API 서버

ApiController 생성

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

    //Controller 계층에서는 service계층만 봐야함
    //      조합, 가공, 연산 진행
    private final BoardService boardService;

    @GetMapping("/board/list")
    public PageResponseDTO<BoardDTO> getList(@RequestBody PageRequestDTO
pageRequestDTO) {
        //pageRequestDTO가 JSON형식으로 들어오니까 RequestBody 걸어줘야함
        log.info("pageRequestDTO: "+pageRequestDTO);

        return boardService.getList(pageRequestDTO);
    }

}
```

결과

서버 실행하면 Error 발생함 => RequestBody의 단점

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing

Fri Oct 22 09:58:23 KST 2021
There was an unexpected error (type=Bad Request, status=400).
Required request body is missing: public org.zerock.sb.dto.PageResp
org.springframework.http.converter.HttpMessageNotReadableExcepti
org.zerock.sb.controller.ApiController.getList(org.zerock.sb.dto.PageRe

parameter가 없어도 처리되어야하니까 일단 *@RequestBody* 제거

제거하면 json data 확인 가능

localhost:8080/api/board/list


```
{"page":1,"size":10,"count":201,"start":1,"end":10,"prev":false,"next":true,"dtoList":[{"bno":
{"bno":200,"writer":"user0","title":"Title...200","content":"Content...200","regDate":"2021-1(
{"bno":199,"writer":"user9","title":"Title...199","content":"Content...199","regDate":"2021-1(
{"bno":198,"writer":"user8","title":"Title...198","content":"Content...198","regDate":"2021-1(
{"bno":197,"writer":"user7","title":"Title...197","content":"Content...197","regDate":"2021-1(
{"bno":196,"writer":"user6","title":"Title...196","content":"Content...196","regDate":"2021-1(
{"bno":195,"writer":"user5","title":"Title...195","content":"Content...195","regDate":"2021-1(
{"bno":194,"writer":"user4","title":"Title...194","content":"Content...194","regDate":"2021-1(
{"bno":193,"writer":"user3","title":"Title...193","content":"Content...193","regDate":"2021-1(
{"bno":192,"writer":"user2","title":"Title...192","content":"Content...192","regDate":"2021-1(
```


Postman 사용해서 test

GET http://localhost:8080/api/board/list

Params Authorization Headers (7) Body Pre-request Script Tes

Query Params

KEY	VALUE

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON


```
    "start": 1,
    "end": 10,
    "prev": false,
    "next": true,
    "dtoList": [
        {
            "bno": 201,
            "writer": "user00",
            "title": "한글제목",
            "content": "한글내용",
            "regDate": "2021-10-07T11:57:19.113836",
            "modDate": "2021-10-07T12:30:49.510745"
        },
        {
            "bno": 200,
            "writer": "user0",
            "title": "Title...200",
            "content": "Content...200",
            "regDate": "2021-10-07T09:48:24.837427",
```


GET 방식으로 데이터 전송 시 받아와지는 것 확인 가능

외부 환경에서도 접속 가능한지 check

현재는 token 없이 direct로 접속

filter 설정해야함

servlet 할 때 쓴 필터 써도되지만 시큐리티에서 제공하는 filter도 있음

TokenCheckFilter 생성

```
@Log4j2
public class TokenCheckFilter extends OncePerRequestFilter {

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

        //끝
        log.info("========== TokenCheckFilter ==========");
        //정상적으로 왔으니 다음단계로 진행시키는 기능
        //문제가 생길 경우 여기로 연결시키면 안되고 튕겨내야함 - JSON Object 사용
        filterChain.doFilter(request, response);
    }
}
```

api라는 경로로 접근할 때만 동작하도록 설정 로그인때는 xXX

사용자 인증 직전에 이 필터를 거치도록 설계할 것

CustomSecurityConfig 수정

```
public class CustomSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        //사용자가 로그인 하기 전에 사용할 것이라고 명시시
       http.addFilterBefore(tokenCheckFilter(),
UsernamePasswordAuthenticationFilter.class);

    }

    @Bean
    public TokenCheckFilter tokenCheckFilter() {
        return new TokenCheckFilter();
    }
}
```

서버 실행

```
: Initializing Spring DispatcherServlet 'dispatc
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 2 ms
: ---------- TokenCheckFilter ----------
: ---------- TokenCheckFilter ----------
: ========== TokenCheckFilter ==========
: pageRequestDTO: PageRequestDTO(page=1, size=10
: --------------search1
```

필터 추가

TokenCheckFilter 수정

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

    String path = request.getRequestURI();  //어디서 호출하는지
    log.info(path);

    if (path.startsWith("/api/")) {
        //api로 들어오면 check token
        //공식 HTTP Authorization token 사용

    } else {

        log.info("========== TokenCheckFilter ==========");
        //정상적으로 왔으니 다음단계로 진행시키는 기능
        //문제가 생길 경우 여기로 연결시키면 안되고 튕겨내야함 - JSON Object 사용
        filterChain.doFilter(request, response);
    }

}
```

링크 확인

```
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 1 ms
: ---------- TokenCheckFilter ----------
: ---------- TokenCheckFilter ----------
: /api/board/list
: ========== TokenCheckFilter ==========
: pageRequestDTO: PageRequestDTO(page=1, siz
: --------------search1
```

build.gradle 추가

```
// json
implementation group: 'org.json', name: 'json', version: '20210307'


implementation 'io.jsonwebtoken:jjwt-api:0.11.2'
runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.2','io.jsonwebtoken:jjwt-
jackson:0.11.2'
```

