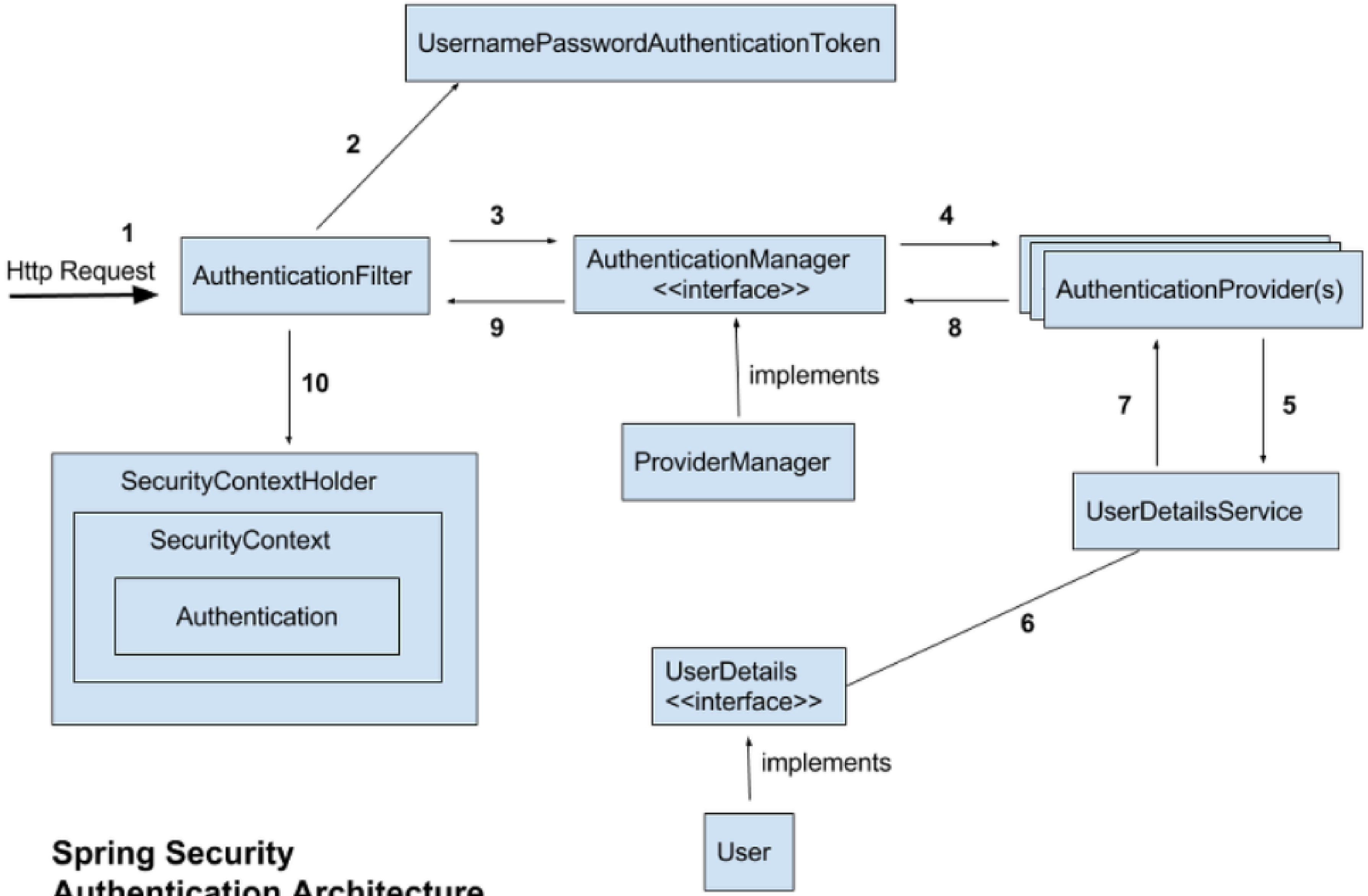


Authorization 와 값을 입력하지 않음

403

```
X-Content-Type-Options:  nosniff
X-XSS-Protection:        0
Cache-Control:           no-cache, no-store, max-age=0, must-revalidate
Pragma:                  no-cache
Expires:                 0
X-Frame-Options:         DENY
Content-Type:            application/json;charset=utf-8
Content-Length:          47 bytes
Date:                    Sat, 06 Sep 2025 14:54:49 GMT
```

```
{
  code: "403",
  message: "FAIL CHECK API TOKEN"
}
```



Spring Security
Authentication Architecture

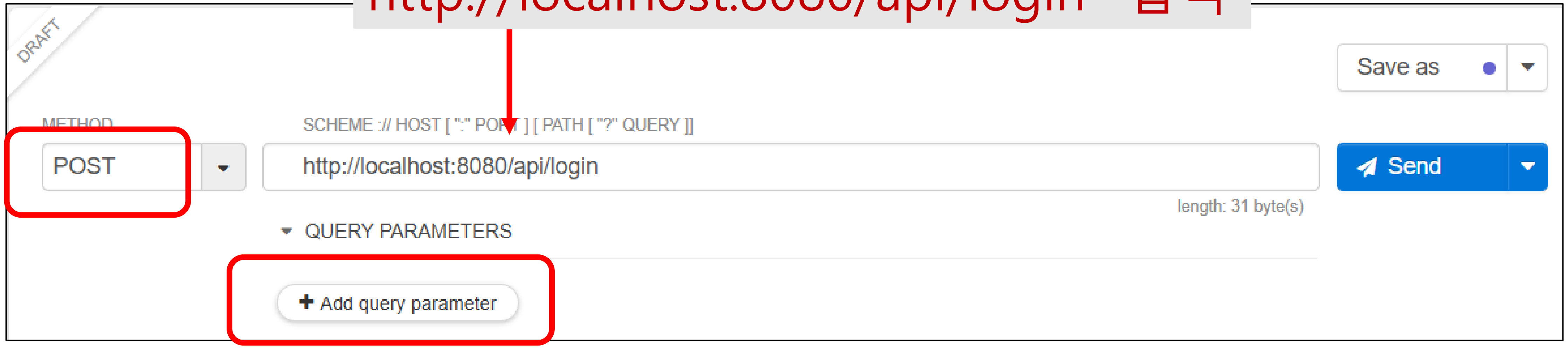
Chathuranga Tennakoon
www.springbootdev.com

API를 위한 인증 처리

```java
package com.edu.security;

import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.authentication.AbstractAuthenticationProcessingFilter;

public class ApiLoginFilter extends AbstractAuthenticationProcessingFilter {

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

	@Override
	public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
											throws AuthenticationException, IOException, ServletException {
		String email = request.getParameter("email");
		String pw = request.getParameter("pwd");

		System.out.println(email+"/"+pwd);

		if (email == null) {
				throw new BadCredentialsException("email cannot be null");
		}

		return null;
	}

}
```

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private AuthenticationConfiguration configuration;

    public SecurityConfig(AuthenticationConfiguration configuration) {
        this.configuration = configuration;
    }
    @Bean
    public ApiCheckFilter apiCheckFilter() {
        return new ApiCheckFilter("/notes/**/*");
    }
    @Bean
    public ApiLoginFilter apiLoginFilter() throws Exception {
        ApiLoginFilter apiLoginFilter = new ApiLoginFilter("/api/login");
        apiLoginFilter.setAuthenticationManager(configuration.getAuthenticationManager());
        return apiLoginFilter;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .formLogin(auth->auth.disable())
            .authorizeHttpRequests(authz -> authz
                .requestMatchers("/", "/notes/**","/api/login").permitAll()
                .anyRequest().authenticated()
            )

            .sessionManagement(session->session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

        http
            .addFilterBefore(apiCheckFilter(), UsernamePasswordAuthenticationFilter.class)
            .addFilterBefore(apiLoginFilter(), UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

