

회원관리 및 게시판 시스템

이예빈

스프링부트 DB 시스템 JPA 설정



스프링 부트 build gradle 을 통해

```
runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
```

마리아 DB 를 사용할수 있게 라이브러리를 가져옵니다.

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
```

또한 JPA 기능을 사용하기 위해 JPA 기능을 넣어줍니다

```
#MariaDB
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=123456
spring.datasource.url=jdbc:mariadb://localhost:3306/user

server.servlet.encoding.force-response=true
```

마리아 DB 시스템 설정

스프링 부트에서 로컬에서 실행되고 있는 DB 의 주소를 설정합니다. 그리고 해당 DB 에 접근할 수 있게 해당 DB 의 username 과 password 를 입력해줍니다.

```
#Jpa Properties
spring.jpa.open-in-view=false
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true
```

spring.jpa.open-in-view=false: HTTP 요청 처리가 끝날 때 영속성 컨텍스트를 닫아 리소스를 릴리즈합니다. 이것은 지연로딩이 필요할 때 추가 쿼리를 실행하여 데이터를 가져오게 됩니다.

spring.jpa.hibernate.ddl-auto=update: Hibernate 에서 데이터베이스 스키마 생성 및 업데이트를 제어하는 설정입니다.

spring.jpa.properties.hibernate.show_sql=true: Hibernate 에서 실행되는 SQL 쿼리를 로깅할지 여부를 제어하는 설정입니다.

spring.jpa.properties.hibernate.format_sql=true: Hibernate 에서 출력되는 SQL 쿼리를 포맷팅하여 가독성을 높이는 설정입니다.

Entity 작업

Entity는 데이터베이스 테이블과 1:1 매핑되는 자바 클래스라고 정의합니다.

Entity 기본 구조:

```
@Entity
@Getter
@Setter
@Table(name = "comment_table")
@NoArgsConstructor(access = AccessLevel.PROTECTED)
```

@Entity: 해당 클래스가 Entity인 것을 선언합니다

@NoArgsConstructor: 접근 제한을 하여 기본 생성자의 무분별한 생성을 막아서 의도하지 않은 엔티티를 만드는 것을 막을 수 있습니다.

@Getter, @Setter: 이 메서드를 사용하면 따로 Entity 별로 get 메서드나 set 메서드를 사용할 필요가 없습니다.

Nullable = false로 지정된 Entity들은 null값이 들어올 수 없습니다.

User Entity: 가입된 회원의 정보를 저장하는 Entity

```
@Table(name = "user_table")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false, unique = true)
    private String name;

    private String roles; // USER, ADMIN
}
```

Id: 유저의 DB의 각 튜플을 식별하기 위해 설정된 프라이머리 키이다. 해당 유저정보가 DB에 들어올 경우 순차적으로 값이 증가합니다

Username: 해당 유저가 설정한 id값으로 unique = true를 통해 중복된 값을 가질 수가 없습니다.

Password : 유저가 직접 설정하는 비밀번호 값으로 해시화 과정을 거쳐 저장됩니다

Name: 유저가 직접 설정하는 웹 서비스에서 직접 사용되는 이름이다. 중복된 값을 가질 수가 없습니다.

Board Entity: 작성되는 게시글을 저장하는 Entity

```
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "user_table")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(nullable = false, unique = true)
    private String username;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false, unique = true)
    private String name;

    private String roles; // USER, ADMIN
```

Id: 게시글 DB의 각 튜플을 식별하기 위해 설정된 프라이머리 키이다. 해당 게시글이 DB에 들어올 경우 순차적으로 값이 증가합니다

writer: 게시글을 작성한 유저의 닉네임을 해당 게시글 DB에 저장합니다.

Title: 게시글 제목을 담은 Entity

Content: 게시글 내용을 담은 Entity TEXT 형식으로 저장됩니다.

Comment Entity: 게시물에 작성된 댓글을 저장하는 Entity

```
@NoArgsConstructor(access = AccessLevel.PROTECTED)
public class CommentEntity extends BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(length = 20, nullable = false)
    private String commentWriter;

    @Column
    private String commentContents;