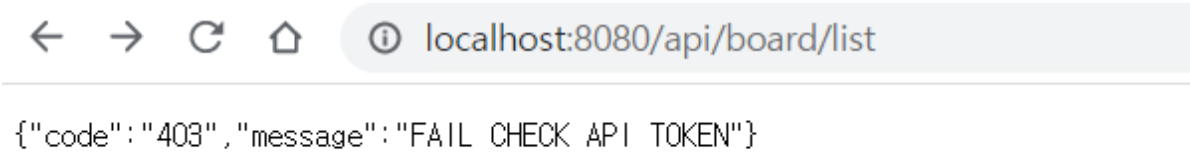
만약 문제생기면 바로 메세지 보낼 수 있도록 설정

TokenCheckFilter 수정

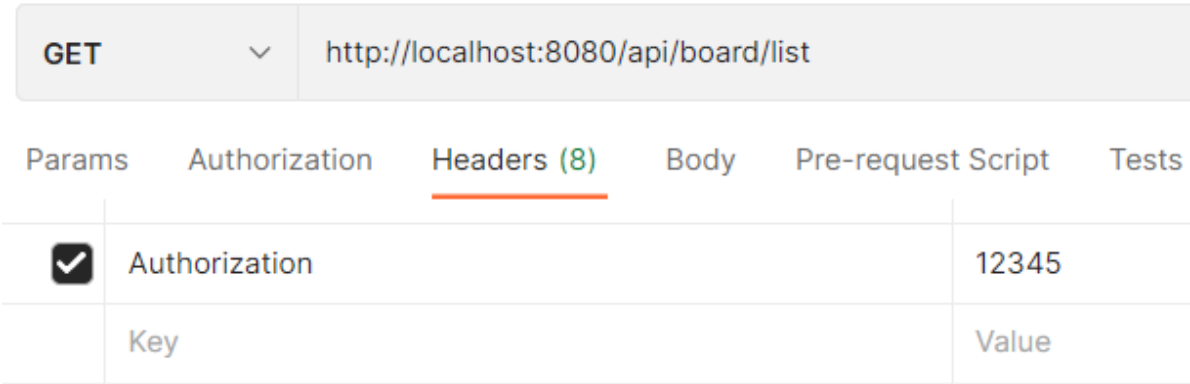
```
if (path.startsWith("/api/")) {

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

    if (authToken == null) {
        //이게 null이면 토큰 발급받고 오라고 보내야함
        log.info("authToken is null............");

        //1. 여기서 메세지 만들어서 보내기vv / 2. Forward 이용해서 보내기
        response.setStatus(HttpServletResponse.SC_FORBIDDEN);
        // json 리턴
        response.setContentType("application/json;charset=utf-8");
        JSONObject json = new JSONObject();
        String message = "FAIL CHECK API TOKEN";
        json.put("code", "403");
        json.put("message", message);

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

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

} else {
    ...
}
```

그냥 `/api/board/list` 접속하면 403 error 발생



postman에서 header 주고 test하면 통과됨



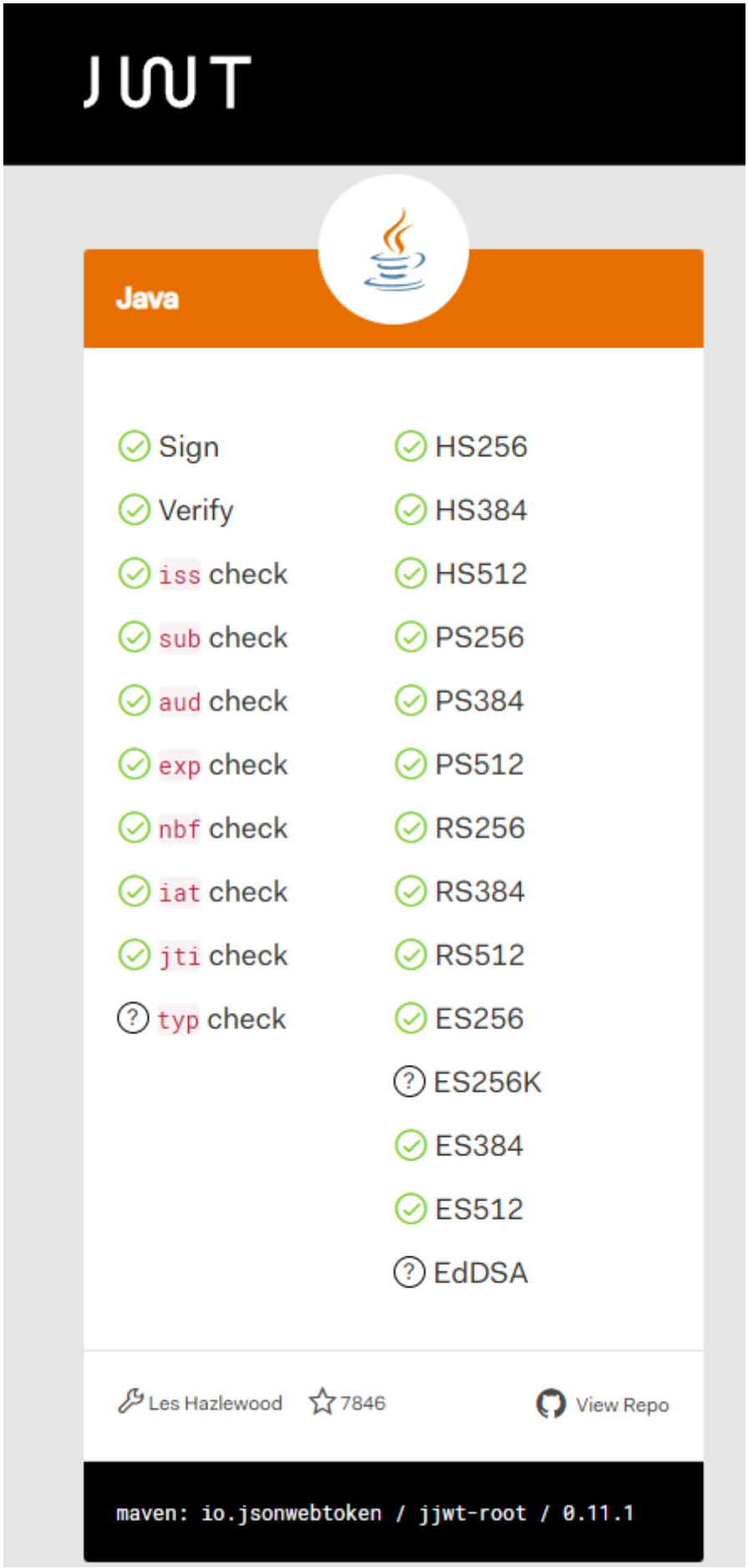
/api 로 들어오면 token 검증 (Authorization)

문제있을 경우 (null) - out.close, return

토큰의 유무 검증이 완료됐으므로 토큰이 유효한지 체크해야함

JWT

사이트에서 JWT가 제대로 만들어졌는지 체크 가능


JWT
Java
Sign
Verify
iss check
sub check
aud check
exp check
nbf check
iat check
jti check
typ check
HS256
HS384
HS512
PS256
PS384
PS512
RS256
RS384
RS512
ES256
ES256K
ES384
ES512
EdDSA
Les Hazlewood
7846
View Repo
maven: io.jsonwebtoken / jjwt-root / 0.11.1

우리가 사용할 토큰!

JWTUtil 생성

```
@Log4j2
public class JWTUtil {

    //1. generate 작업 필요
    public String generateToken(String content) {


        return null;
    }

    //2. JWT check
    public void validateToken(String token) throws JwtException {


    }

}
```

Config 걸어줘야함

```
@Bean
public TokenCheckFilter tokenCheckFilter() {
    //토큰을 체크하도록 넣어줘야함
    return new TokenCheckFilter(jwtUtil());  //넣어주면서 생성자 수정해야함
}

@Bean
public JWTUtil jwtUtil () {
    return new JWTUtil();
}
```

TokenCheckFilter 수정

```
    private JWTUtil jwtUtil;

    public TokenCheckFilter(JWTUtil jwtUtil) {
        this.jwtUtil = jwtUtil;
    }

...

    if (authToken == null) {

    }
```

```
    //jwt 검사
    jwtUtil.validateToken(authToken);
    //검사를 했는데 예외발생? -> 토큰에 문제 있다는 것 =>나중에 메세징처리

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

JWTUtil 수정

```
private final static String secretKey =
"helloworld1111122222333333333444444444444555555555";

private SecretKey key;

public JWTUtil() {
        key = Keys.hmacShaKeyFor(secretKey.getBytes(StandardCharsets.UTF_8));
}

    //1. generate 작업 필요
    public String generateToken(String content) {

        long timeAmount = 60;  //분단위

        String jws = Jwts.builder() // (1)
                .setSubject(content)      // (2)
                .setIssuedAt(new Date()) /*언제 발행?*/

.setExpiration(Date.from(ZonedDateTime.now().plusMinutes(timeAmount).toInstant()
)) /*언제까지? timeAmount - 유효기간*/
                .signWith(key, SignatureAlgorithm.HS256)          // (3) 키값 부여
                .compact();             // (4) 발행


        return jws;
    }
```

암호화 라이브러리 추가해줌

JWT 는 반드시 test 필수!!