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

http://localhost:8080/api/login 입력

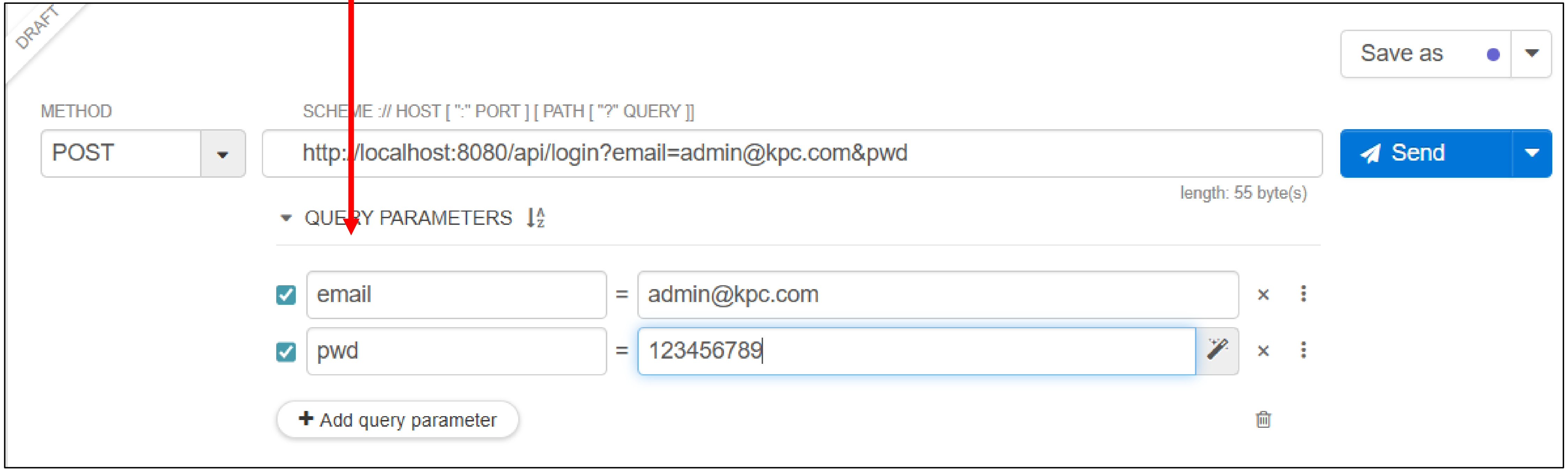


METHOD: POST

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

http://localhost:8080/api/login

length: 31 byte(s)



METHOD: POST

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

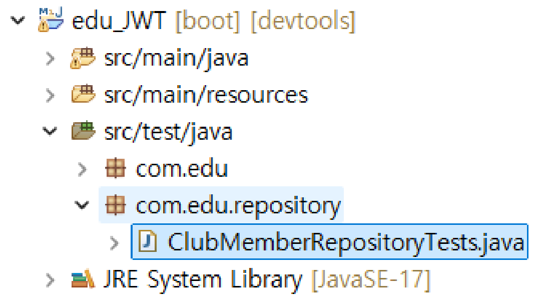
http://localhost:8080/api/login?email=admin@kpc.com&pwd

length: 55 byte(s)

✓ email	= admin@kpc.com
✓ pwd	= 123456789

테스트 사용자 등록

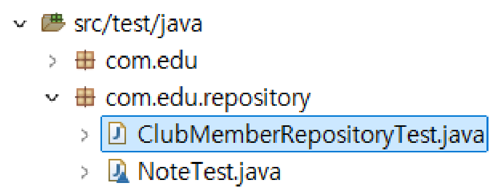
```java
@SpringBootTest
public class ClubMemberRepositoryTest {

    @Autowired
    private ClubMemberRepository clubMemberRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Test
    public void testCreateNewMember() {
        // Given
        String rawPassword = "12345";
        String encodedPassword = passwordEncoder.encode(rawPassword);

        ClubMember clubMember = ClubMember.builder()
                .email("guest@kt.com")
                .name("홍길동")
                .password(encodedPassword) // 인코딩된 비밀번호 사용
                .fromSocial(false)
                .build();

        clubMember.addMemberRole(ClubMemberRole.USER);

        // When
        clubMemberRepository.save(clubMember);

        // Then
        Optional<ClubMember> result = clubMemberRepository.findById("guest@kt.com");
        assertThat(result).isPresent();
    }
```



```
edu_JWT [boot] [devtools]
  src/main/java
  src/main/resources
  src/test/java
    com.edu
    com.edu.repository
      ClubMemberRepositoryTests.java
  JRE System Library [JavaSE-17]
```

테스트 사용자 등록

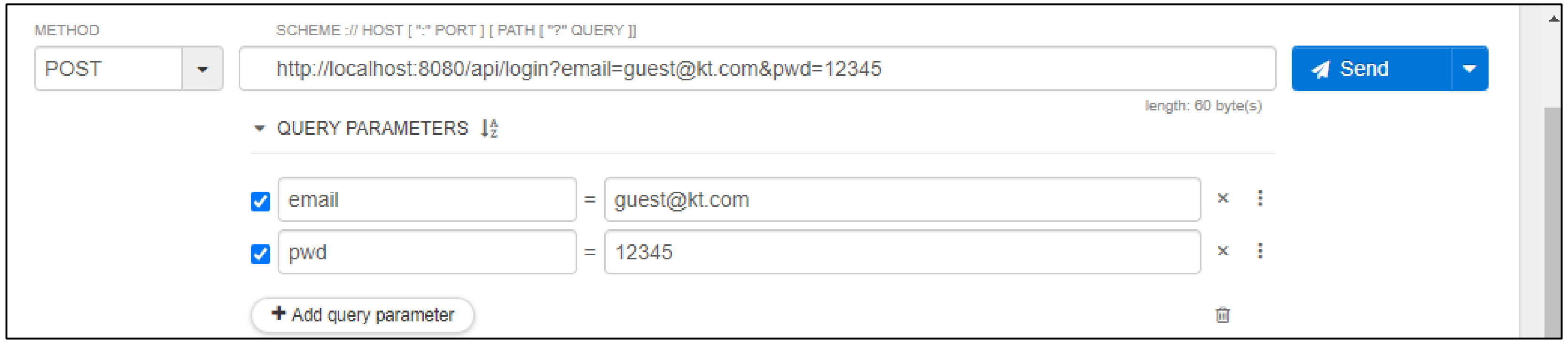
```java
    @Test
    public void testDeleteMemberByEmail() {
        // Given
        String email = "guest@kt.com";

        // When
        clubMemberRepository.deleteById(email);

        // Then
        Optional<ClubMember> result = clubMemberRepository.findById(email);
        assertThat(result).isNotPresent(); // 삭제 후에는 존재하지 않아야 함

        // 확인용 출력
        if (result.isEmpty()) {
            System.out.println("Member with email " + email + " has been successfully deleted.");
        }
    }
}
```



```
src/test/java
  com.edu
  com.edu.repository
    ClubMemberRepositoryTest.java
    NoteTest.java
```

인증 처리 – AuthenticationManager

```java
public class ApiLoginFilter extends AbstractAuthenticationProcessingFilter {

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

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

          String email = request.getParameter("email");
          String pw = request.getParameter("pwd");

          System.out.println(email+"/"+pwd);

          UsernamePasswordAuthenticationToken authToken = new UsernamePasswordAuthenticationToken(email, pwd);

          return getAuthenticationManager().authenticate(authToken);

     }

}
```