    /* Board:Comment = 1:N */
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "board_id")
    @JsonBackReference
    private BoardEntity boardEntity;

    public Long getBoardId() {
        return boardEntity != null ? boardEntity.getId() : null;
    }
}
```

id: 댓글 DB의 각 튜플을 식별하기 위해 설정된 프라이머리 키이다. 해당 댓글이 DB에 들어올 경우 순차적으로 값이 증가합니다

commentWriter: 댓글을 작성한 유저의 닉네임을 해당 게시물 DB에 저장합니다.

commentContents: 댓글 내용을 담은 Entity

ManyToOne(fetch = FetchType.LAZY): 이 어노테이션은 댓글 Entity 가 게시물 Entity 에 1 대 다수의 관계가 되게 설정해줍니다. **FetchType.LAZY** 를 사용하면 연관 엔티티가 실제로 필요할 때만 데이터베이스에서 가져옵니다.

@JsonBackReference: 다대일 관계를 가진 엔티티끼리 서로 참고할 때 순환 참조 문제가 발생하는데 이렇게 될 경우 서로 계속 참조하려고만 하여서 무한 루프가 발생합니다, 이를 방지하기 위해 사용합니다.

Like Entity: 게시글의 추천을 나타내는 Entity

```
public class Like {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "heart_id")  
    private Long id;  
  
    @ManyToOne(fetch = LAZY)  
    @JoinColumn(name = "user_id")  
    private User user;  
  
    @ManyToOne(fetch = LAZY)  
    @JoinColumn(name = "board_id")  
    private BoardEntity boardEntity;  
}
```

Id: 좋아요 B의 각 튜플을 식별하기 위해 설정된 프라이머리 키입니다. 해당 댓글이 DB에 들어올 경우 순차적으로 값이 증가합니다

User,boardEntity: 유저와 게시판과 좋아요 간의 1대 다수의 관계를 설정해줍니다. 이 같은 연결된 DB를 통해 해당 게시글의 추천을 한 유저를 판별할 수 있습니다.

Dto 작업

```
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class RegisterDto {  
    private String username;  
    private String password;  
    private String name;  
}
```

DTO: DTO 는 프로세스 간에 데이터를 전달하는 용도의 객체이다. 비즈니스 로직을 포함하지 않는 데이터를 전달하기 위한 단순한 객체를 뜻합니다. 예를 들어 회원가입 등에서 유저 정보의 입력을 Entity 를 통해 직접 입력 받을 경우 유출되면 안되는 민감한 정보들을 포함시키고 있기 때문에 Dto 를 사용합니다.

JPA 레포지터리

엔티티만으로는 데이터베이스에 데이터를 저장하거나 조회 할 수 없습니다. 데이터 처리를 위해서는 실제 데이터베이스와 연동하는 JPA 리포지터리가 필요합니다

리포지터리란?

리포지터리는 엔티티에 의해 생성된 데이터베이스 테이블에 접근하는 메서드들(예: `findAll`, `save` 등)을 사용하기 위한 인터페이스입니다. 데이터 처리를 위해서는 테이블에 어떤 값을 넣거나 값을 조회하는 등의 **CRUD(Create, Read, Update, Delete)**가 필요합니다. 이 때 이러한 **CRUD**를 어떻게 처리할지 정의하는 계층이 바로 리포지터리입니다.

유저 리포지토리

```
public interface UserRepository extends JpaRepository<User, Integer> {  
    public User findByUsername(String username);  
    public User findByName(String name);  
}
```

findByUsername: User Entity 와 연동되어 있는 DB 에서 username 을 가지고 있는 User 정보를 조회할 수 있는 메서드입니다.

findByName: User Entity 와 연동되어 있는 DB 에서 name 값을 가지고 있는 User 정보를 조회할 수 있는 메서드입니다.

게시판 리포지토리

```
public interface BoardRepository extends JpaRepository<BoardEntity, Long> {  
    List<BoardEntity> findByTitleContaining(String keyword);  
    List<BoardEntity> findAll();  
    List<BoardEntity> findByWriter(String writer);  
}
```

findAll: 모든 게시글을 조회할 수 있는 메서드입니다

findByWriter: 작성자를 통해 게시글을 조회할 수 있습니다

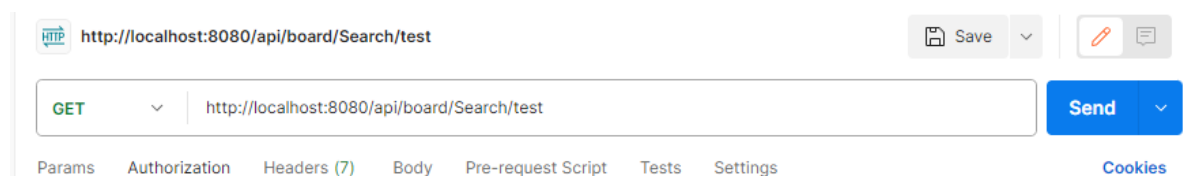
Custom Query

@Query: 데이터베이스 쿼리의 성능을 향상시키기 위해 사용되는 커스텀 쿼리 이러한 쿼리를 통해 불필요한 쿼리를 최소화하고 효율적인 데이터 검색을 수행할 수 있습니다.