```
@SpringBootTest
@Log4j2
public class JWTTests {

    @Autowired
    JWTUtil jwtUtil;

    @Test
    public void testGenerate() {
```

```
        String jwtStr = jwtUtil.generateToken("user11");

        log.info(jwtStr);
    }
}
```

결과

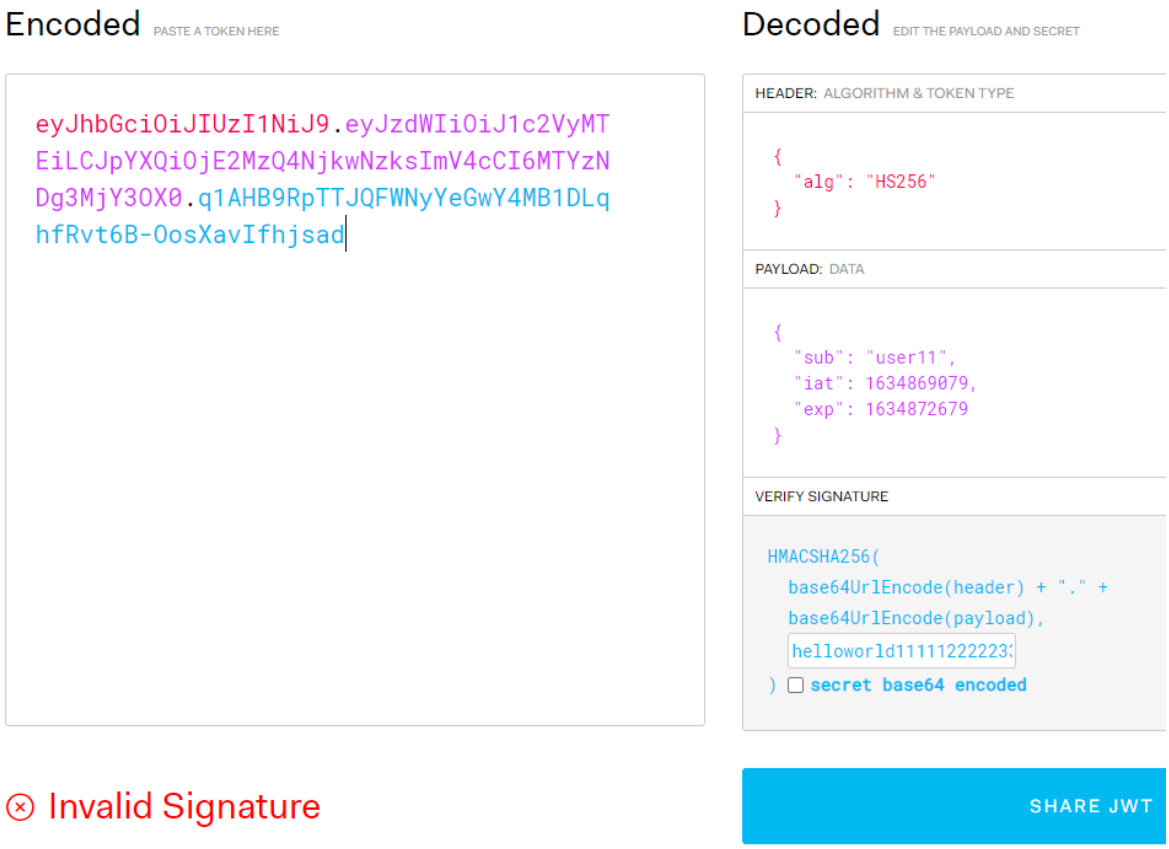
```
spring.jpa.open-in-view is enabled by default. Therefore, database
Adding welcome page: class path resource [static/index.html]
CustomSecurityConfig...configure..........
CustomSecurityConfig...configure..........
CustomSecurityConfig...configure..........
CustomSecurityConfig...configure..........
Will secure any request with [org.springframework.security.web.cont
Started JWTTests in 7.376 seconds (JVM running for 10.49)
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTEiLCJpYXQiOjE2MzQ4NjkwNzksIm
Closing JPA EntityManagerFactory for persistence unit 'default'
```

암호화된 JWT 생성됨

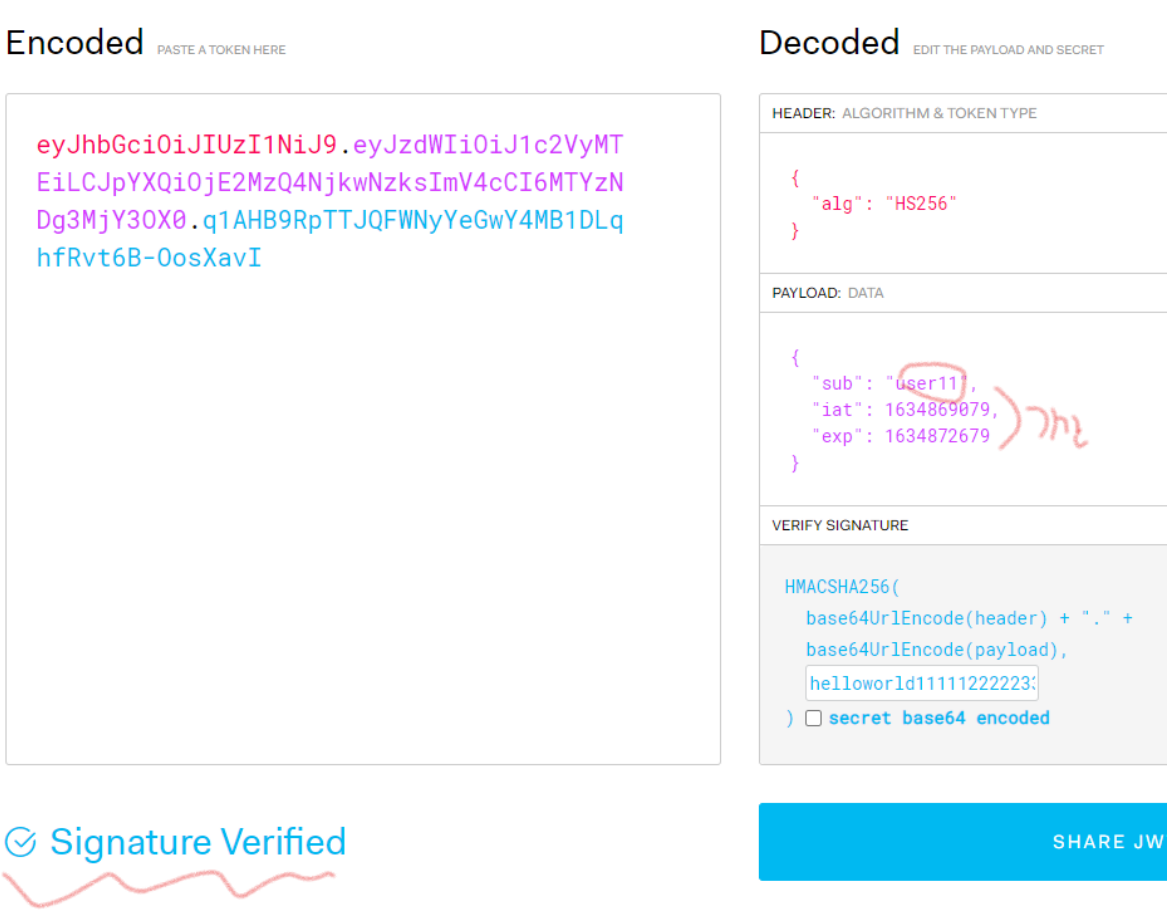
JWT 사이트에서 체크

