

스프링 부트 웹 프로젝트

chapter12

API 서비스 만들기

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

Contents

part.1

API 서버를 위한 구성

part.2

API 서버를 위한 필터

part.3

API를 위한 인증 처리

API 서버 구성

읽어주세요!

지금까지 우리는 스프링 프로젝트 하나로 사이트를 구현하는 방식을 배웠습니다.

그런데 API(기능)을 구현하고 화면은 리액트와 같은 프론트 기술로 구현하는 방식도 있습니다.

그리고 두 개의 프로젝트를 연결하는 방식입니다.

세미 프로젝트는 첫번째 방식으로 진행하고, 파이널 프로젝트는 두번째 방식으로 진행할 예정입니다.

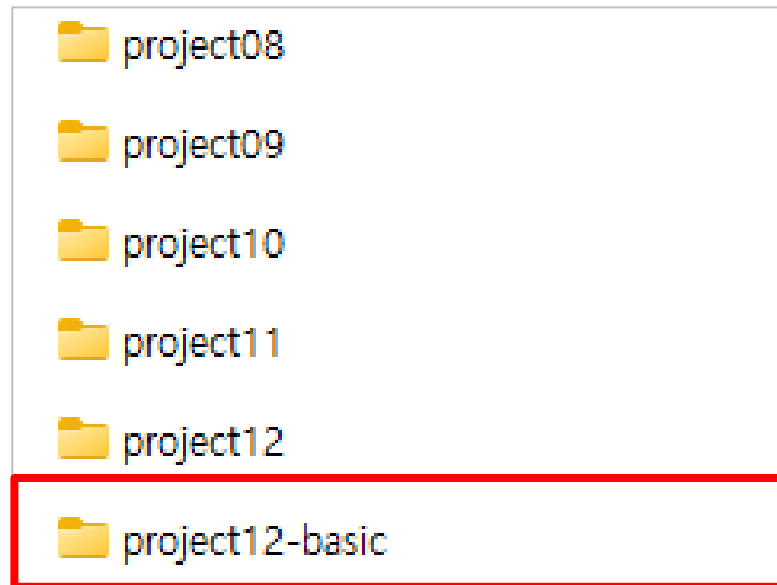
개발 방식이 혼동될 수 있기 때문에, 12장은 파이널 프로젝트를 시작하기 전에 다루도록 하겠습니다.

이 부분 양해 부탁드립니다^^

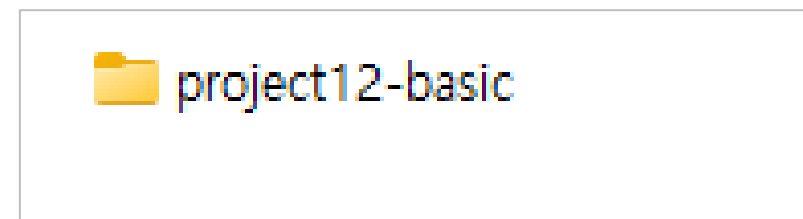


해당 프로젝트는 부분적으로 완성된 구조를 사용할 예정입니다.

spring-study 저장소에서 "project12-basic"을 복사하고, 본인의 저장소로 가져오세요.



spring-study 폴더



my-workspace 폴더

[illegible]

초기 서버-브라우저 통신 방식

초기의 웹 서버는 주로 브라우저와 통신하며, 사용자는 브라우저를 통해 서버에 데이터를 요청하였다. 그리고 서버는 요청을 받아서 HTML 페이지를 생성하고 브라우저에 전송하는 방식이었다.



다양한 클라이언트 프로그램의 등장

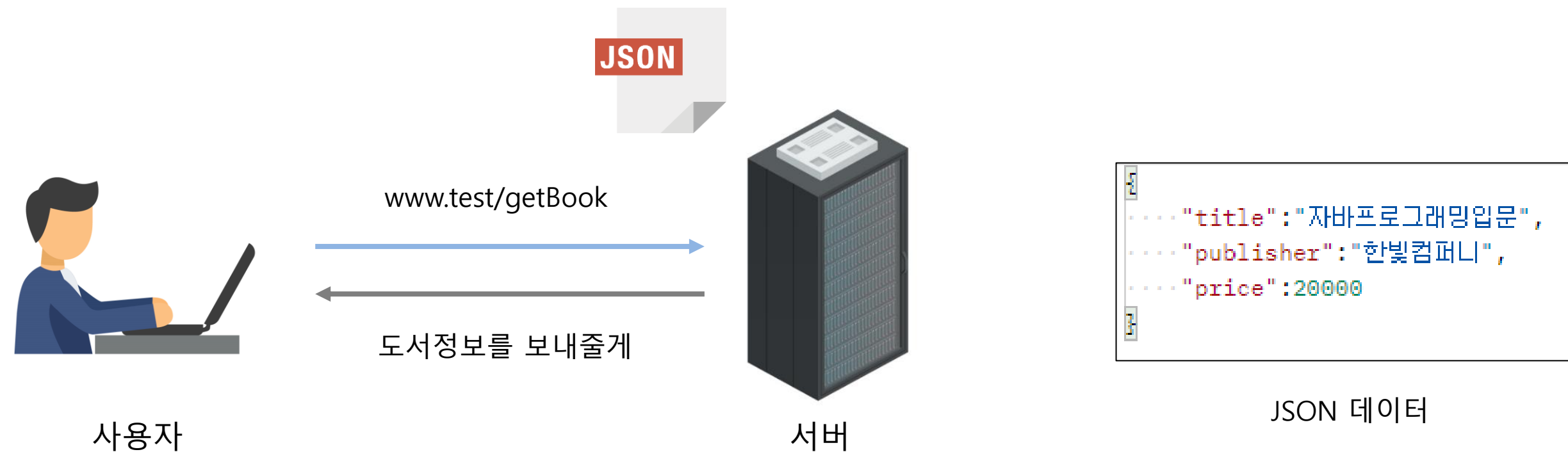
현재는 다양한 종류의 클라이언트 프로그램이 등장하였다. (브라우저, 모바일 앱, 스마트 TV 등)

화면과 데이터의 분리

최근 트렌드는 화면과 데이터를 분리하여 개발하는 방식이다.