```
poter3526 *
public interface BoardRepository extends JpaRepository<BoardEntity, Long> {
    poter3526 *
    @Cacheable("boardsByTitleContaining")
    List<BoardEntity> findByTitleContaining(String keyword);
    poter3526
    List<BoardEntity> findAll();
    poter3526
    List<BoardEntity> findByWriter(String writer);

    new *
    @Query("SELECT b FROM BoardEntity b WHERE b.title LIKE %:keyword%")
    List<BoardEntity> findCustomByTitleContaining(@Param("keyword") String keyword);
}
```

위의 코드에서 @Query 어노테이션을 사용하여 커스텀 쿼리를 사용하였으며, 이를 통해 "findByTitleContaining" 메서드보다 성능을 향상시킬 수 있습니다. "LIKE %: keyword%" 와 같은 사용자 지정 조건을 쿼리에 추가하여 특정 키워드를 포함하는 타이틀을 검색할 수 있습니다. 이를 통해 데이터베이스 쿼리를 더 효율적으로 최적화하고 필요한 데이터만을 검색할 수 있으며 성능을 향상시킬 수 있습니다.



Test 라는 키워드를 제목으로 가지고 있는 게시판 검색

```
L
{
  "createdAt": "2023-10-31T19:00:23.080942",
  "modifiedDate": "2023-10-31T19:00:23.080942",
  "id": 34,
  "writer": "painkiller",
  "title": "test",
  "content": "testcontent"
},
{
  "createdAt": "2023-10-31T19:00:29.156",
  "modifiedDate": "2023-10-31T19:00:29.156",
  "id": 35,
  "writer": "painkiller",
  "title": "test1",
  "content": "testcontent1"
}
]
```

JSON 형식으로 출력된 게시물

Controller 클래스

Controller 시스템: MVC(Model View Controller)의 디자인 패턴에서 모델 뷰를 분리하기 위한 양측 사이에 배치된 인터페이스 구현입니다.

@RestController는 Spring Framework에서 사용되는 어노테이션 중 하나로, 웹 애플리케이션에서 RESTful 웹 서비스를 개발할 때 사용되고, 데이터(API 응답)를 생성하는 데 중점을 둡니다.

회원 정보 저장을 위한 Controller

```
@RestController
@RequiredArgsConstructor
@Controller
public class UserController {
    private final UserService userService;

    @ResponseStatus(HttpStatus.CREATED)
    @PostMapping("/test/auth/register")
    public User register(@RequestBody RegisterDto registerDto) {
        System.out.println(registerDto);
        userService.register(registerDto);
        return null;
    }

    @GetMapping("/Get/Username")
    public String getUsername(@AuthenticationPrincipal PrincipalDetails principalDetails){
        String username;
        username=userService.getUsername(principalDetails.getUser());
        return username;
    }
}
```

1. 회원가입

http://localhost:8080/test/auth/register 을 통해 회원가입 실행

@ResponseStatus(HttpStatus.CREATED)를 사용하여 해당 컨트롤러 메서드에서 생성된 리소스에 대한 응답 상태 코드를 HTTP 201 Created로 지정합니다. 이것은 주로 POST 요청이나 다른 리소스 생성 작업에 사용됩니다. 이렇게 상태 코드를 설정하면 클라이언트와 서버 간의 의사 소통에 대한 정확성과 일관성을 유지됩니다.

회원 정보 서비스 기술 설명

1.회원정보 저장