POST

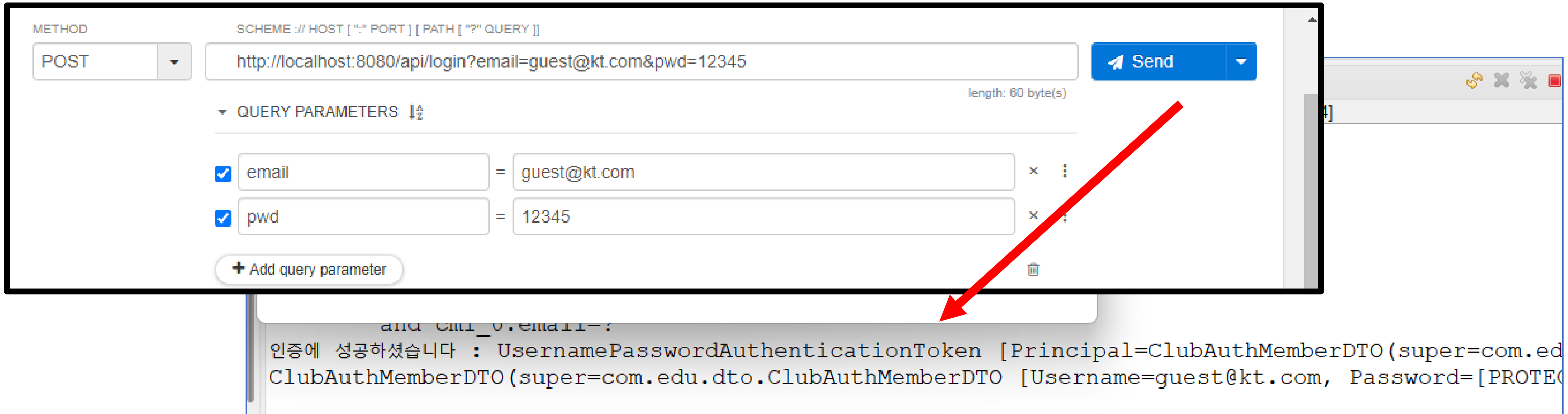
http://localhost:8080/api/login?email=guest@kt.com&pwd=12345

length: 60 byte(s)

✓	email	=	guest@kt.com
✓	pwd	=	12345

인증 성공 후 처리

```
package com.edu.security;

public class ApiLoginFilter extends AbstractAuthenticationProcessingFilter {

    ~ 생략 ~
    @Override
    protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
                                                                     FilterChain chain,
        Authentication authResult) throws IOException, ServletException {
        System.out.println("인증에 성공하셨습니다 : " + authResult);
        System.out.println(authResult.getPrincipal());

    }
}
```



METHOD: POST

URL: http://localhost:8080/api/login?email=guest@kt.com&pwd=12345

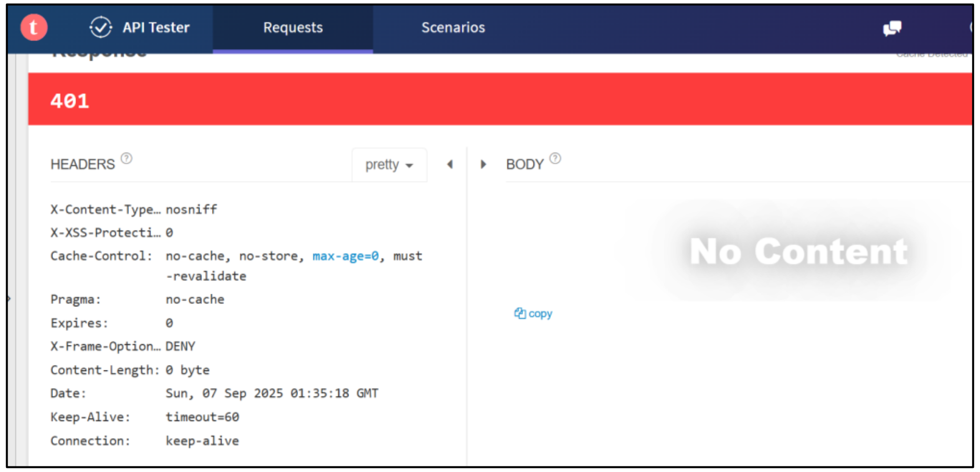
length: 60 byte(s)

email	=	guest@kt.com
pwd	=	12345

```
     and cm1_0.email=?
인증에 성공하셨습니다 : UsernamePasswordAuthenticationToken [Principal=ClubAuthMemberDTO(super=com.ed
ClubAuthMemberDTO(super=com.edu.dto.ClubAuthMemberDTO [Username=guest@kt.com, Password=[PROTE
```

인증 실패 후 처리

```java
package com.edu.security;

public class ApiLoginFilter extends AbstractAuthenticationProcessingFilter {

    ~ 생 략 ~

    @Override
    protected void unsuccessfulAuthentication(HttpServletRequest request,
                                              HttpServletResponse response, AuthenticationException failed) {

        response.setStatus(401);
    }
}
```



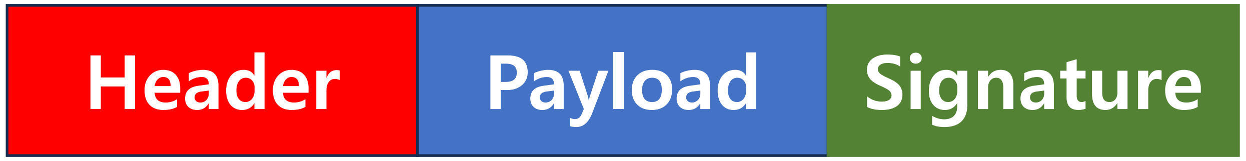
401

```
X-Content-Type… nosniff
X-XSS-Protecti… 0
Cache-Control:  no-cache, no-store, max-age=0, must
                -revalidate
Pragma:         no-cache
Expires:        0
X-Frame-Option… DENY
Content-Length: 0 byte
Date:           Sun, 07 Sep 2025 01:35:18 GMT
Keep-Alive:     timeout=60
Connection:     keep-alive
```

JWT(JSON Web Token)

- JWT란?
  - JSON Web Token(JWT)은 JSON 객체를 사용하여 두 당사자 간에 정보를 안전하게 전송하기 위한 개방형 표준(RFC 7519)이다.
  - 토큰은 자체적으로 정보를 포함하고 있으며, 일반적으로 인증 및 정보 교환에 사용된다.
- JWT의 특징
  - **자가 포함(Self-contained)**: JWT는 필요한 모든 정보를 자체적으로 포함하고 있어 추가적인 데이터베이스 조회가 불필요할 수 있다.
  - **경량(Lightweight)**: JSON 형식으로 인코딩되어 있어 비교적 크기가 작으며, URL, HTTP 헤더, 쿠키 등 다양한 장소에서 사용될 수 있다.
  - **디지털 서명**: JWT는 보안성을 위해 HMAC 또는 RSA 알고리즘을 사용하여 서명된다.

JWT 기본 구조

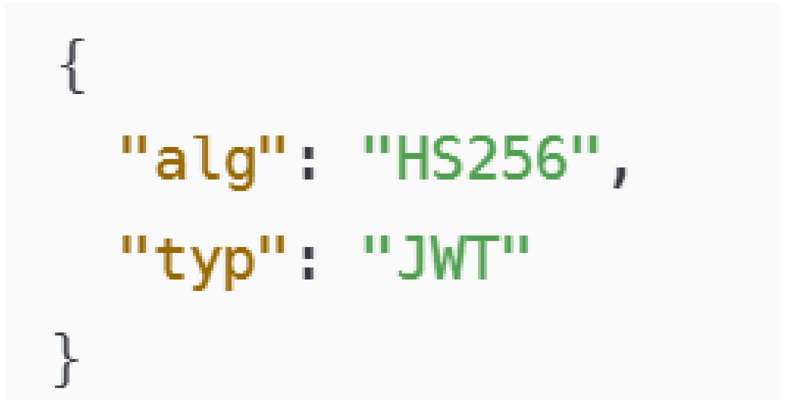


- **Header** : 토큰 유형, 서명 알고리즘 정보, Base64Url 인코딩(JSON)
- **Payload** : 사용자 정보 및 Claims 저장, Base64Url 인코딩(JSON)
- **Signature** : 토큰 위·변조 여부 검증, Header + Payload를 비밀키로 서명
- **Header.Payload.Signature** : 3개의 문자열을 **"."(점)** 으로 구분

```
`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ik
pvaG4gRG9lIiwiYWRtaW4iOnRydWV9.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`
```