화면은 클라이언트 UI에 맞게 개발하고, 서버는 순수한 데이터만 제공한다.

사용자가 서버에 데이터를 요청하면, 서버는 데이터를 전송한다.



API란?

다른 어플리케이션이나 사용자에게 데이터 제공하는 어플리케이션이다.

데이터 처리를 담당하는 서버를 "API 서버"라고 부른다.

일반 웹사이트는 브라우저를 통해 정보를 시각적으로 제공하지만, API는 데이터를 직접 제공한다.

API 설계 방식

API를 설계 방식에는 여러가지가 있으며, 주로 사용되는 방식은 REST와 SOAP 이다.

Rest API는 사용자 요청을 파악할 수 있도록 URL 주소를 설계하는 방식이다.

HTTP 메소드(GET, POST, PUT, DELETE 등)를 사용하여 설계한다.

<rest api 방식의 메소드>

GET + /board

게시물 조회 요청

POST + /board

게시물 등록 요청

API 서버 설계

- 게시물 관리 메소드를 만든다.
- 각 기능에 맞는 HTTP 메소드를 사용하고, URL 주소를 리소스를 통해 표현한다.
- 데이터만 처리하고, 화면은 만들지 않는다.

URL 설계

기능	URL	메소드
게시물 등록	/board/register	POST
게시물 목록 조회	/board/list	GET
게시물 상세 조회	/board/read	GET
게시물 수정	/board/modify	PUT
게시물 삭제	/board/remove	DELETE

기능	URL	메소드
회원 등록	/register	POST
회원 목록 조회	/member/list	GET

게시물 컨트롤러 만들기

- 사용자 요청을 처리하기 위해, BoardController 클래스를 생성한다.
- @RestController: @Controller + @ResponseBody
- 모든 메소드에 @ResponseBody 가 자동으로 적용되어, JSON형식으로 데이터가 반환된다.

BoardController

```
@RestController
@RequestMapping("/board")
public class BoardController {
}
```

컨트롤러에서 등록 처리

- POST 방식으로 게시물 등록 메소드를 추가한다.
- 게시물정보를 파라미터로 수집한다.
- 201 성공코드와 새로운 게시물 번호를 전송한다.

BoardController

```
@PostMapping
public ResponseEntity<Integer> register(@RequestBody BoardDTO dto) {
    int no = service.register(dto);
    return new ResponseEntity<>(no, HttpStatus.CREATED);
}
```

컨트롤러에서 목록 조회 처리

- GET 방식으로 게시물 목록 조회 메소드를 추가한다.
- 응답 메시지에 200 성공코드와 게시물 목록을 담는다.
- 결과값은 JSON형식으로 변환하여 전송한다.

BoardController

```
@GetMapping
public ResponseEntity<List<BoardDTO>> getList() {
    List<BoardDTO> list = service.getList();
    return new ResponseEntity<>(list, HttpStatus.OK);
}
```

컨트롤러에서 상세 조회 처리

- GET 방식으로 게시물 상세 조회 메소드를 추가한다.
- 파라미터로 게시물 번호를 수집한다.
- 응답 메시지에 200 성공코드와 게시물 데이터를 담는다.
- 결과값은 JSON형식으로 변환하여 전송한다.

BoardController

```
@GetMapping
public ResponseEntity<BoardDTO> read(@RequestParam int no) {
    BoardDTO dto = service.read(no);
    return new ResponseEntity<>(dto, HttpStatus.OK);
}
```

컨트롤러에서 수정 처리

- PUT 방식으로 게시물 수정 메소드를 추가한다.
- 파라미터로 변경된 게시물 정보를 수집한다.
- 204 성공코드를 전송한다.

BoardController

```
@PutMapping
public ResponseEntity modify(@RequestBody BoardDTO dto) {
    service.modify(dto);
    return new ResponseEntity(HttpStatus.NO_CONTENT);
}
```

컨트롤러에서 삭제 처리

- DELETE 방식으로 게시물 삭제 메소드를 추가한다.
- 파라미터로 게시물 번호를 수집한다.
- 204 성공코드를 전송한다.

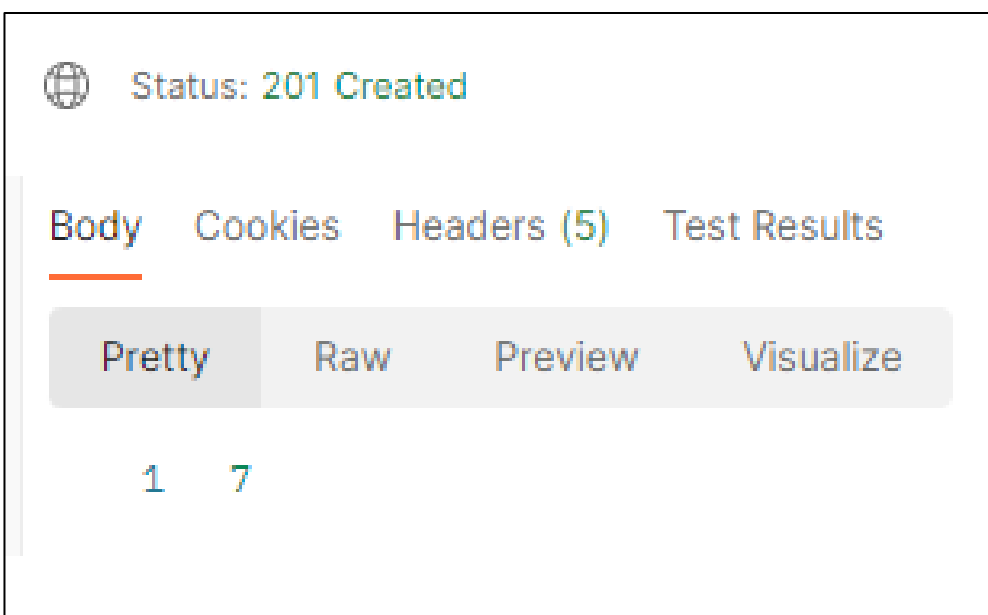
BoardController

```
@DeleteMapping
public ResponseEntity remove(@RequestParam int no) {
    service.remove(no);
    return new ResponseEntity(HttpStatus.NO_CONTENT);
}
```