```
public void register(RegisterDto registerDto) {  
    User user = new User();  
    user.setName(registerDto.getName());  
    user.setPassword(bCryptPasswordEncoder.encode(registerDto.getPassword()));  
    user.setUsername(registerDto.getUsername());  
    user.setRoles("ROLE_USER");  
    userRepository.save(user);  
}
```

등록하는 유저들의 정보를 Entity 객체로 그대로 받지 말고 안전을 위해 필요한 정보만을 위해 Dto로 받습니다.

Dto의 값을 User Entity 값에 넣어주고 User값을 userRepository를 통해 저장합니다.

또한 비밀번호는 스프링부트에서 제공하는 bCryptPasswordEncoder 메서드를 통해 암호 해시화 과정을 거쳐서 저장한다. 또한 이미 가입된 유저가 있을경우 똑같은 아이디를 입력해도 중복 가입되지 않습니다.

회원가입 데이터 전송전

	id	name	password	roles	username
	1	painkiller	\$2a\$10\$aFuffpA.TYfXD/TNBioOt...	ROLE_USER	adafafa@gmail.com
	2	drxman	\$2a\$10\$geYcM.9wtfCP3sV78Q7/B...	ROLE_USER	a@na.com
▶	6	NIY BE	\$2a\$10\$naeH9zQ/a1Y0rtcYktBW7...	ROLE_USER	poter3526@gmail.com
*	NULL	NULL	NULL	NULL	NULL

http://localhost:8080/test/auth/register 회원가입 데이터 전송

The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:8080/test/auth/register`
- Method: `POST`
- Body (JSON):

```
{  
  "username": "test@test.com",  
  "password": "testtest12",  
  "name": "simpletest"  
}
```

전송 이후 생겨난 유저 정보

	id	name	password	roles	username
	1	painkiller	\$2a\$10\$aFuffpA.TYFxD/TNBioOt...	ROLE_USER	adafafa@gmail.com
	2	drxman	\$2a\$10\$geYcM.9wtfCP3sV78Q7/B...	ROLE_USER	a@na.com
	5	testingman	\$2a\$10\$BWePynQkdkBNtxZ27u7...	ROLE_USER	testing@test.com
	6	NIY BE	\$2a\$10\$naeH9zQ/a1Y0rtcYktBW7...	ROLE_USER	poter3526@gmail.com
▶	7	simpletest	\$2a\$10\$dOFZpSVxHFMplhbbH7i1L...	ROLE_USER	test@test.com
*	NULL	NULL	NULL	NULL	NULL

비밀번호 또한 해시화 과정을 거쳐 저장됨

2. 해당 유저의 이름 가져오기

```
public String getUsername(User user) {  
    String Username;  
    Username = user.getName();  
    return Username;  
}
```

제공받은 JWT 토큰을 스프링부트 내부에 저장된 JWT 시크릿키를 통해 해석을 시도합니다. 만약 해석을 성공할 시 해당 유저의 이름을 반환할 수 있습니다.

게시판 Controller

```
@RestController
@RequestMapping(value = "/api/board", produces = MediaType.APPLICATION_JSON_VALUE)
public class BoardTestController {

    @Autowired
    private BoardService boardService;

    @GetMapping
    public List<BoardEntity> getAllPosts(){
        List<BoardEntity> posts = boardService.getAllBoards();
        return posts;
    }

    @GetMapping("/{id}")
    public ResponseEntity<BoardDTO> detail(@PathVariable Long id, Model model){
        BoardDTO boardDTO=boardService.getPost(id);
        if (boardDTO == null) {
            return ResponseEntity.notFound().build();
        }

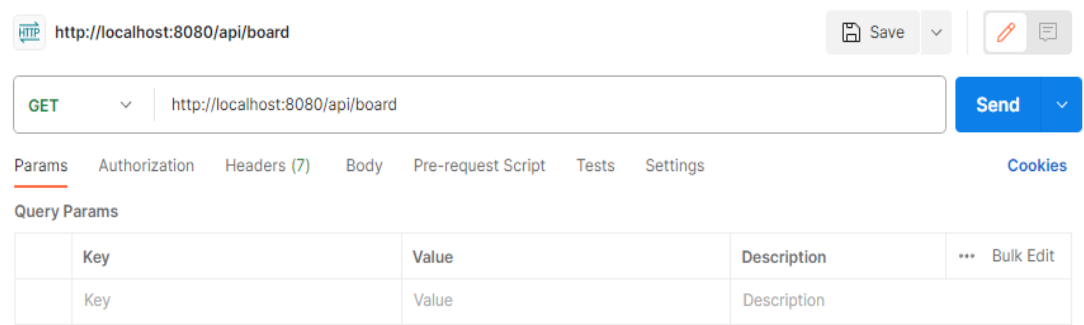
        return ResponseEntity.ok(boardDTO);
    }
}
```

게시판 기술 설명

1. 게시판 글 목록 가져오기

```
public List<BoardEntity> getAllBoards() {
    List<BoardEntity> posts = boardRepository.findAll();
    return posts;
}
```

boardRepository 를 통해 모든 게시글들을 List에다 저장해서 JSON형식으로 반환합니다.



<http://localhost:8080/api/board> 를 통해 게시판 가져오기