Header

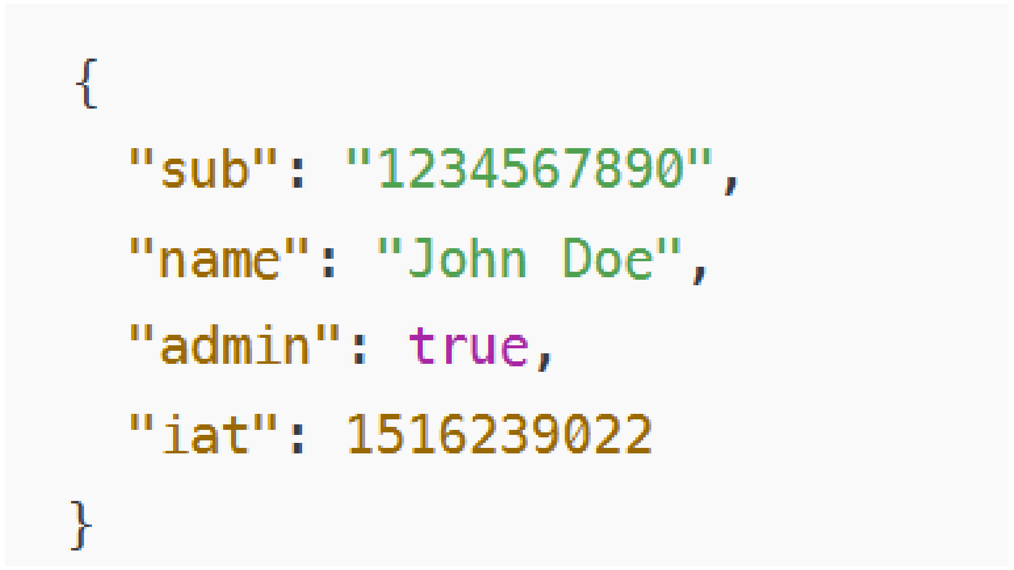
- 토큰의 메타데이터(metadata) 영역
- 토큰 유형(type)과 서명 알고리즘(alg) 정보 포함
- JSON 객체 형태이며, Base64Url 방식으로 인코딩됨



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

Payload

- JWT의 본문(body) 영역
- 사용자 정보와 토큰에 포함될 데이터 저장.
- Base64Url 방식으로 인코딩된 JSON 객체.
- 내부에 다양한 Claims가 포함됨



```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true,
  "iat": 1516239022
}
```

Claims

- Payload에 포함된 개별 데이터(속성)
- 사용자 정보나 토큰 관련 내용을 나타냄
- 표준으로 예약된 키워드들

키워드	설　　명
iss	토큰 발급자 (issuer)
sub	토큰 주제 (subject, 사용자 식별자)
aud	토큰 대상자 (audience)
exp	만료 시간 (expiration)
nbf	사용 가능 시작 시간 (not before)
iat	발급 시간 (issued at)
jti	JWT ID (토큰 고유 식별자)

Signature

- header와 payload를 비밀키로 서명한 값 → 토큰 위변조 방지
- 서명(signature) 생성 방식

```
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  secretKey
)
```

1. heade와 payload를 각각 Base64Url로 인코딩
2. 두 문자열을 . 으로 연결
3. secretKey로 HMAC-SHA256 같은 알로리즘 적용
4. 위의 값을 Base64Url 인코딩

(1) Header

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

→ Base64Url → eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

(2) Payload

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
}
```

→ Base64Url → eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9

(3) Signature

서명 알고리즘: HMAC-SHA256 , 비밀키: your-256-bit-secret

```
HMACSHA256(
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9", "your-256-bit-secret"
)
```

→ Base64Url → **TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ**

JWT 활용

인증(Authentication)

- 사용자가 로그인하면 서버는 사용자 정보를 기반으로 JWT를 생성하고 이를 클라이언트에게 전달한다.
- 클라이언트는 이후의 모든 요청에서 JWT를 HTTP헤더에 포함시켜 서버에 전송하며, 서버는 이 토큰을 검증하여 사용자의 신원을 확인한다.

정보 교환

- JWT는 클라이언트와 서버간에 정보를 안전하게 교환하는데 사용할 수 있다.
- 자가 포함된 구조로 인해 서버는 별도의 DB 조회 없이도 JWT에 포함된 정보를 신뢰할 수 있다.

JWT와 세션 기반 인증 비교

세션 기반 인증

- 서버가 사용자 세션을 유지하며, 클라이언트는 세션 ID를 서버에 전송해 인증을 받는다.
- 서버에 세션을 유지하기 위한 추가적인 메모리와 리소스가 필요하다.

JWT 기반 인증

- 서버가 상태를 저장하지 않고, 클라이언트가 JWT를 통해 인증을 수행한다.
- 서버에 별도의 상태 저장이 필요하지 않아 확장성이 뛰어나다

JWT의 보안 고려 사항

- 서명 검증

  서버는 클라이언트가 제공한 JWT의 서명을 반드시 검증해야 한다. 이를 통해 토큰이 변조되지 않았음을 확인할 수 있다.

- 비밀키 관리

  HMAC 알고리즘을 사용하는 경우, 비밀키는 안전하게 관리되어야 한다. 비밀키가 노출되면 누구나 토큰을 발급할 수 있게 되어 주의가 필요하다.

- 토큰 만료 처리

  JWT는 만료 시간을 지정할 수 있으며, 서버는 만료된 토큰에 대해 적절히 처리해야 한다. 예를 들어 만료된 토큰을 수락하지 않고 새로운 토큰을 요구할 수 있다.

- 정보의 노출

  JWT의 Payload는 Base64로 인코딩되기 때문에 누구나 디코딩하여 내용을 확인할 수 있다. 따라서 민감한 정보를 토큰에 포함시키지 않아야 한다.

JWT 발급 및 검증

1. 사용자가 로그인을 하면 JWT 발급
2. 사용자는 JWT 토큰을 가지고 모든 경로의 자원에 접근
3. 접근 시 JWT의 유효성을 검증
4. JWT가 서버에서 생성된 것인지 확인
5. JWT의 유효기간 검증
6. JWT 발급과 검증 작업을 수행하는 클래스 필요

JWT 라이브러리

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.12.6</version>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.12.6</version>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.12.6</version>
</dependency>
```