```
key :
helloworld111112222233333333344444444444555555555

token :
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTEiLCJpYXQiOjE2MzQ4NjkwNzksImV4cCI6MTYzNDg
3MjY3OX0.q1AHB9RpTTJQFWNyYeGwY4MB1DLqhfRvt6B-OosXavI
```

맞지 않는 토큰값 입력 시 - Invalid



맞는 토큰값 입력 시



Signature Verified

JWTUtil 수정

```
//2. JWT check
public void validateToken(String token) throws JwtException {

    Jws<Claims> jws = Jwts.parserBuilder()  // (1)
            .setSigningKey(key)         // (2)
            .build()                    // (3)
            .parseClaimsJws(token); // (4)

}
```

test 진행

```
    @Test
    public void testValidate() {

 //eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTEiLCJpYXQiOjE2MzQ4NjkwNzksImV4cCI6MTYzNDg3MjY3OX0.q1AHB9RpTTJQFWNyYeGwY4MB1DLqhfRvt6B-OosXavI
        String str = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTEiLCJpYXQiOjE2MzQ4NjkwNzksImV4cCI6MTYzNDg3MjY3OX0.q1AHB9RpTTJQFWNyYeGwY4MB1DLqhfRvt6B-OosXavI";
        String wrongStr = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTEi2345678LCJpYXQiOjE2MzQ4NjkwNzksImV4cCI6MTYzNDg3MjY3OX0.q1AHB9RpTTJQFWNyYeGwY4MB1DLqhfRvt6B-OosXavI";
        String timeoutStr = "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTEiLCJpYXQiOjE2MzQ4NzA0NTgsImV4cCI6MTYzNDg3MDQ1OH0.R0g2mXzCMuqdWAz9K9IxHsEknZefrme7ke3sjTzOofI";
        //JWT 사이트에서는 맞다고 했지만 우리 메소드를 통해서도 검증이 되는지
        //잘못된, 만료된 토큰을 넣었을 때 제대로 결과가 나오는지 체크해야함
        try {
//              jwtUtil.validateToken(str);
            jwtUtil.validateToken(wrongStr);
//              jwtUtil.validateToken(timeoutStr);
        } catch (ExpiredJwtException ex) {
            log.info("expired.............");
            log.error(ex.getMessage());
        } catch (JwtException ex) {
            log.info("JWTException.............");
            log.error(ex.getMessage());
        }
    }
```

catch를 여러개 걸어서 어떤 예외를 발생시키는지 걸 수 있음

예외를 이용해서 각 예외마다 다른 결과 도출할 수도 있음

잘못된 토큰일 때

```
Will secure any request with [org.springframework.security.web.co
Started JWTTests in 7.542 seconds (JVM running for 10.751)
JWTException.............
JWT signature does not match locally computed signature. JWT vali
Closing JPA EntityManagerFactory for persistence unit 'default'
```

만료된 토큰일 때

```
Will secure any request with [org.springframework.security.web.co
Started JWTTests in 7.832 seconds (JVM running for 11.082)
expired.............
JWT expired at 2021-10-22T02:40:58Z. Current time: 2021-10-22T02:
```

필터추가하기

TokenGenerateFilter 생성

```
@Log4j2
public class TokenGenerateFilter extends AbstractAuthenticationProcessingFilter
{

    private JWTUtil jwtUtil;

    public TokenGenerateFilter(String defaultFilterProcessesUrl,
AuthenticationManager authenticationManager, JWTUtil jwtUtil) {
        //defaultFilterProcessesUrl - 로그인 경로
        super(defaultFilterProcessesUrl, authenticationManager);
        this.jwtUtil = jwtUtil;
    }

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

}
```

필터를 어디서 동작시킬지!

CustomSecurityConfig 수정

```
@Override
protected void configure(HttpSecurity http) throws Exception {

    //사용자가 로그인 하기 전에 사용할 것이라고 명시
   http.addFilterBefore(tokenCheckFilter(),
UsernamePasswordAuthenticationFilter.class);
   http.addFilterBefore(tokenGenerateFilter(),
UsernamePasswordAuthenticationFilter.class);

}

@Bean
public TokenGenerateFilter tokenGenerateFilter() throws Exception{
    return new TokenGenerateFilter("/jsonApiLogin", authenticationManager(),
jwtUtil());
}
```

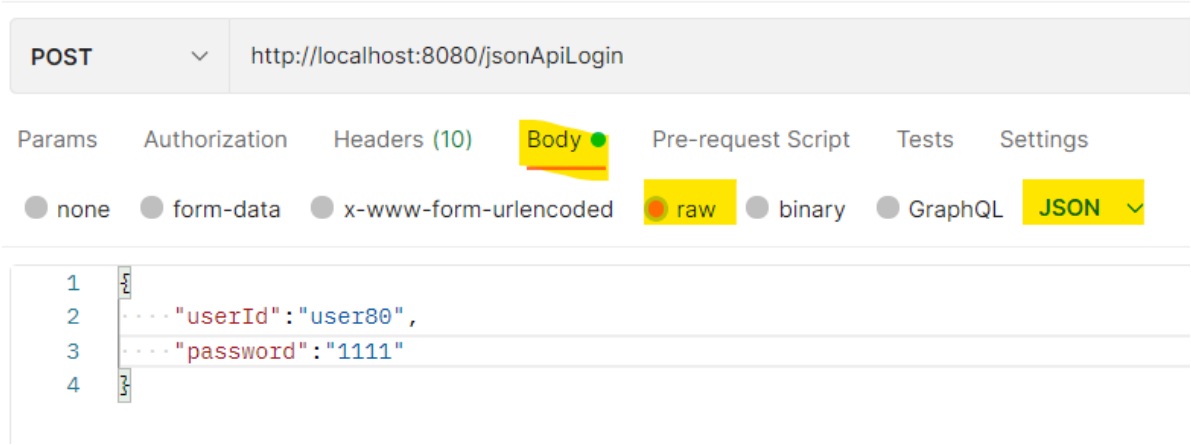
TokenGenerateFilter 수정 및 작동 확인

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

    //JSON 문자열 얻어오기
    String requestStr = extracted(request);

    log.info("try to login with json for api...............");
    log.info(requestStr);

    JSONObject jObject = new JSONObject(requestStr);

    String userId = jObject.getString("userId");
    String password = jObject.getString("password");

    UsernamePasswordAuthenticationToken authToken =
            new UsernamePasswordAuthenticationToken(userId, password);

    Authentication result = getAuthenticationManager().authenticate(authToken);

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

    return result;
}

// request를 JSON화 해주는 메소드
private String extracted(HttpServletRequest request)  {
    InputStream inputStream = null;
    ByteArrayOutputStream bos = null;
```