```
[
  {
    "createdDate": "2023-03-16T14:36:08.304162",
    "modifiedDate": "2023-03-16T15:59:25.612471",
    "id": 1,
    "writer": "테스트",
    "title": "테스트",
    "content": "테스트입니다"
  },
  {
    "createdDate": "2023-03-16T16:31:58.442845",
    "modifiedDate": "2023-03-16T16:31:58.442845",
    "id": 3,
    "writer": "아자르",
    "title": "아자르",
    "content": "아자르입니다"
  },
  {
    "createdDate": "2023-03-16T16:32:12.723738",
    "modifiedDate": "2023-03-16T16:32:12.723738",
    "id": 4,
    "writer": "람머스 ",
    "title": "람머스",
    "content": "람머스입니다"
  }
]
```

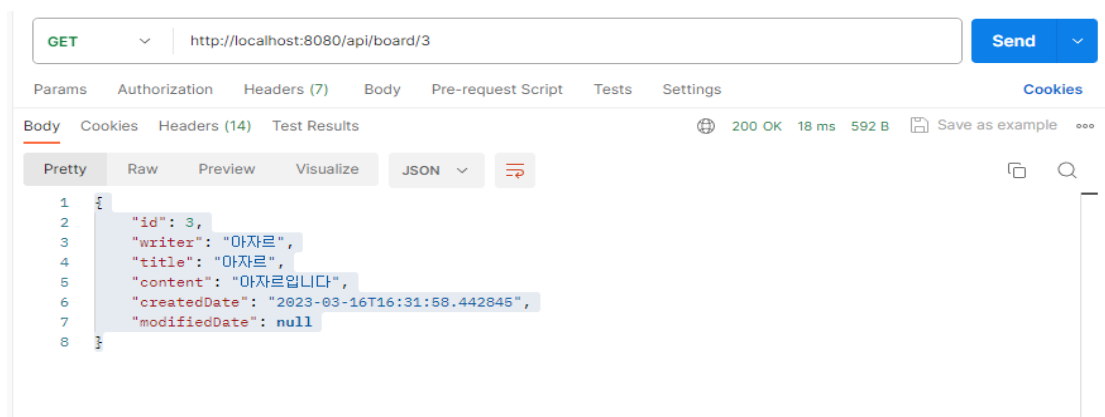
JSON 형식으로 반환된 게시물

2. 해당 글의 id 별로 글 상세내용 출력

```
@GetMapping("/{id}")
public ResponseEntity<BoardDTO> detail(@PathVariable Long id, Model model){
    BoardDTO boardDTO=boardService.getPost(id);
    if (boardDTO == null) {
        return ResponseEntity.notFound().build();
    }

    return ResponseEntity.ok(boardDTO);
}
```

링크에서 파라미터를 받아와 해당 게시글의 Dto를 받아와 ResponseEntity를 통해 해당 게시글의 상세 정보를 JSON형식으로 반환



3. 게시글 유저 확인

```
public boolean checkUser(Long id, User user) {  
    BoardEntity boardEntity = boardRepository.findById(id).get();  
    System.out.println(boardEntity.getWriter());  
    System.out.println(user.getName());  
    if(boardEntity.getWriter().equals(user.getName())){  
        return true;  
    }else{  
        return false;  
    }  
}
```

각 게시글 id 별로 현재 유저가 작성한 유저인지 JWT Token을 통해 확인 가능 확인을 통해 현재 글을 수정 또는 삭제가 가능하게 합니다

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/api/board/9/check
- Headers (8):**

Key	Value	Description
Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...	
Key	Value	Description
- Body:** true
- Status:** 200 OK, 31 ms, 453 B

로그인을 한 유저의 JWT Token을 통해 해당 유저가 작성한 건지 확인해줍니다. 현재 글은 해당 유저가 작성한 것이 맞으므로 true를 반환해줍니다.

HTTP http://localhost:8080/api/board/3/check

GET http://localhost:8080/api/board/3/check

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Headers 7 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...				
	Key	Value	Description			

Body Cookies Headers (14) Test Results 200 OK 23 ms 454 B Save as example

Pretty Raw Preview Visualize JSON