토큰 발급 – application.properties

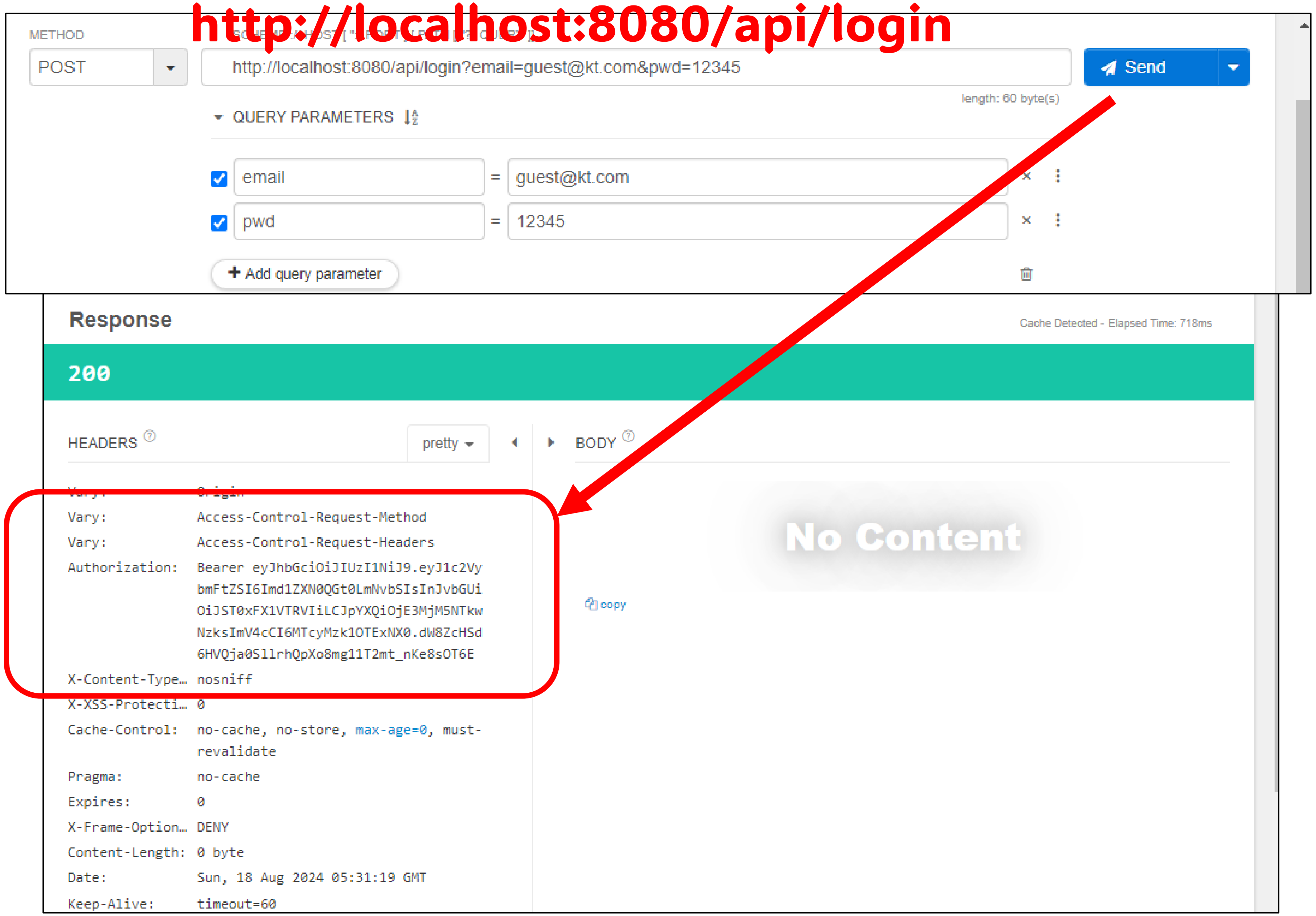
```
# JWT (JSON Web Token)에서 사용할 비밀 키 (Secret Key)를 설정합니다.
# 이 키는 토큰의 서명(Signature)을 생성하고 검증하는 데 사용됩니다.
# HMAC-SHA 알고리즘 (예: HmacSHA256)을 사용하여 JWT 서명을 생성하기 위해 최소 256비트(32바이트) 길이의 키가 필요합니다.
# 이 키는 애플리케이션의 보안을 위해 외부로 노출되지 않도록 주의해야 합니다.
# 또한, 실무 환경에서는 이 키를 환경 변수나 외부 설정 파일에 저장하는 것이 좋습니다.
jwt.secretKey=sPq3RQpdU2t6p1JgZbL9jt8M4E+QX1R9OFWtBFA1d5M=
```

```java
package com.edu.jwt;

import java.util.Date;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;
import javax.crypto.SecretKey;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class JWTUtil {

private final SecretKey secretKey;

     public JWTUtil(@Value("${jwt.secretKey}") String secretKeyString) throws Exception {
          this.secretKey = Keys.hmacShaKeyFor(secretKeyString.getBytes("UTF-8"));
     }

     public String generateToken(String username, String role, Long expiredMs) {

          return Jwts.builder()
                         .claim("username", username)
                         .claim("role", role)
                         .issuedAt(new Date(System.currentTimeMillis()))
                         .expiration(new Date(System.currentTimeMillis() + expiredMs)).signWith(secretKey).compact();
     }

}
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private final AuthenticationConfiguration configuration;
    private final JWTUtil jwtUtil;

    public WebSecurityConfig(AuthenticationConfiguration configuration, JWTUtil jwtUtil) {
        this.configuration = configuration;
        this.jwtUtil = jwtUtil;
    }

    @Bean
    public ApiCheckFilter apiCheckFilter() {
        return new ApiCheckFilter("/notes/**/*");
    }

    @Bean
    public ApiLoginFilter apiLoginFilter() throws Exception {
        ApiLoginFilter apiLoginFilter = new ApiLoginFilter("/api/login", jwtUtil);
        apiLoginFilter.setAuthenticationManager(configuration.getAuthenticationManager());
        apiLoginFilter.setAuthenticationFailureHandler(new ApiLoginFailHandler());
        return apiLoginFilter;
    }
    ~ 생략 ~
```

```java
public class ApiLoginFilter extends AbstractAuthenticationProcessingFilter {

     private final JWTUtil jwtUtil;

     public ApiLoginFilter(String defaultFilterProcessesUrl, JWTUtil jwtUtil) {
          super(new AntPathRequestMatcher(defaultFilterProcessesUrl));
          this.jwtUtil = jwtUtil;
     }
     ~ 생략 ~
     @Override
     protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,
                                                   Authentication authResult) throws IOException, ServletException {
          System.out.println("인증에 성공하셨습니다 : " + authResult);
          System.out.println(authResult.getPrincipal());

          // 토큰 발급
          ClubAuthMemberDTO authMember = (ClubAuthMemberDTO)authResult.getPrincipal();
          String email = authMember.getEmail();  // 이메일 추출
          Collection<? extends GrantedAuthority> authorities = authMember.getAuthorities();     // Role 추출
          Iterator<? extends GrantedAuthority> iterator = authorities.iterator();
          GrantedAuthority auth = iterator.next();
          String role = auth.getAuthority();

          String token = jwtUtil.generateToken(email, role, 60*60*10L);
          response.addHeader("Authorization", "Bearer " + token);
     }
}
```

http://localhost:8080/api/login



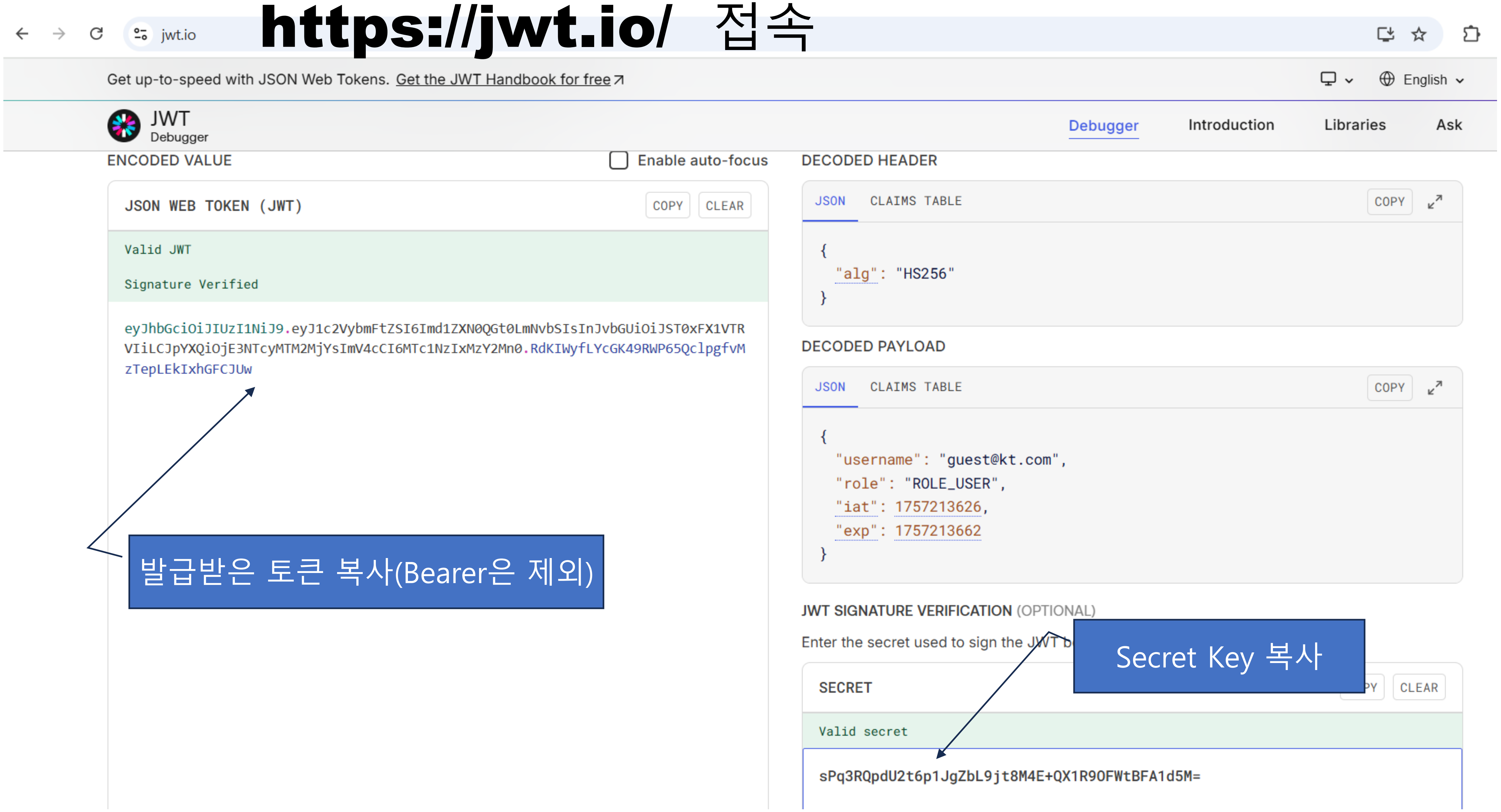
```
http://localhost:8080/api/login?email=guest@kt.com&pwd=12345
```