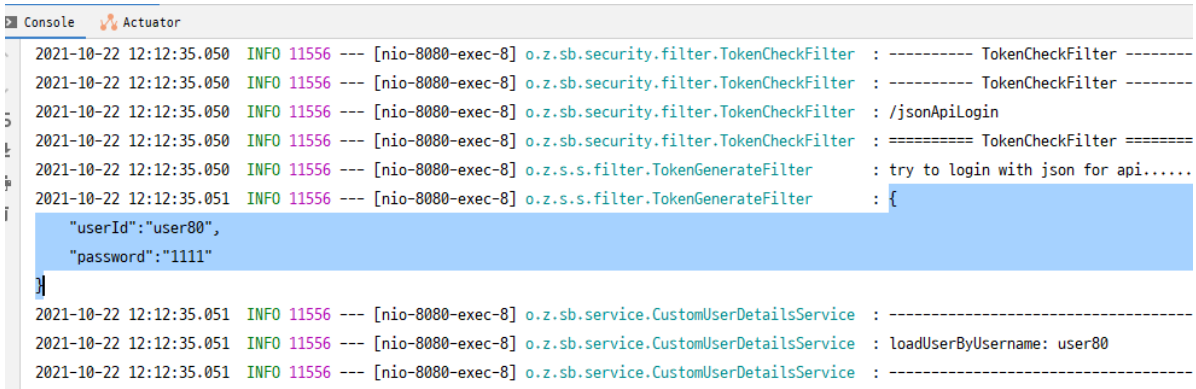
```
    try {
        inputStream = request.getInputStream();
        bos = new ByteArrayOutputStream();
        byte[] arr = new byte[1024];

        while (true) {
            int count = inputStream.read(arr);
            if (count == -1) {
                break;
            }
            bos.write(arr, 0, count);
        }
    }catch(Exception e){

    }finally {
        try{inputStream.close(); }catch(Exception e){}
        try {bos.close();}catch(Exception e){}
    }
    return bos.toString();
}
```

아직 성공시 토큰 발행까진 아님

postman으로 테스트



JSON 형식으로 로그인처리

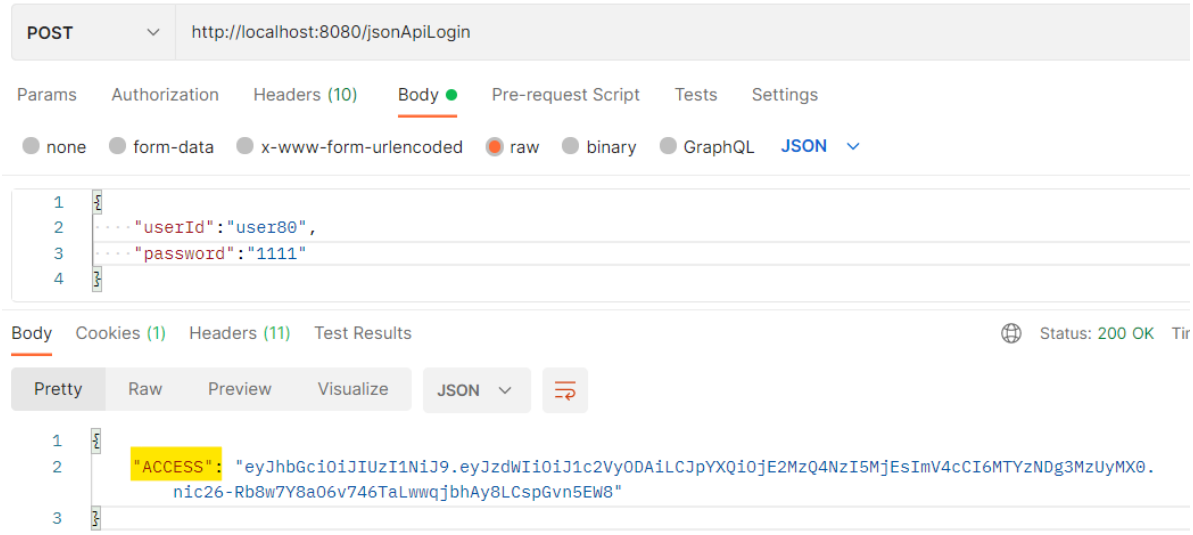


결과가 날아오는 것 확인 가능

TokenGenerateFilter 수정 - 성공시!

```
@Override
protected void successfulAuthentication(HttpServletRequest request,
HttpServletResponse response, FilterChain chain, Authentication authResult)
throws IOException, ServletException {
    log.info("successfulAuthentication: " + authResult);

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

    String mid = memberDTO.getMid();
    log.info("MEMBER MID: " + mid);

    String token = jwtUtil.generateToken(mid);  //토큰 생성
    //이제 전송하면됨

    JSONObject res = new JSONObject(Map.of("ACCESS", token));

    response.setContentType("application/json");
    PrintWriter out  = response.getWriter();
    out.println(res.toString());
    out.close();
}
```

postman으로 test



로그인 성공 시 access key와 함께 token이 value로 넘어옴

로그확인

```
---------------------------
UsernamePasswordAuthenticationToken [Principal=MemberDTO(mid=user80, mpw=$2a$10$NmWypMdm.XBb/DuCCBkLqe
successfulAuthentication: UsernamePasswordAuthenticationToken [Principal=MemberDTO(mid=user80, mpw=$2a
```

로그인 성공 시 로그인 정보가 같이 넘어옴

/jsonApiLogin -> Spring security -> Authentication -> 필터 -> access token

localstorage 사용

나중에 /api/board/list 호출 시 access token 발급받았던거 같이 들고가서 검증받고 맞으면 ok / 아니면 invalied & 메세지