```
1 false
```

해당글은 현재유저가 작성한글이 아니므로 false를 반환해줍니다.

4.CRUD 시스템

CREATE

```
public void saveBoard(BoardEntity boardEntity, User user) {
    boardEntity.setWriter(user.getName());
    boardRepository.save(boardEntity);
}
```

입력 받은 게시글 Entity등과 JWT Token을 통해 받은 유저정보를 boardRepository를 통해 게시글 DB 에 저장합니다.

Result Grid						
Filter Rows:						
	id	created_date	modified_date	content	title	writer
	14	2023-04-21 01:01:35.648405	2023-04-21 01:01:35.648405	=====	=====	painkiller
	15	2023-04-21 01:03:06.160561	2023-04-21 01:03:06.160561	=====	=====	painkiller
	16	2023-04-21 01:03:12.758496	2023-04-21 01:03:12.758496	=====	=====	painkiller
	17	2023-04-21 01:03:18.200792	2023-04-21 01:03:18.200792	=====	=====	painkiller
	18	2023-04-21 01:04:09.351641	2023-04-21 01:04:09.351641	=====	=====	painkiller
	19	2023-04-21 01:04:15.706790	2023-04-21 01:04:15.706790	=====	=====	painkiller
	20	2023-04-21 01:04:23.777619	2023-04-21 01:04:23.777619	=====	삭제 테스트	painkiller
	21	2023-04-21 01:04:30.406413	2023-04-21 01:04:30.406413	=====	=====	painkiller
	22	2023-04-21 01:04:37.560415	2023-04-21 01:04:37.560415	=====	=====	painkiller
	23	2023-04-21 01:04:44.660429	2023-04-21 01:04:44.660429	=====	=====	painkiller
	24	2023-04-21 01:33:37.821748	2023-04-21 01:33:37.821748	=====	=====	painkiller
	25	2023-04-21 01:33:44.851428	2023-04-21 01:33:44.851428	=====	=====	painkiller
	26	2023-04-21 01:33:51.350745	2023-04-21 01:33:51.350745	=====	=====	painkiller
	27	2023-04-21 01:34:02.669332	2023-04-21 01:34:02.669332	=====	=====	painkiller
	28	2023-04-21 01:34:10.063734	2023-04-21 01:34:10.063734	=====	=====	painkiller
	29	2023-04-21 01:34:16.306599	2023-04-21 01:34:16.306599	=====	=====	painkiller
*	NULL	NULL	NULL	NULL	NULL	NULL

만들기전 게시판 데이터베이스

The screenshot shows a REST client interface with the following details:

- URL: `http://localhost:8080/api/board/save`
- Method: `POST`
- Body (JSON):

```
1 {
2   "title": "테스트 제목",
3   "content": "테스트 내용입니다."
4 }
```

게시글 저장 전송

28	2023-04-21 01:34:16.306599	2023-04-21 01:34:16.306599	테스트 내용입니다.	테스트 제목	painkiller
29	2023-04-21 01:34:16.306599	2023-04-21 01:34:16.306599	테스트 내용입니다.	테스트 제목	painkiller
33	2023-10-25 23:00:15.185275	2023-10-25 23:00:15.185275	테스트 내용입니다.	테스트 제목	painkiller

데이터베이스에 저장된 글

Update

```
public BoardEntity updateBoard(Long boardId, BoardEntity boardDetails, User user) {
    BoardEntity boardEntity = boardRepository.findById(boardId).orElse(null);
    if(boardEntity != null){
        if(boardEntity.getWriter().equals(user.getName())){
            boardEntity.setTitle(boardDetails.getTitle());
            boardEntity.setContent(boardDetails.getContent());
            BoardEntity updatedBoard = boardRepository.save(boardEntity);
            return updatedBoard;
        }else{
            return null;
        }
    }else{
        return null;
    }
}
```

파라미터를 통해 받은 게시글 id를 이용해서 해당 게시글을 가져온다 그리고 JWT Token 을 이용해 유저정보와 게시글을 작성한 유저와 일치할 경우 해당 게시글을 수정할 수 있습니다.

28	2023-04-21 01:34:16.306599	2023-04-21 01:34:16.306599	테스트 내용입니다.	테스트 제목	painkiller
29	2023-04-21 01:34:16.306599	2023-04-21 01:34:16.306599	테스트 내용입니다.	테스트 제목	painkiller
33	2023-10-25 23:00:15.185275	2023-10-25 23:00:15.185275	테스트 내용입니다.	테스트 제목	painkiller

업데이트 되기 전 글

PUT http://localhost:8080/api/board/update/33 Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```

1 {
2   ... "title": "바꾼 테스트 제목",
3   ... "content": "바꾼 테스트 내용입니다."
4 }

```

업데이트 값 전송

33	2023-10-25 23:00:15.185275	2023-10-25 23:04:58.998308	바꾼 테스트 내용...	바꾼 테스트 ...	painkiller
----	----------------------------	----------------------------	--------------	------------	------------