테스트

1. Postman 프로그램을 실행한다.
2. 게시물 등록, 조회, 수정, 삭제 API를 호출한다.

게시물 등록 처리 결과



회원 컨트롤러 만들기

- 회원 등록 메소드를 추가한다.
- 회원 목록 조회 메소드를 추가한다.

MemberController

```
@RestController
public class MemberController {

    @Autowired
    MemberService service;

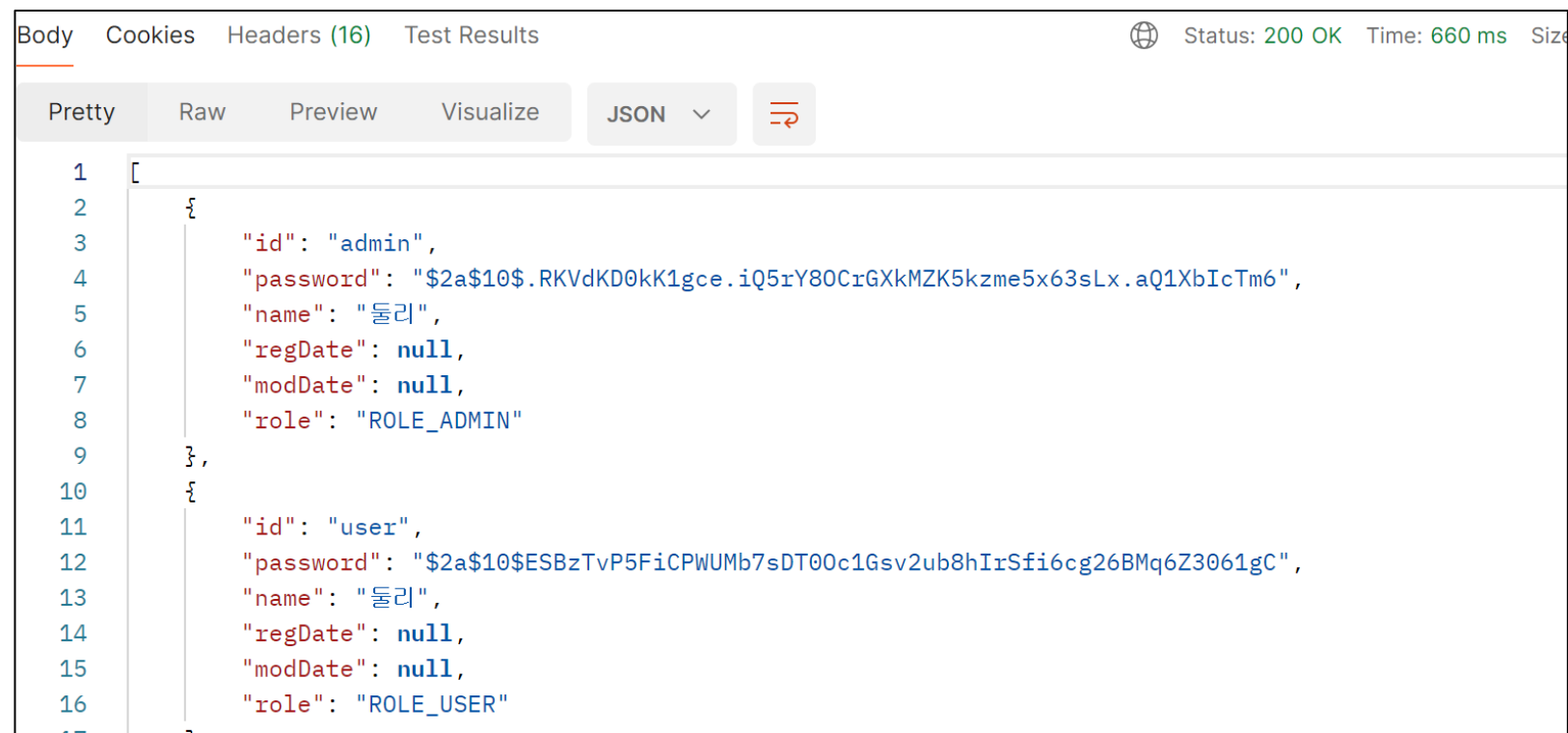
    @PostMapping("/register")
    public ResponseEntity<Boolean> register(@RequestBody Member
        boolean result = service.register(dto);
        return new ResponseEntity<>(result, HttpStatus.CREATED)
    }

    @GetMapping("/member/list")
    public ResponseEntity<List<MemberDTO>> getList() {
        List<MemberDTO> list = service.getList();
        return new ResponseEntity<>(list, HttpStatus.OK); //200
    }
}
```