TokenGenerateFilter 수정 - 실패시

```java
@Override
protected void unsuccessfulAuthentication(HttpServletRequest request,
HttpServletResponse response, AuthenticationException failed) throws
IOException, ServletException {
    log.info("unsuccessfulAuthentication: " + failed);

    response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);

    // json 리턴
    response.setContentType("application/json;charset=utf-8");
    JSONObject json = new JSONObject();
    String message = failed.getMessage();
    json.put("code", "401");
    json.put("message", message);

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

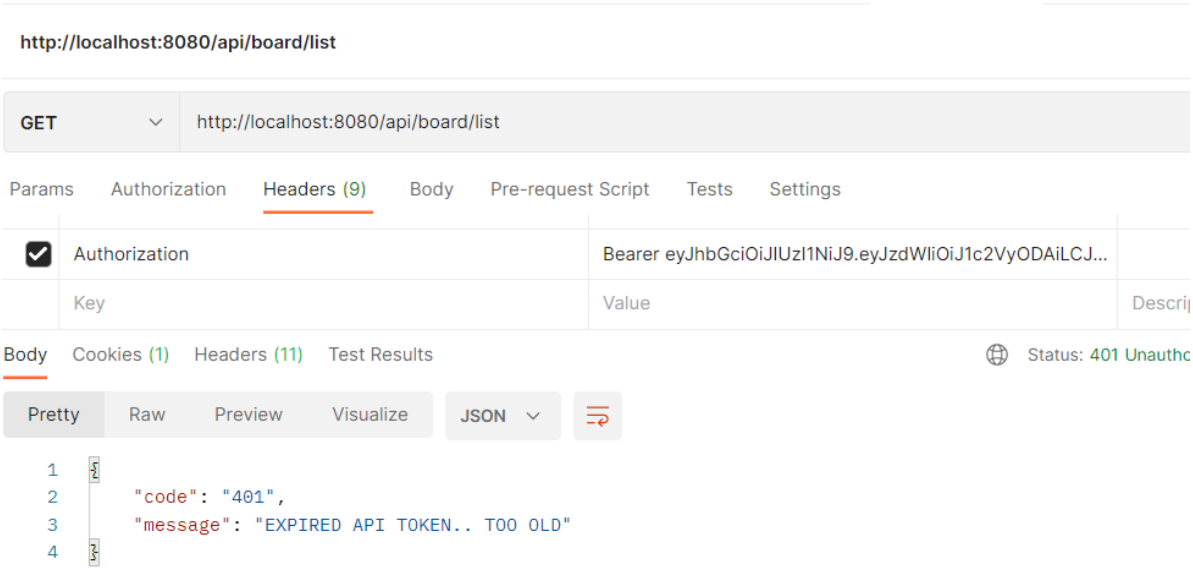
postman test - 없는 사용자로 로그인 시

```
{
    "userId":"user800",
    "password":"1111"
}
```

```
ısuccessfulAuthentication: org.springframework.security.authentication.BadCredentialsException: 자격 증명에 실패하였습니다.
```

postman test

이제 key값 앞에 ***Bearer*** 붙여서 넣어야함 - HTTP 인증 스킴



토큰이 만료될 시 too old 같은 error message 확인할 수 있고

expired 되면 다시 post 로 발급받아야함

재발급받은 코드 넣어서 get 방식 test 돌리면 정상적으로 출력되는 것 확인 가능

WebStorm으로 server - client test

Server 작업

CORSFilter 생성

```
@Component
@Order(Ordered.HIGHEST_PRECEDENCE)  //가장 먼저 작동하도록 설정
public class CORSFilter extends OncePerRequestFilter {
    //CORS (Cross Origin Resource Sharing)

    @Override
```

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

        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Credentials", "true");
        response.setHeader("Access-Control-Allow-Methods","*");
        response.setHeader("Access-Control-Max-Age", "3600");
        response.setHeader("Access-Control-Allow-Headers",
                "Origin, X-Requested-With, Content-Type, Accept, Key,
Authorization");


        if ("OPTIONS".equalsIgnoreCase(request.getMethod())) {
            response.setStatus(HttpServletResponse.SC_OK);
        } else {
            filterChain.doFilter(request, response);
        }
    }

}
```

Client 작업

WebStorm 에서 새로운 빈 프로젝트 생성 - ***apitest***

login.html 생성

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

<h1>Login</h1>

<input type="text" name="username" >

<input type="text" name="password" >
```

```
<button class="btn loginBtn" onclick="loginAjax()">login</button>

<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

<script>

    function loginAjax() {

        const param = {userId:"user1", password:1111}

        //post 방식으로 login 호출
        axios.post("http://192.168.0.23:8080/jsonApiLogin",param).then(response
=> {
            console.log(response)

            console.log(response.data)

        }).catch(function(err){
            console.log(err)
            alert(err.message)
        })

    }

</script>

</body>
</html>
```

결과

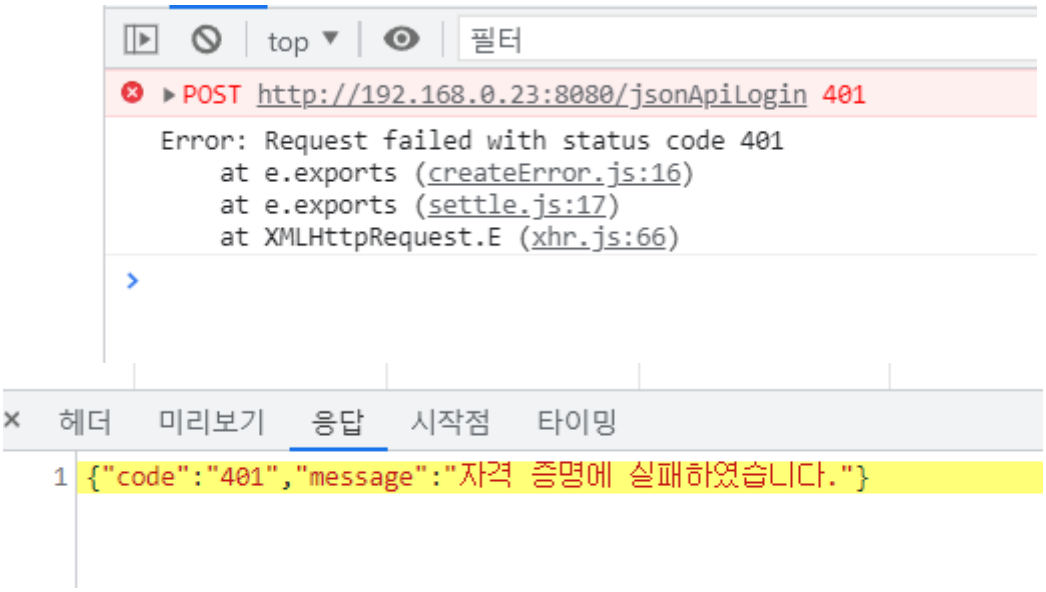
실존하는 id와 password 등록 시

```
POST http://192.168.0.23:8080/jsonApiLogin 500
Error: Request failed with status code 500
    at e.exports (createError.js:16)
    at e.exports (settle.js:17)
    at XMLHttpRequest.E (xhr.js:66)
6 --- [nio-8080-exec-7] o.a.c.c.C.[.[.[/].[dispatcherServlet]    : Servle