email	guest@kt.com
pwd	12345

200

```
Vary:             Origin
Vary:             Access-Control-Request-Method
Vary:             Access-Control-Request-Headers
Authorization:    Bearer eyJhbGciOiJIUzI1NiJ9.eyJ1c2Vy
                  bmFtZSI6Imd1ZXN0QGt0LmNvbSIsInJvbGUi
                  OiJST0xFX1VTRVIiLCJpYXQiOjE3MjM5NTkw
                  NzksImV4cCI6MTcyMzk1OTExNX0.dW8ZcHSd
                  6HVQja0S1lrhQpXo8mg11T2mt_nKe8sOT6E
X-Content-Type…   nosniff
X-XSS-Protecti…   0
Cache-Control:    no-cache, no-store, max-age=0, must-
                  revalidate
Pragma:           no-cache
Expires:          0
X-Frame-Option…   DENY
Content-Length:   0 byte
Date:             Sun, 18 Aug 2024 05:31:19 GMT
Keep-Alive:       timeout=60
```

https://jwt.io/ 접속



jwt.io
Get up-to-speed with JSON Web Tokens. Get the JWT Handbook for free
English
JWT Debugger
Debugger
Introduction
Libraries
Ask

ENCODED VALUE

Enable auto-focus

JSON WEB TOKEN (JWT) COPY CLEAR

```
Valid JWT

Signature Verified

eyJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Imd1ZXN0QGt0LmNvbSIsInJvbGUiOiJST0xFX1VTR
VIiLCJpYXQiOjE3NTcyMTM2MjYsImV4cCI6MTc1NzIxMzY2Mn0.RdKIWyfLYcGK49RWP65QclpgfvM
zTepLEkIxhGFCJUw
```

발급받은 토큰 복사(Bearer은 제외)

DECODED HEADER

JSON CLAIMS TABLE COPY

```
{
  "alg": "HS256"
}
```

DECODED PAYLOAD

JSON CLAIMS TABLE COPY

```
{
  "username": "guest@kt.com",
  "role": "ROLE_USER",
  "iat": 1757213626,
  "exp": 1757213662
}
```

JWT SIGNATURE VERIFICATION (OPTIONAL)

Enter the secret used to sign the JWT b

Secret Key 복사

SECRET PY CLEAR

```
Valid secret

sPq3RQpdU2t6p1JgZbL9jt8M4E+QX1R9OFWtBFA1d5M=
```

토큰 검증

```
@Component
public class JWTUtil {
    ~ 생 략 ~
   public String getUsername(String token) {

        return Jwts.parser().verifyWith(secretKey).build()
                .parseSignedClaims(token)
                .getPayload()
                .get("username", String.class);
    }

    public String getRole(String token) {

        return Jwts.parser().verifyWith(secretKey).build()
            .parseSignedClaims(token)
            .getPayload()
            .get("role", String.class);
    }

    public Boolean isExpired(String token) {

        return Jwts.parser().verifyWith(secretKey).build()
            .parseSignedClaims(token)
            .getPayload()
            .getExpiration()
            .before(new Date());
    }
}
```

토큰 검증

```java
package com.edu.security;

import java.util.Collections;

public class ApiCheckFilter extends OncePerRequestFilter {

	private final AntPathMatcher antPathMatcher;
	private final String pattern;
	private final JWTUtil jwtUtil;

	public ApiCheckFilter(String pattern, JWTUtil jwtUtil) {
		this.antPathMatcher = new AntPathMatcher();
		this.pattern = pattern;
		this.jwtUtil = jwtUtil;
	}

	@Override
	protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
	                                                                                 throws ServletException, IOException {
		if (antPathMatcher.match(pattern, request.getRequestURI())) {
			boolean checkHeader = checkAuthHeader(request);
			if (checkHeader) {
				filterChain.doFilter(request, response);
				return;
			} else {
				sendErrorResponse(response, HttpServletResponse.SC_FORBIDDEN, "403", "FAIL CHECK API TOKEN");
				return;
			}
		}
		filterChain.doFilter(request, response);
	}
```

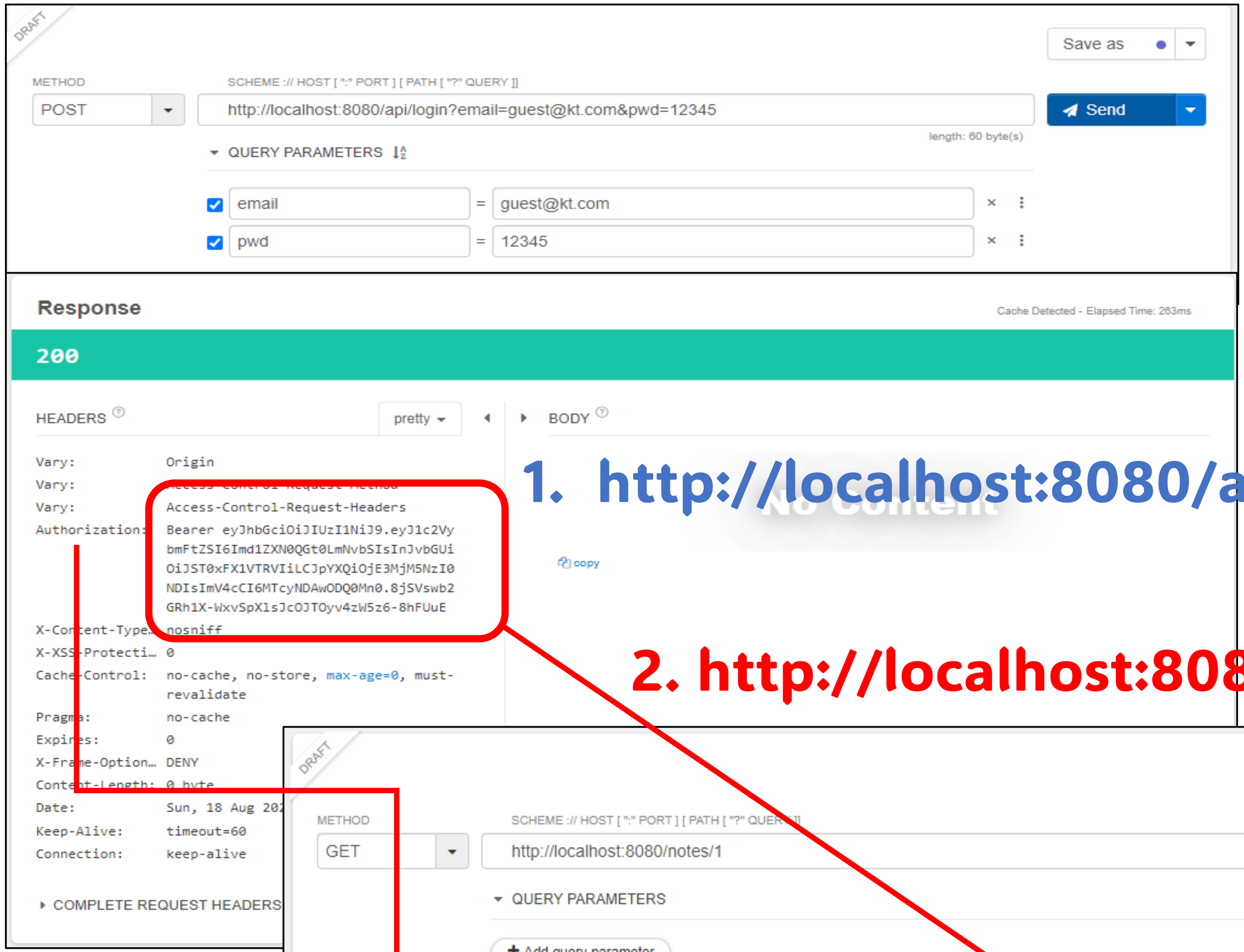
```java
private boolean checkAuthHeader(HttpServletRequest request) {

    boolean checkResult = false;
    String authorization = request.getHeader("Authorization");

    if (StringUtils.hasText(authorization)) {
        String token = authorization.split(" ")[1];
        if(jwtUtil.isExpired(token)) {
            System.out.println("token expired");
            return false;
        }
        String username = jwtUtil.getUsername(token);
        String role  = jwtUtil.getRole(token);

        SimpleGrantedAuthority authority = new SimpleGrantedAuthority(role);
        ClubAuthMemberDTO authMember =
                new ClubAuthMemberDTO(username, "password", Collections.singletonList(authority));

        Authentication authToken =
                new UsernamePasswordAuthenticationToken(authMember, null, authMember.getAuthorities());

        SecurityContextHolder.getContext().setAuthentication(authToken);

        checkResult = true;

    }else {   System.out.println("token null");   }

    return checkResult;
}
```

```java
    //오류 메시지 응답
    private void sendErrorResponse(HttpServletResponse response,
                                   int status, String code, String message)
                                                        throws IOException {
        response.setStatus(status);
        response.setContentType("application/json;charset=utf-8");

        Map<String, String> responseMap = new HashMap<>();
        responseMap.put("code", code);
        responseMap.put("message", message);

        ObjectMapper objectMapper = new ObjectMapper();
        String jsonResponse = objectMapper.writeValueAsString(responseMap);

        try (PrintWriter out = response.getWriter()) {
            out.print(jsonResponse);
        }
    }

}
```