테스트

1. Postman 프로그램을 실행한다.
2. 회원 등록, 조회 API를 호출한다.

회원 목록 조회 결과



```
Body  Cookies  Headers (16)  Test Results  Status: 200 OK  Time: 660 ms  Size: 1000B
Pretty  Raw  Preview  Visualize  JSON  ↗
1  [
2    {
3      "id": "admin",
4      "password": "$2a$10$.RKVdKD0kK1gce.iQ5rY80CrGXkMZX5kzme5x63sLx.aQ1XbIcTm6",
5      "name": "둘리",
6      "regDate": null,
7      "modDate": null,
8      "role": "ROLE_ADMIN"
9    },
10   {
11     "id": "user",
12     "password": "$2a$10$ESBzTvP5FiCPWUMb7sDT00c1Gsv2ub8hIrSfi6cg26BMq6Z3061gC",
13     "name": "둘리",
14     "regDate": null,
15     "modDate": null,
16     "role": "ROLE_USER"
17   }
18 ]
```

API 서버 필터

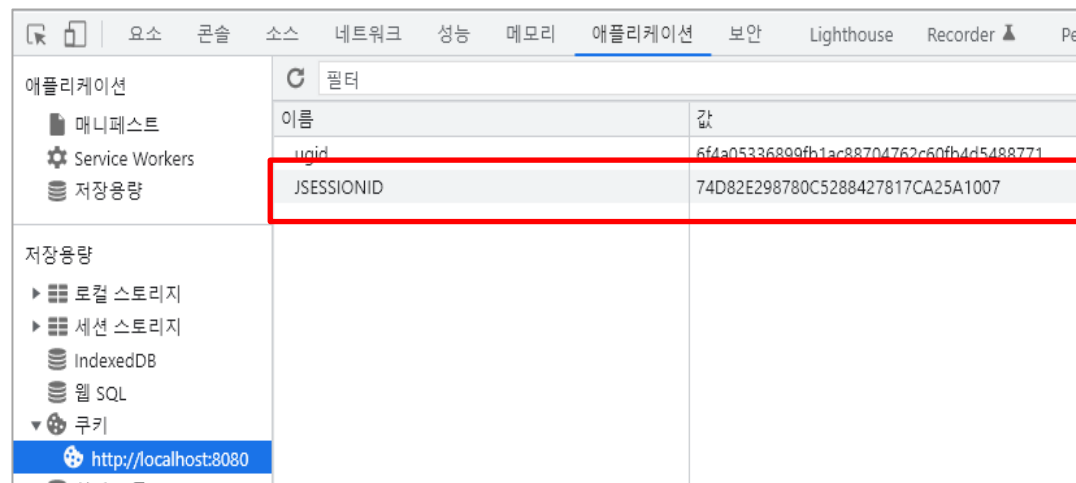
사용자 인증 방식

외부에서 무분별하게 서비스를 호출하면 서버에 부담이 될 수 있다.

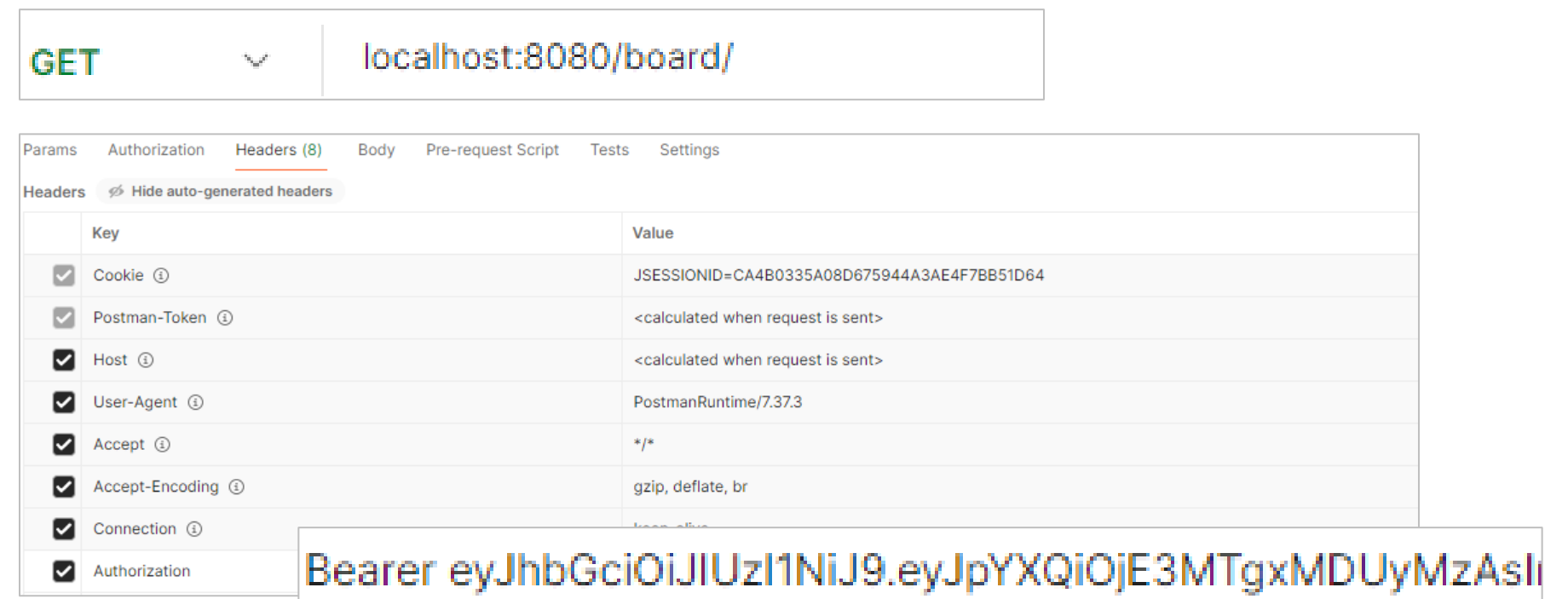
따라서 인증된 사용자에게만 서비스를 제공해야 한다.

기존 웹 사이트는 쿠키나 세션을 사용하여 사용자 인증을 처리 했지만, API는 주로 인증키를 사용한다.