업데이트 이후 글

DELETE

HTTP http://localhost:8080/api/board/delete/33 Save

PUT http://localhost:8080/api/board/delete/33 Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```

1 {
2   ... "title": "바꾼 테스트 제목",
3   ... "content": "바꾼 테스트 내용입니다."
4 }

```

삭제 요청 전송

28	2023-04-21 01:34:10.063734	2023-04-21 01:34:10.063734	=====...	=====	painkiller
29	2023-04-21 01:34:16.306599	2023-04-21 01:34:16.306599	=====...	=====	painkiller
*	NULL	NULL	NULL	NULL	NULL

해당 글 삭제

현재 글을 작성한 유저와 현재 유저가 다를 경우

```

board_table boardentit0_
where
  boardentit0_.id=?
게시글아이디 유저와 토큰 유저가 다릅니다

```

해당 안내문이 뜨고 글은 삭제되지 않습니다.

5. 해당 유저의 작성 글 출력

해당유저의 JWT Token을 받아 boardRepository를 통해 유저가 작성한 글을 List에 저장 후 JSON 형식으로 반환합니다.

GET

http://localhost:8080/api/board/Get/MyPage/Board

Send

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

Cookies

Headers

9 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...				
	Key	Value	Description			

Body

Cookies

Headers (14)

Test Results

200 OK

39 ms

4.79 KB

Save as example

Pretty

Raw

Preview

Visualize

JSON

```

1  {
2    "createdDate": "2023-04-14T12:55:06.903366",
3    "modifiedDate": "2023-04-20T13:11:40.847788",
4    "id": 9,
5    "writer": "painkiller",
6    "title": "e o 테스트",
7    "content": "테스트 입니다\n\n수정 테스트!!!!"
8  },
9  {
10   "createdDate": "2023-04-21T01:01:14.974308",
11   "modifiedDate": "2023-04-21T01:01:14.974308",
12   "id": 11,
13   "writer": "painkiller",
14   "title": "e e e e e",
15   "content": "e e e e e e e e e e e e e e e"
16 },
17

```

Comment Controller

```
@RestController
@RequestMapping("/api/comments")
public class CommentRestController {
    private final CommentService commentService;

    public CommentRestController(CommentService commentService) { this.commentService = commentService; }

    // 게시글에 해당하는 댓글 리스트 조회
    @GetMapping("/board/{boardId}")
    public List<CommentEntity> getCommentsByBoardId(@PathVariable Long boardId) {
        return commentService.getCommentsByBoardId(boardId);
    }

    @GetMapping("/{commentId}/check")
    public boolean checkComment(@PathVariable Long commentId, @AuthenticationPrincipal PrincipalDetails principalDetails){
        boolean check = commentService.checkUser(commentId, principalDetails.getUser());
        return check;
    }

    @GetMapping("/get/count/{boardId}")
    public long getCommentCountByBoard(@PathVariable Long boardId){
        long count = commentService.getCommentCountBoard(boardId);
        return count;
    }

    @PostMapping("/save/{boardId}")
    public CommentEntity save(@PathVariable Long boardId, @RequestBody CommentEntity commentEntity, @AuthenticationPrincipal PrincipalDetails principalDetails){
        commentService.saveComment(boardId, commentEntity, principalDetails.getUser());
        return null;
    }

    @DeleteMapping("/delete/{commentId}")
    public CommentEntity delete(@PathVariable Long commentId, @AuthenticationPrincipal PrincipalDetails principalDetails){
        commentService.deleteComment(commentId, principalDetails.getUser());
        return null;
    }