kpoint : JSONObject["password"] is not a string.
lueFormatException(JSONObject.java:2628) ~[json-20210307.jar:na]
```

password String으로 바꿔줌

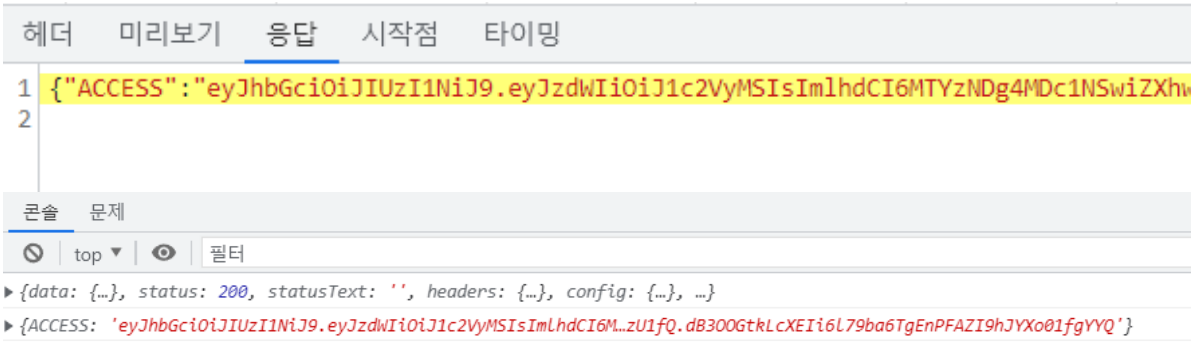
```
const param = {userId:"user1", password:"1111"}
```

틀린 계정 입력 시



자격증명 실패

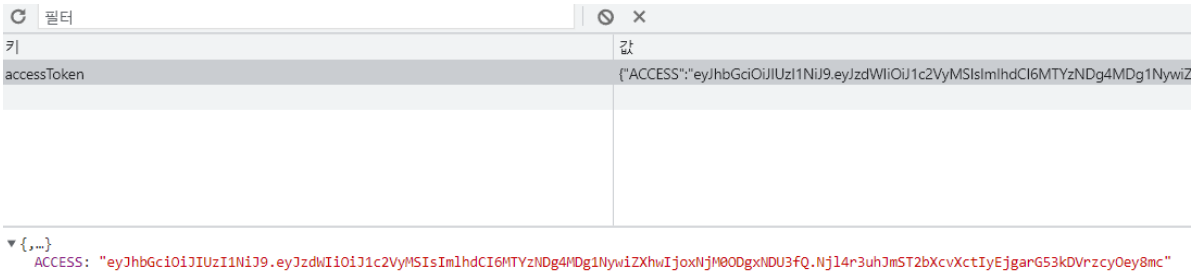
옳은 계정 입력 시



ACCESS Token 발급

localStorage 저장

```
localStorage.setItem("accessToken", JSON.stringify(response.data))
```



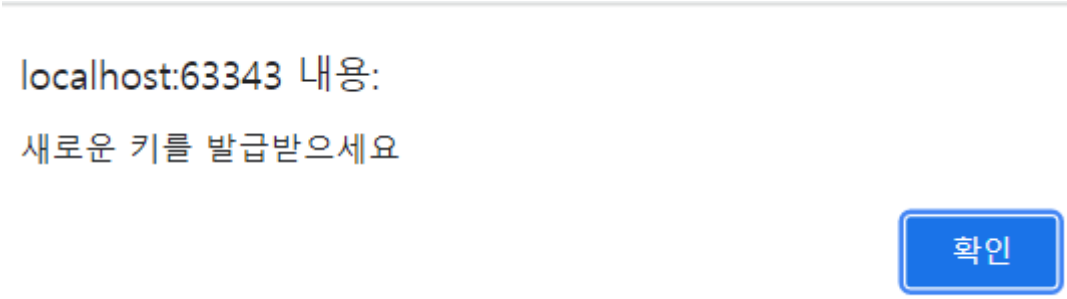
accessToken 이란 이름으로 토큰이 로컬스토리지에 저장됨

apiTest.html 생성

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

<button class="btn">CLICK</button>

<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

<script>

  document.querySelector(".btn").addEventListener("click", function () {

    const tokenStr = localStorage.getItem("accessToken")

    if (!tokenStr) {
      alert("새로운 키를 발급받으세요")
      self.location = "login.html"  //이동
      return
    }

  }, false)

</script>

</body>
</html>
```

결과

apiTest로 접속 시 경고창이 뜨고 login 으로 튕겨짐

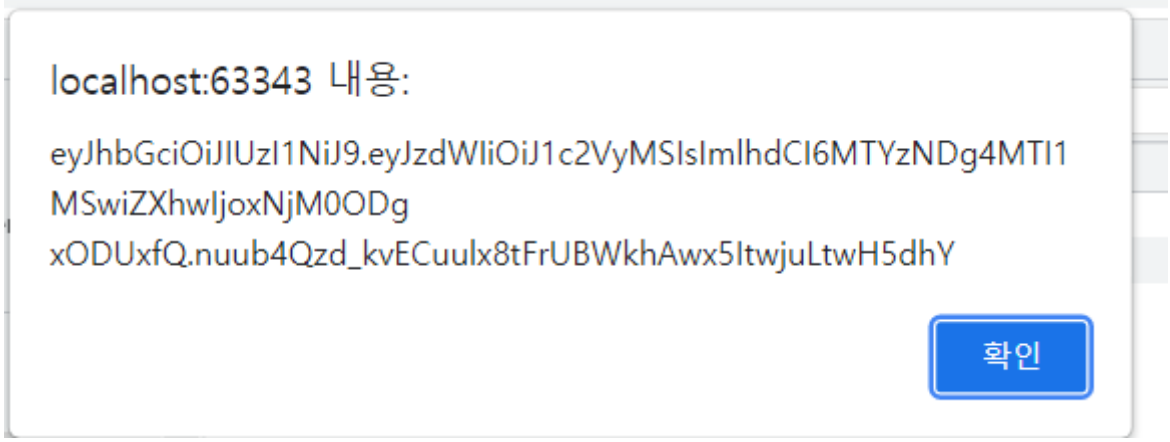


튕겨지고 버튼 누르면 accesstoken 새로 발급되어서 다시 apiTest 접속하면 튕겨지지 않음

apiTest 수정

```
const tokenObj = JSON.parse(tokenStr)

const accessTokenValue = tokenObj.ACCESS

alert(accessTokenValue)
```