고유한 키를 발급하고, 키를 이용해서 사용자를 인증하는 방식을 사용한다.



쿠키나 세션을 이용한 방식

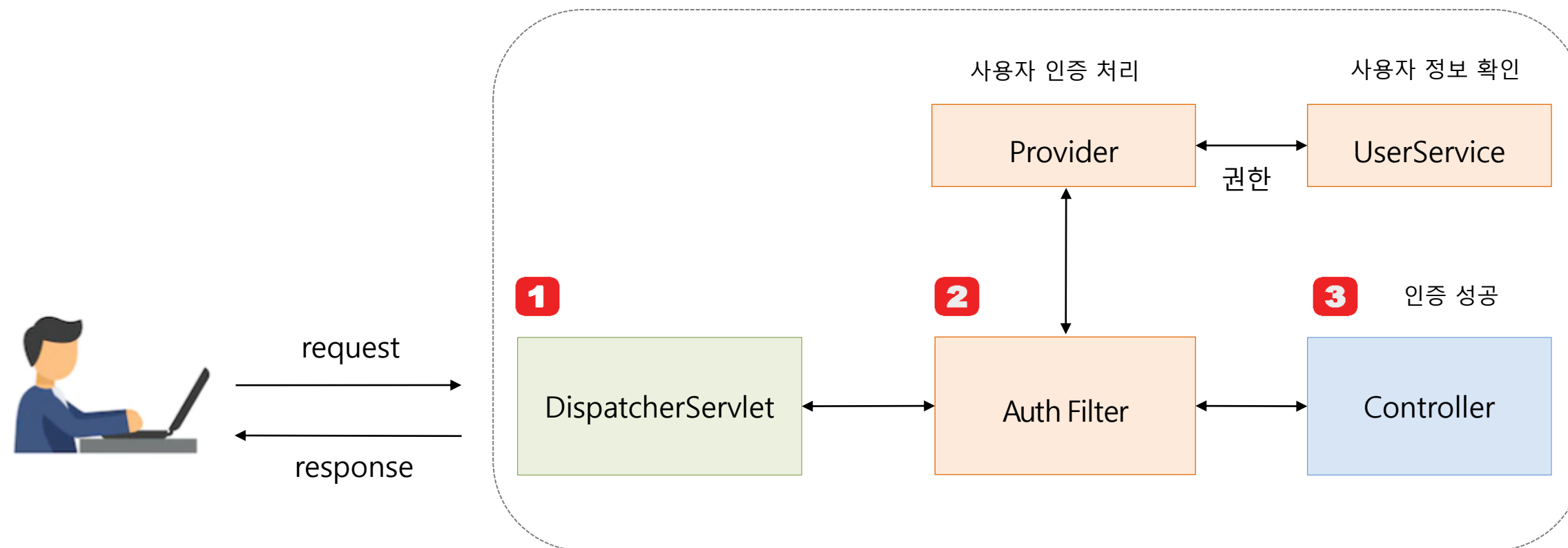


토큰을 이용한 방식

API 서버 필터

API를 위한 필터

외부에서 특정한 API를 호출할 때 인증에 사용할 토큰을 같이 전송하고, 서버에서는 이를 검증해야 한다.
이 과정에서 필요한 것은 토큰을 검사하는 필터(filter)이다.
필요한 필터를 생성하고, 이를 스프링 시큐리티에 추가한다.
그러면 스프링 동작 과정에서 필터가 수행된다.



Security 클래스 만들기

사용자 인증을 처리하기 위해, "build.gradle" 파일에 시큐리티 라이브러리와 JSON 라이브러리를 추가한다.

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    compileOnly 'org.projectlombok:lombok'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
    testImplementation 'org.springframework.security:spring-security-test'  
    testRuntimeOnly 'org.junit.platform:junit-platform-launcher'  
    implementation group: 'org.mariadb.jdbc', name: 'mariadb-java-client', ver  
    //시큐리티 라이브러리 추가  
    implementation 'org.springframework.boot:spring-boot-starter-security'  
    //json 라이브러리 추가  
    implementation 'io.jsonwebtoken:jjwt:0.9.1'  
    implementation 'javax.xml.bind:jaxb-api:2.3.0'  
    implementation 'net.minidev:json-smart:2.4.7'  
}
```

주석을 해제하세요!

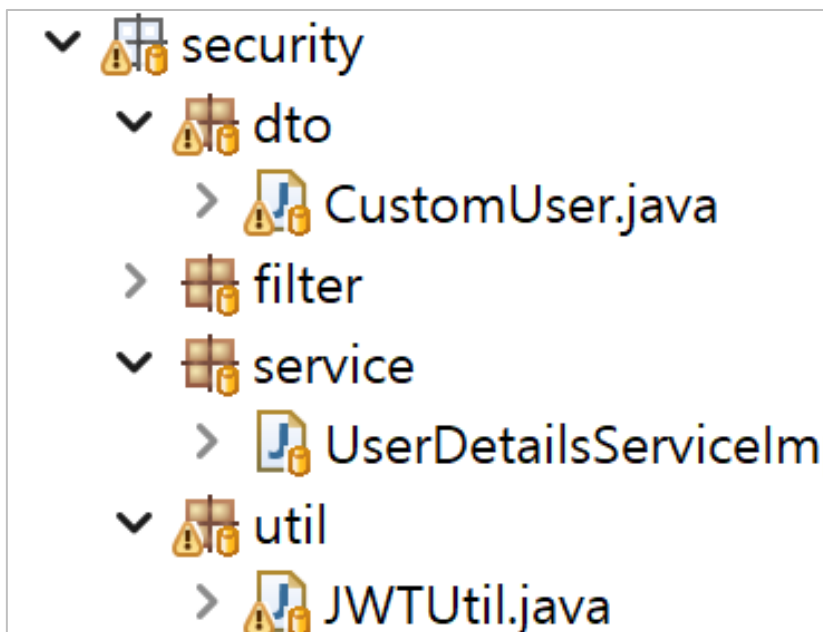
사용자 인증 클래스 만들기

사용자 인증과 관련된 클래스를 만든다.

CustomUser: 사용자 정보를 담는 클래스

UserDetailsServiceImpl: 사용자를 인증하고 권한을 부여하는 클래스

JWTUtil: JWT(JSON Web Token)을 생성하고 검증하는 클래스



CustomUser, UserDetailsServiceImpl, JWTUtil
클래스들을 주석 해제 해주세요!

Security 클래스 만들기

어플리케이션의 보안 설정을 하기 위한 클래스를 만든다.

사용자 정보 클래스, 인증 서비스, JWT 처리 클래스를 빈으로 등록한다.

config
SecurityConfig.java

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    UserDetailsService customUserDetailsService() {
        return new UserDetailsServiceImpl();
    }

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

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

MemberService에서 사용자 비밀번호 처리

패스워드 인코더를 사용하여 사용자의 패스워드를 암호화한다.