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    private final AuthenticationConfiguration configuration;
    private final JWTUtil jwtUtil;

    public WebSecurityConfig(AuthenticationConfiguration configuration, JWTUtil jwtUtil) {
        this.configuration = configuration;
        this.jwtUtil = jwtUtil;
    }

    @Bean
    public ApiCheckFilter apiCheckFilter() {
        return new ApiCheckFilter("/notes/**/*", jwtUtil);
    }

    @Bean
    public ApiLoginFilter apiLoginFilter() throws Exception {
        ApiLoginFilter apiLoginFilter = new ApiLoginFilter("/api/login", jwtUtil);
        apiLoginFilter.setAuthenticationManager(configuration.getAuthenticationManager());
        return apiLoginFilter;
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
             .csrf(csrf -> csrf.disable())
             .formLogin(auth->auth.disable())
             .authorizeHttpRequests(authz -> authz
                                            .requestMatchers("/", "/notes/**","/api/login").permitAll()
                                            .anyRequest().authenticated() )

            .sessionManagement(session->session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

        http
            .addFilterBefore(apiCheckFilter(), UsernamePasswordAuthenticationFilter.class)
            .addFilterBefore(apiLoginFilter(), UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

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

1. http://localhost:8080/api/login 요청 후 토큰 복사

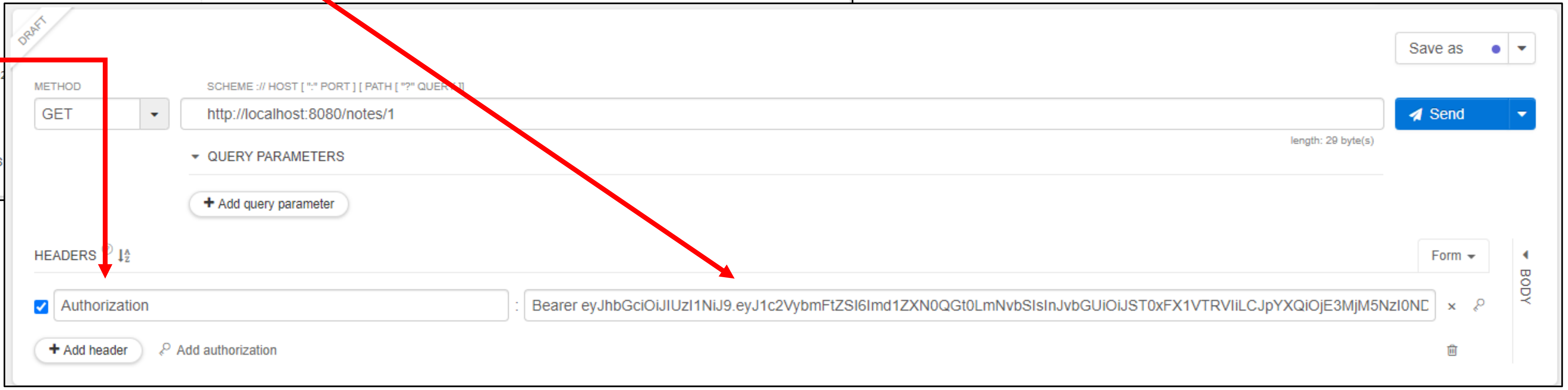
2. http://localhost:8080/notes/1 요청 시 Header 지정



POST	http://localhost:8080/api/login?email=guest@kt.com&pwd=12345	
✓	email	guest@kt.com
✓	pwd	12345

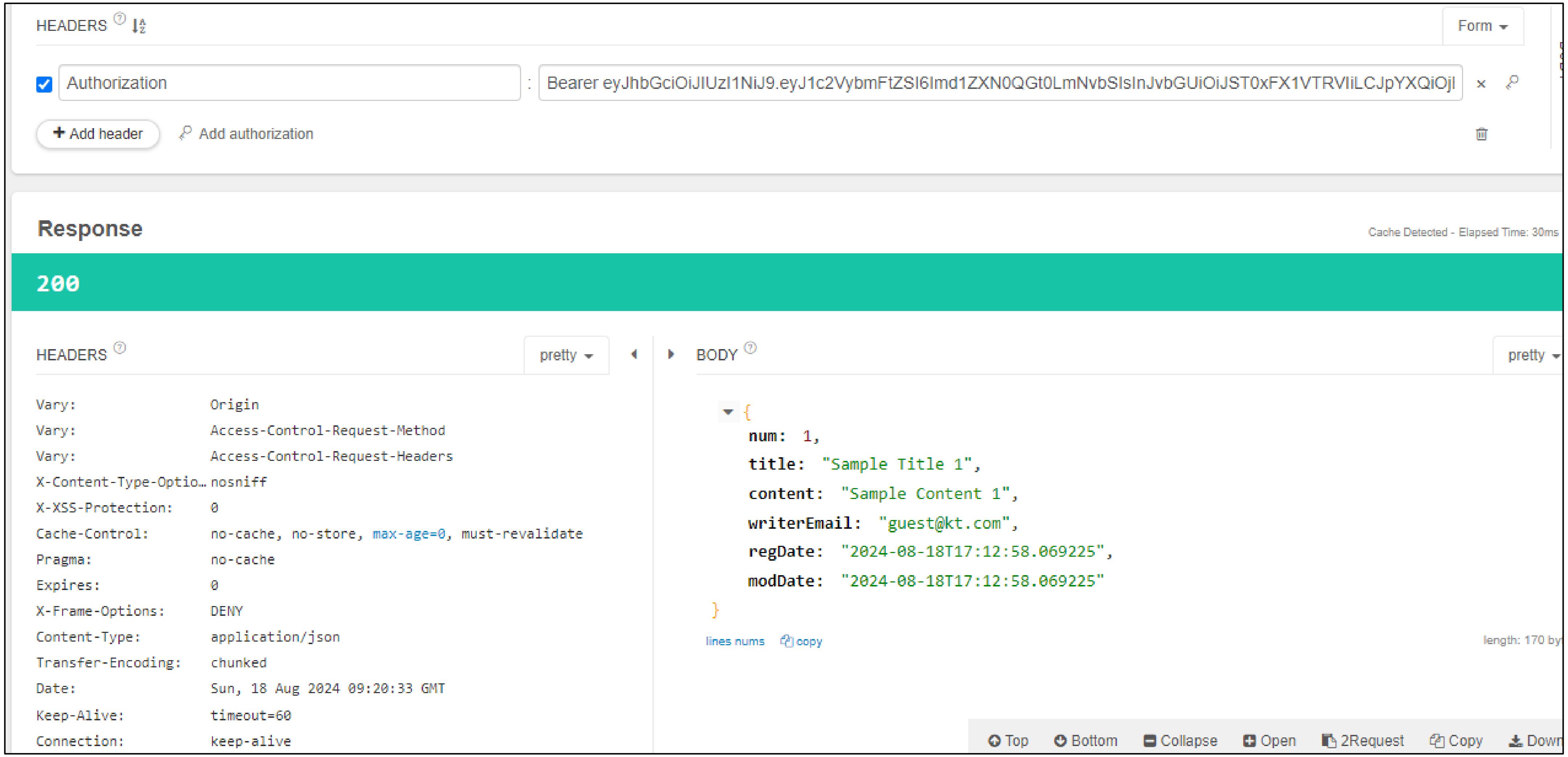
200

```
Vary:           Origin
Vary:           Access-Control-Request-Method
Vary:           Access-Control-Request-Headers
Authorization:  Bearer eyJhbGciOiJIUzI1NiJ9.eyJ1c2Vy
                bmFtZSI6Imd1ZXN0QGt0LmNvbSIsInJvbGUi
                OiJST0xFX1VTRVIiLCJpYXQiOjE3MjM5NzI0
                NDIsImV4cCI6MTcyNDAwODQ0Mn0.8jSVswb2
                GRh1X-WxvSpXlsJcOJTOyv4zW5z6-8hFUuE
X-Content-Type… nosniff
X-XSS-Protecti… 0
Cache-Control:  no-cache, no-store, max-age=0, must-
                revalidate
Pragma:         no-cache
Expires:        0
X-Frame-Option… DENY
Content-Length: 0 byte
Date:           Sun, 18 Aug 202
Keep-Alive:     timeout=60
Connection:     keep-alive
```