click 했을때 받은 토큰이 뜨는지 확인

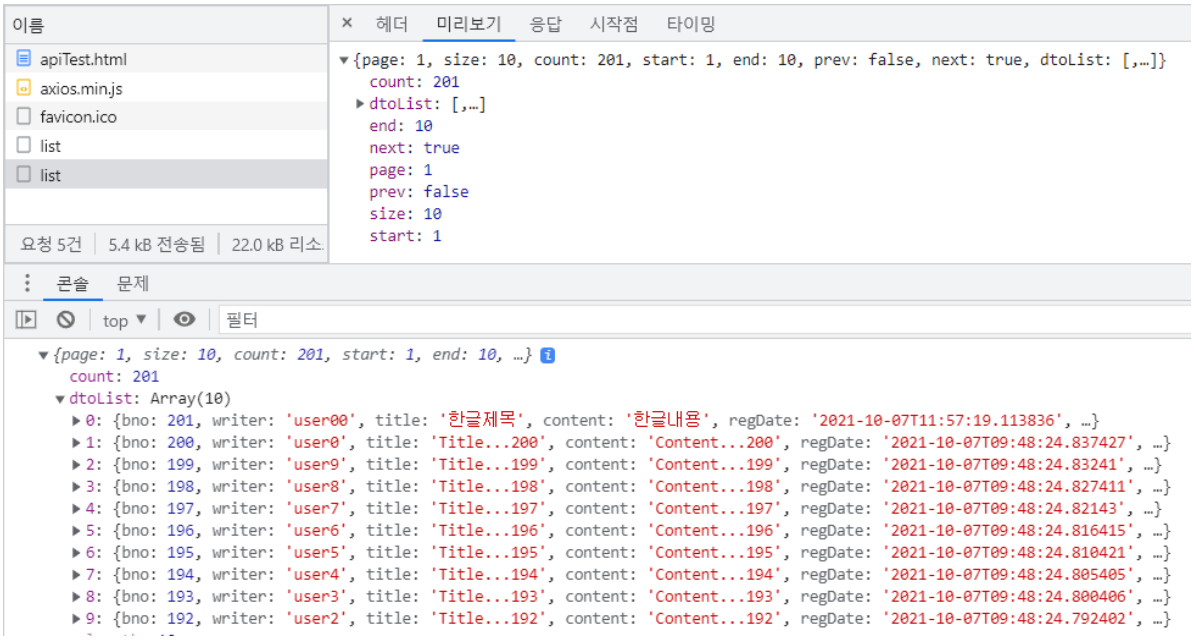


apiTest 수정

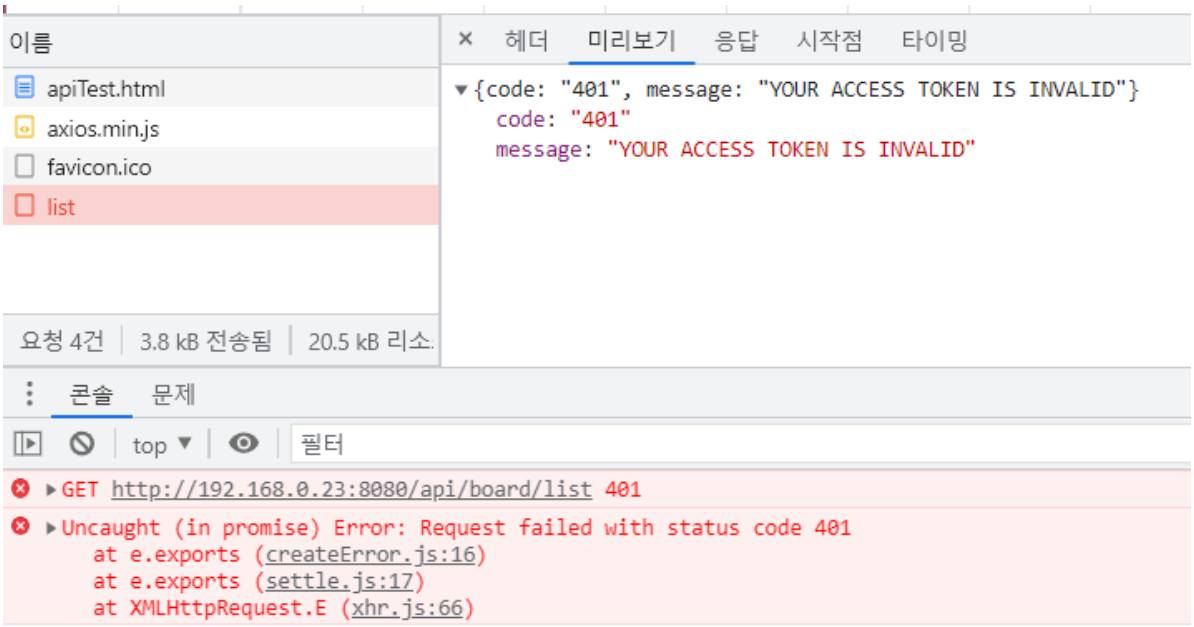
```
<script>

  document.querySelector(".btn").addEventListener("click", function () {

    const tokenStr = localStorage.getItem("accessToken")

    if (!tokenStr) {
      alert("새로운 키를 발급받으세요")
      self.location = "login.html"  //이동
      return
    }

    const tokenObj = JSON.parse(tokenStr)

    const accessTokenValue = tokenObj.ACCESS

    alert(accessTokenValue)

    sendRequest(accessTokenValue)

  }, false)

  async function sendRequest(token) {

    const headerObj = {Authorization : "Bearer " + token}
```

```
    const response = await axios.get("http://192.168.0.23:8080/api/board/list",
{
      headers : headerObj
    })

    console.log(response.data)

  }

</script>
```

결과

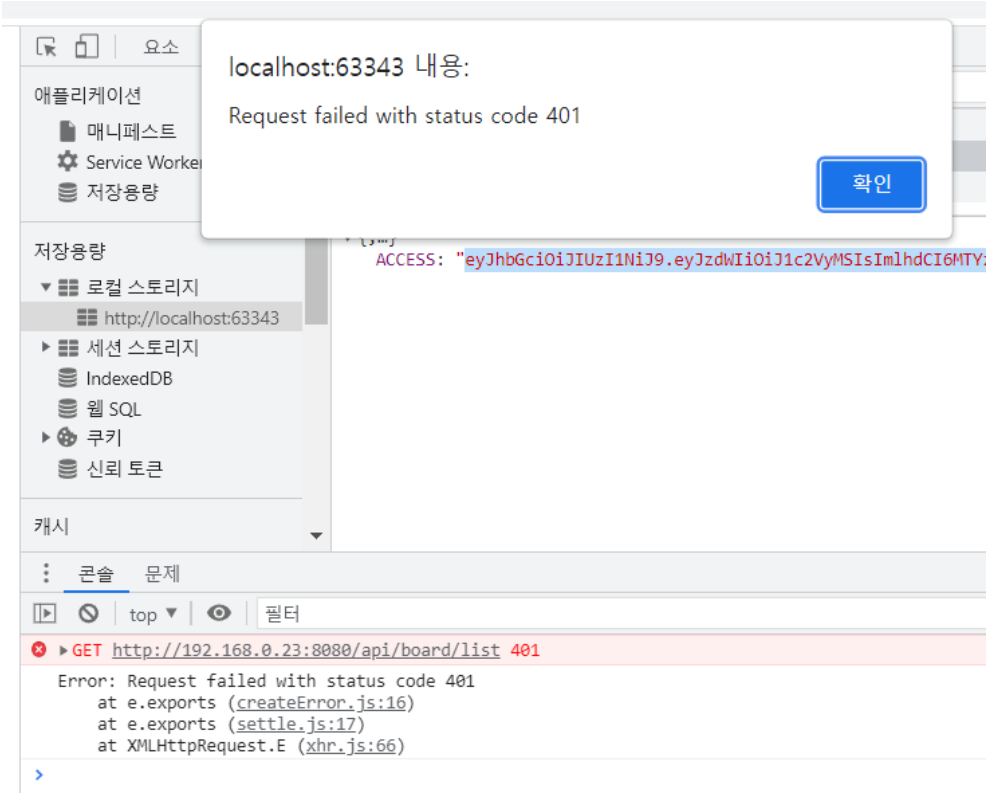


잘못된 토큰일 경우



매번 이렇게 일일히 지정하면 귀찮으니까 util.get / util.post 등의 함수를 정의하고 알아서 localstorage 에 있는거 가져다 쓰도록 설정하면 편하다!

기간 만료된 token일 경우



Server쪽

token 발행

TokenGenerateFilter

로그인 성공 -> Access / Refresh token 2개 생성

내용물은 같지만 기간에 차이가 있는 것

Token

entity - 이름 / 기한

JWTUtil

token check

access token <-

Autherization Header

(파라미터로 던짐 / custom header만들면 CORSFilter에서 걸림)

clint에서 check

access token의 유효기간 확인