```
@Override
public boolean register(MemberDTO dto) {
    String id = dto.getId();
    MemberDTO getDto = read(id);
    if (getDto != null) {
        System.out.println("사용중인 아이디입니다.");
        return false;
    }
    Member entity = dtoToEntity(dto);

    // 패스워드 인코더로 패스워드 암호화하기
    String enPw = passwordEncoder.encode(entity.getPassword());
    entity.setPassword(enPw);
}
```


ApiCheckFilter 만들기

필터를 통해 요청이 들어 올 때마다 인증정보를 검사한다.

ApiCheckFilter를 만들기 위해서 OncePerRequestFilter 클래스를 상속받고, doFilterInternal() 함수를 구현해야 한다.

- OncePerRequestFilter: 필터 기능을 제공하는 추상 클래스로, 상속으로 구현해서 사용한다.
- doFilterInternal(): HTTP 요청을 받아 들이고, 필요한 보안 처리를 수행하는 함수이다.

```
@Log4j2
public class ApiCheckFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServl

        log.info("REQUESTURI: " + request.getRequestURI());

        log.info(antPathMatcher.match(pattern, request.getRequestURI()));

        if(antPathMatcher.match(pattern, request.getRequestURI())) {

            log.info("ApiCheckFilter.....");
            log.info("ApiCheckFilter.....");
            log.info("ApiCheckFilter.....");
```

주석을 해제 해주세요!

인증체크필터 등록하기

SecurityConfig에서 인증 체크 필터를 등록하고, /member 또는 /board 요청이 들어오면 이 필터를 실행시킨다. 스프링 시큐리티는 기본적으로 UsernamePasswordAuthenticationFilter이라는 필터가 사용자의 아이디와 패스워드를 기반으로 인증을 처리한다.

이 필터 보다 토큰 기반의 필터가 먼저 실행되도록 우선순위를 설정해야 한다.

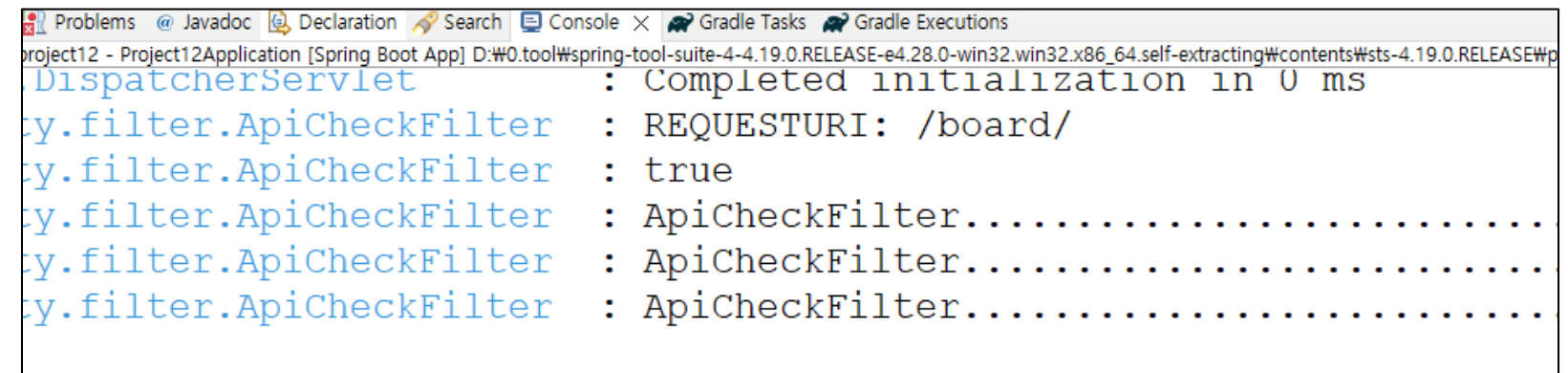
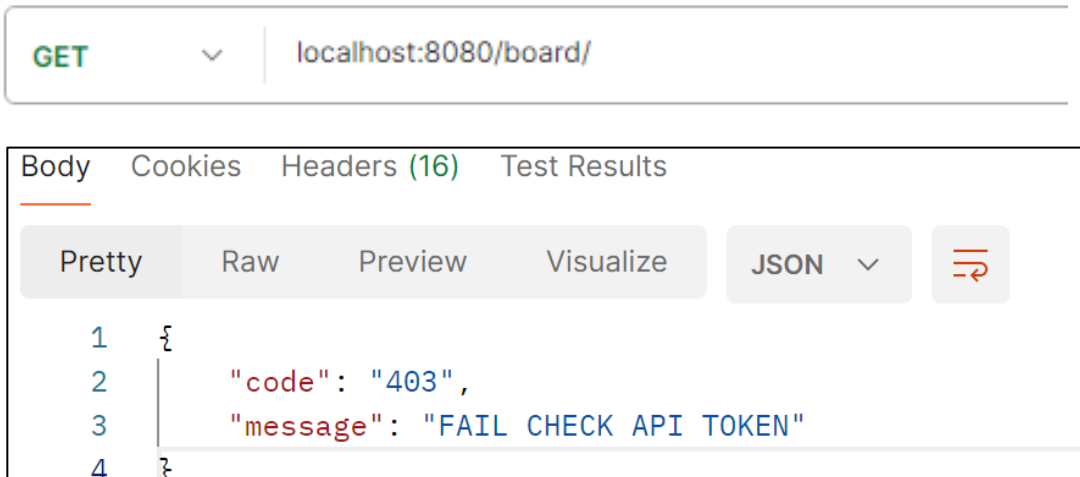
- UsernamePasswordAuthenticationFilter: 사용자 이름과 비밀번호를 사용하는 시큐리티의 기본 필터

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests(auth -> {
            auth.requestMatchers("/member/**", "/board/**").authenticated();
        })
        .addFilterBefore(new ApiCheckFilter(arr, jwtUtil(), customUs
```