GET	http://localhost:8080/notes/1	
✓	Authorization	Bearer eyJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Imd1ZXN0QGt0LmNvbSIsInJvbGUiOiJST0xFX1VTRVIiLCJpYXQiOjE3MjM5NzI0ND

JWT 인증 성공 후 응답 정보



HEADERS

Form

Authorization : Bearer eyJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Imd1ZXN0QGt0LmNvbSIsInJvbGUiOiJST0xFX1VTRVIiLCJpYXQiOjI

Add header   Add authorization

Response

Cache Detected - Elapsed Time: 30ms

200

HEADERS

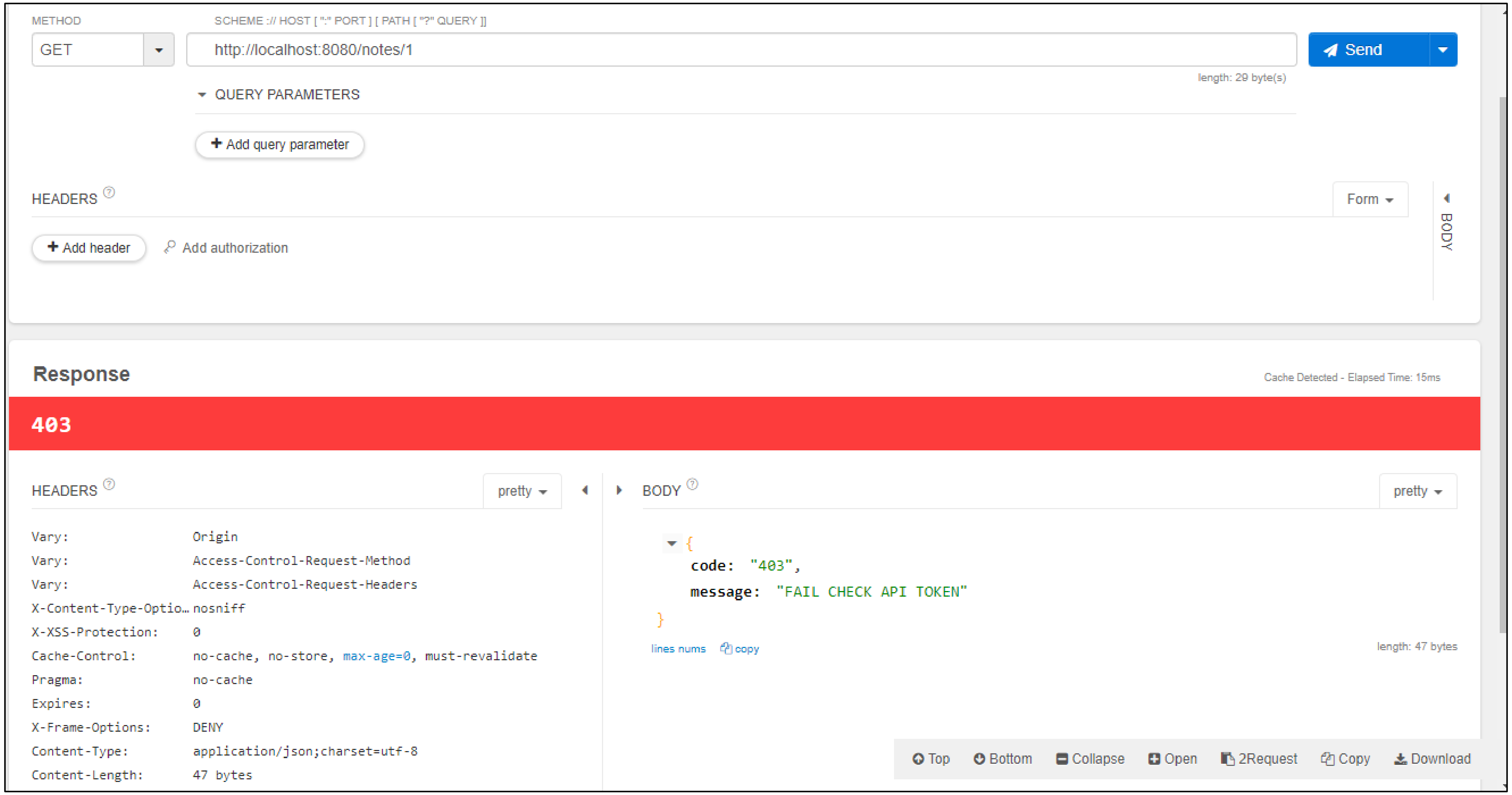
```
Vary:                    Origin
Vary:                    Access-Control-Request-Method
Vary:                    Access-Control-Request-Headers
X-Content-Type-Optio… nosniff
X-XSS-Protection:        0
Cache-Control:           no-cache, no-store, max-age=0, must-revalidate
Pragma:                  no-cache
Expires:                 0
X-Frame-Options:         DENY
Content-Type:            application/json
Transfer-Encoding:       chunked
Date:                    Sun, 18 Aug 2024 09:20:33 GMT
Keep-Alive:              timeout=60
Connection:              keep-alive
```

BODY

```
{
  num:  1,
  title:  "Sample Title 1",
  content:  "Sample Content 1",
  writerEmail:  "guest@kt.com",
  regDate:  "2024-08-18T17:12:58.069225",
  modDate:  "2024-08-18T17:12:58.069225"
}
```

length: 170 by

JWT 인증 실패 후 응답 정보


METHOD
GET

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]
http://localhost:8080/notes/1

length: 29 byte(s)

QUERY PARAMETERS

HEADERS

Response

Cache Detected - Elapsed Time: 15ms

403

HEADERS

```
Vary:                   Origin
Vary:                   Access-Control-Request-Method
Vary:                   Access-Control-Request-Headers
X-Content-Type-Optio… nosniff
X-XSS-Protection:       0
Cache-Control:          no-cache, no-store, max-age=0, must-revalidate
Pragma:                 no-cache
Expires:                0
X-Frame-Options:        DENY
Content-Type:           application/json;charset=utf-8
Content-Length:         47 bytes
```

BODY

```
{
  code:  "403",
  message:  "FAIL CHECK API TOKEN"
}
```

length: 47 bytes