테스트

이제 localhost:8080/board/list 주소를 호출한다.

이때 ApiCheckFilter가 동작되는지 확인한다.



인증 처리

API를 위한 인증 처리

인증 처리

로그인을 처리하는 API를 만든다.

사용자가 로그인에 성공하면 인증키를 발급한다.

이 인증키를 헤더에 넣어서 API 요청 하는데 사용할 수 있고, 일정 기간 동안 유효하다.

GET



localhost:8080/api/login?id=user1&pw=1234

Body Cookies (1) Headers (19) Test Results

Pretty

Raw

Preview

Visualize

Text



```
1 eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOiE3MTgxOTcxOTIsImV4cCI6MTcyMDc4OTE5Miwic3ViIjoiaXNlcjEifQ.CAil09sNtCqkUERBFTw333o1ktUjtJ3m1oqRcxAA1DQ
```

ApiLoginFilter 만들기

- ApiLoginFilter를 만들기 위해서는 AbstractAuthenticationProcessingFilter 클래스를 상속받고, attemptAuthentication() 함수를 구현해야 한다.
- 파라미터로 사용자 아이디와 패스워드를 받아서, 토큰 객체를 생성한다.
- AuthenticationManager 에게 토큰 객체를 전달하면, 인증 매니저는 실제 인증 작업을 수행한다. 이 과정에서 아이디와 패스워드가 맞는지 확인한다.
- AbstractAuthenticationProcessingFilter: 필터 기능을 제공하는 추상 클래스로, 상속으로 구현해서 사용한다.
- attemptAuthentication(): HTTP 요청을 처리하고 사용자를 인증하는 함수이다.

```
@Log4j2
public class ApiLoginFilter extends AbstractAuthenticationProcessingFilter {

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

        String id = request.getParameter("id");
        String pw = request.getParameter("pw");

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

주석을 해제 해주세요!

SecurityConfig에서 인증필터 처리하기

스프링 시큐리티에서 인증필터를 설정한다.

인증 매니저를 생성하고, 시큐리티에 등록한다.

/api/login로 요청이 들어오면 인증을 처리하는 ApiLoginFilter 필터를 생성한다.

인증매니터에 ApiLoginFilter을 등록한다.

UsernamePasswordAuthenticationFilter 기본 필터가 동작하기전에, ApiLoginFilter를 먼저 처리하도록 수행순서를 설정한다.

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

    http.addFilterBefore(apiCheckFilter(), UsernamePasswordAuthenticationFilter.

    AuthenticationManagerBuilder authenticationManagerBuilder = http
        .getSharedObject(AuthenticationManagerBuilder.class);
    AuthenticationManager authenticationManager = authenticationManagerBuilder.b

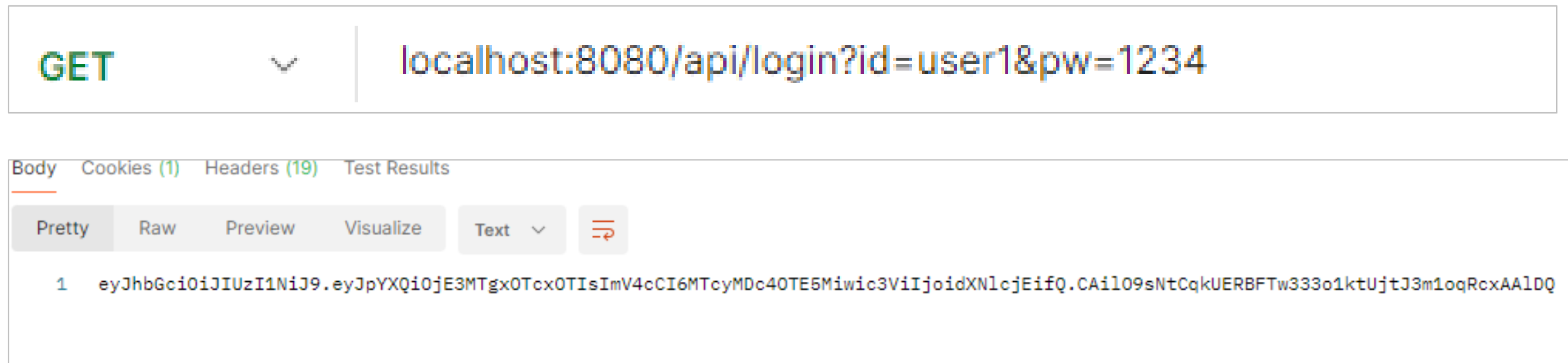
    http.authenticationManager(authenticationManager);

    ApiLoginFilter apiLoginFilter = new ApiLoginFilter("/api/login", jwtUtil());
    apiLoginFilter.setAuthenticationManager(authenticationManager);

    http.addFilterBefore(apiLoginFilter, UsernamePasswordAuthenticationFilter.cl

    return http.build();
}
```

1. 로그인 API를 호출하여 토큰을 발급받는다.



2. 헤더에 발급받은 토큰을 추가하고, 게시물 조회 API를 호출한다.

key: Authorization

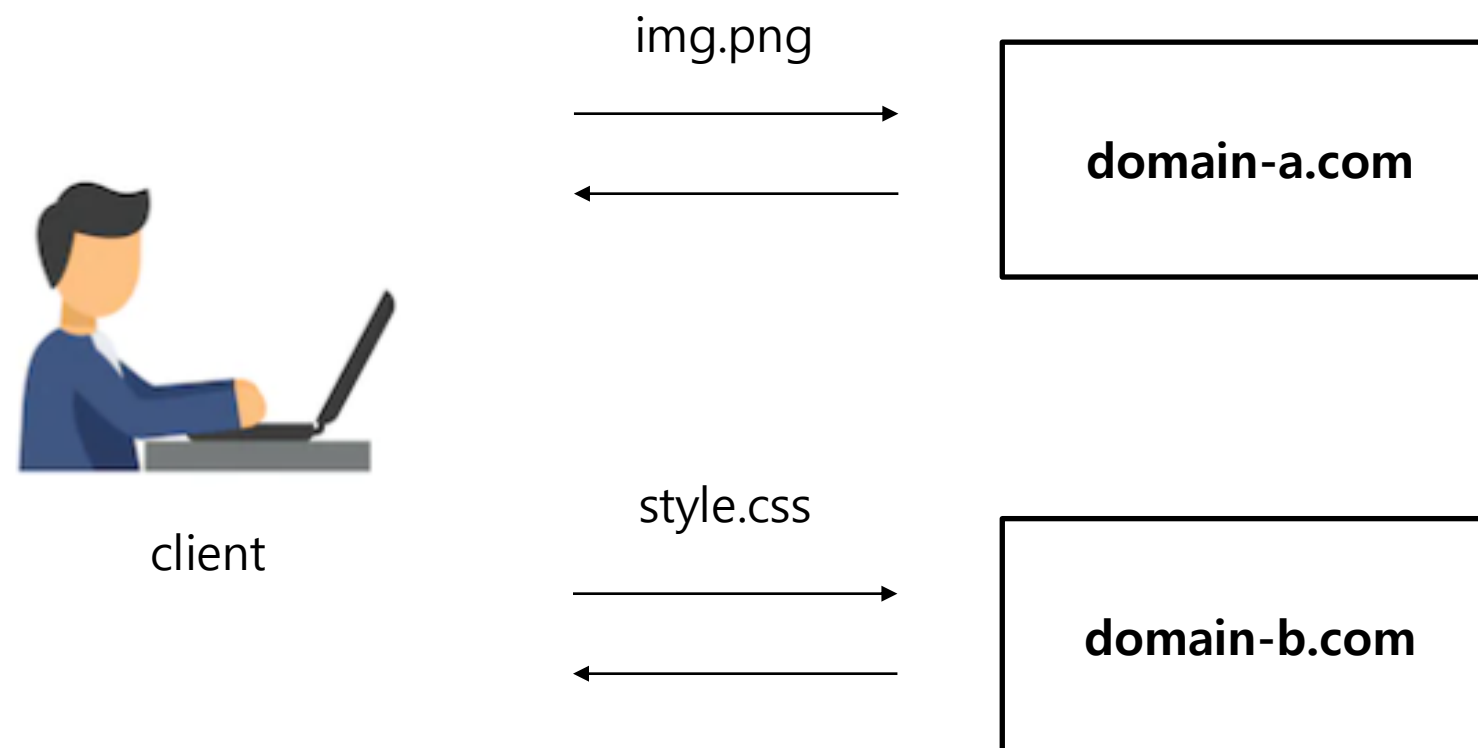
value: 토큰

GET localhost:8080/board/list

Params	Authorization	Headers (8)	Body	Pre-request Script	Tests	Settings
Headers Hide auto-generated headers						
	Key	Value				
<input checked="" type="checkbox"/>	Cookie ⓘ	JSESSIONID=CA4B0335A08D675944A3AE4F7BB51D64				
<input checked="" type="checkbox"/>	Postman-Token ⓘ	<calculated when request is sent>				
<input checked="" type="checkbox"/>	Host ⓘ	<calculated when request is sent>				
<input checked="" type="checkbox"/>	User-Agent ⓘ	PostmanRuntime/7.37.3				
<input checked="" type="checkbox"/>	Accept ⓘ	*/*				
<input checked="" type="checkbox"/>	Accept-Encoding ⓘ	gzip, deflate, br				
<input checked="" type="checkbox"/>	Connection ⓘ	keep-alive				
<input checked="" type="checkbox"/>	Authorization	eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOiE3MTg4MTI2OT...				

CORS(Cross-Origin Resource Sharing)

다른 도메인에서 리소스를 요청할 때 발생하는 보안 문제이다.



CORSFilter 만들기

- CORSFilter를 만들기 위해서는 OncePerRequestFilter 클래스를 상속받고, doFilterInternal() 함수를 구현해야 한다.
- 모든 도메인에서의 요청을 허용하고, 모든 메소드 방식을 허용한다.

```
@Component
@Order(Ordered.HIGHEST_PRECEDENCE)
public class CORSFilter extends OncePerRequestFilter {

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

        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Credentials", "true");
        response.setHeader("Access-Control-Allow-Methods", "*");
```

주석을 해제해 주세요!

CORSFilter 만들기

- HTTP 메서드가 OPTIONS일 경우 프리플라이트 요청은 200 OK 코드로 응답한다.