    @GetMapping("/get/MyPage/Comment")
    public List<CommentEntity> getMyPageBoard(@AuthenticationPrincipal PrincipalDetails principalDetails){
        List<CommentEntity> comments = commentService.getMyComments(principalDetails.getUser());
        return comments;
    }
}
```

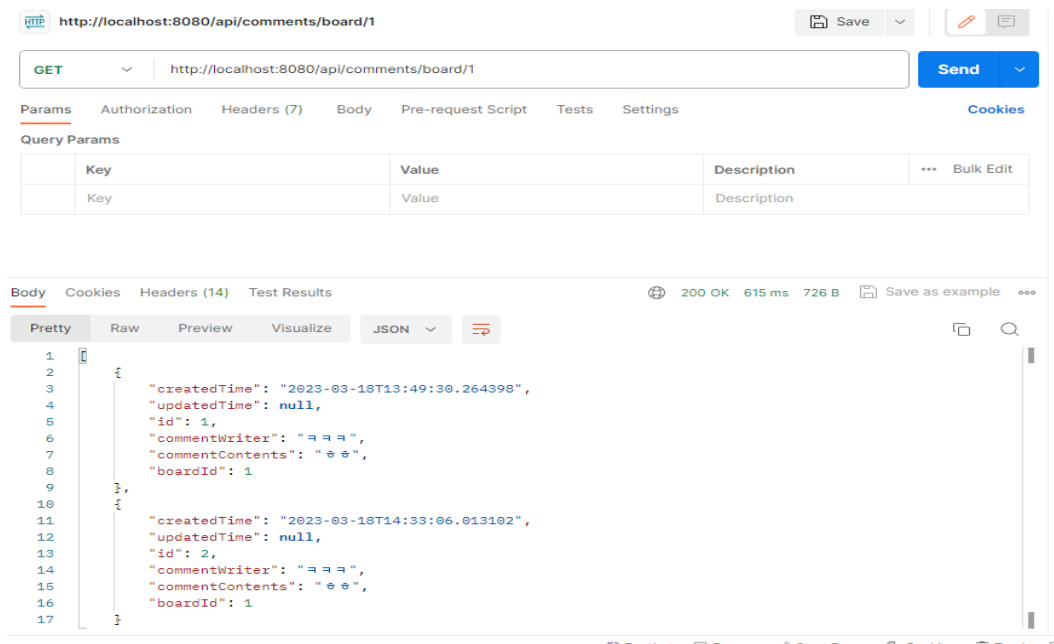
Windows 정품 인증
[설정]으로 이동하여 Windows를 정품 인증합니다.

Comment 서비스 기능

1. 게시글 별로 댓글 출력

```
public List<CommentDTO> findAll(Long boardId) {
    BoardEntity boardEntity = boardRepository.findById(boardId).get();
    List<CommentEntity> commentEntityList = commentRepository.findAllByBoardEntityOrderByDesc(boardEntity);
    List<CommentDTO> commentDTOList = new ArrayList<>();
    for (CommentEntity commentEntity : commentEntityList) {
        CommentDTO commentDTO = CommentDTO.toCommentDTO(commentEntity, boardId);
        commentDTOList.add(commentDTO);
    }
    return commentDTOList;
}
```

링크를 통해 게시글 파라미터 id 를 받아와 해당 게시글의 댓글들을 List 에 저장후 JSON 형식으로 반환해줍니다.



1 번 게시글의 저장된 댓글 목록을 출력한 상태입니다.

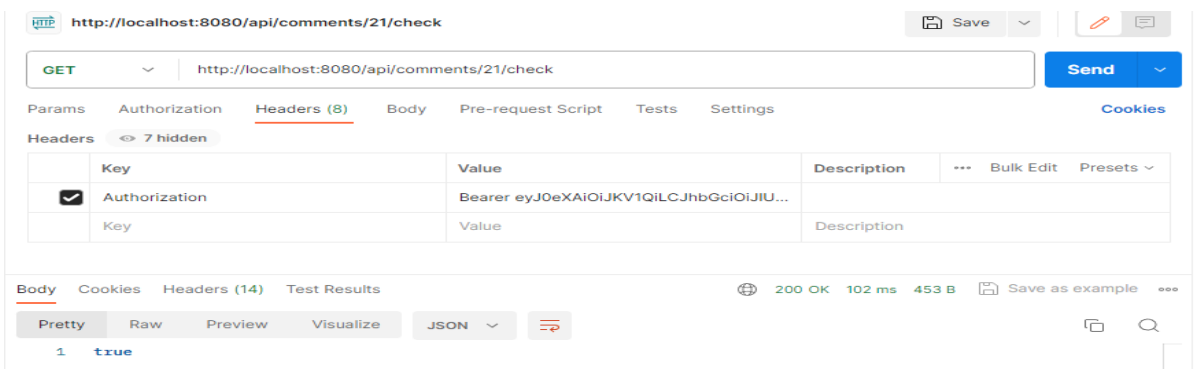
2. 현재 유저가 작성한 댓글인지 확인

```

public boolean checkUser(Long commentId, User user) {
    CommentEntity commentEntity = commentRepository.findById(commentId).get();
    if(commentEntity.getCommentWriter().equals(user.getName())){
        return true;
    }else{
        return false;
    }
}

```

JWT Token 을 통해 유저 정보를 받아 해당 유저가 작성한 댓글일 경우 댓글을 삭제할 수 있습니다.



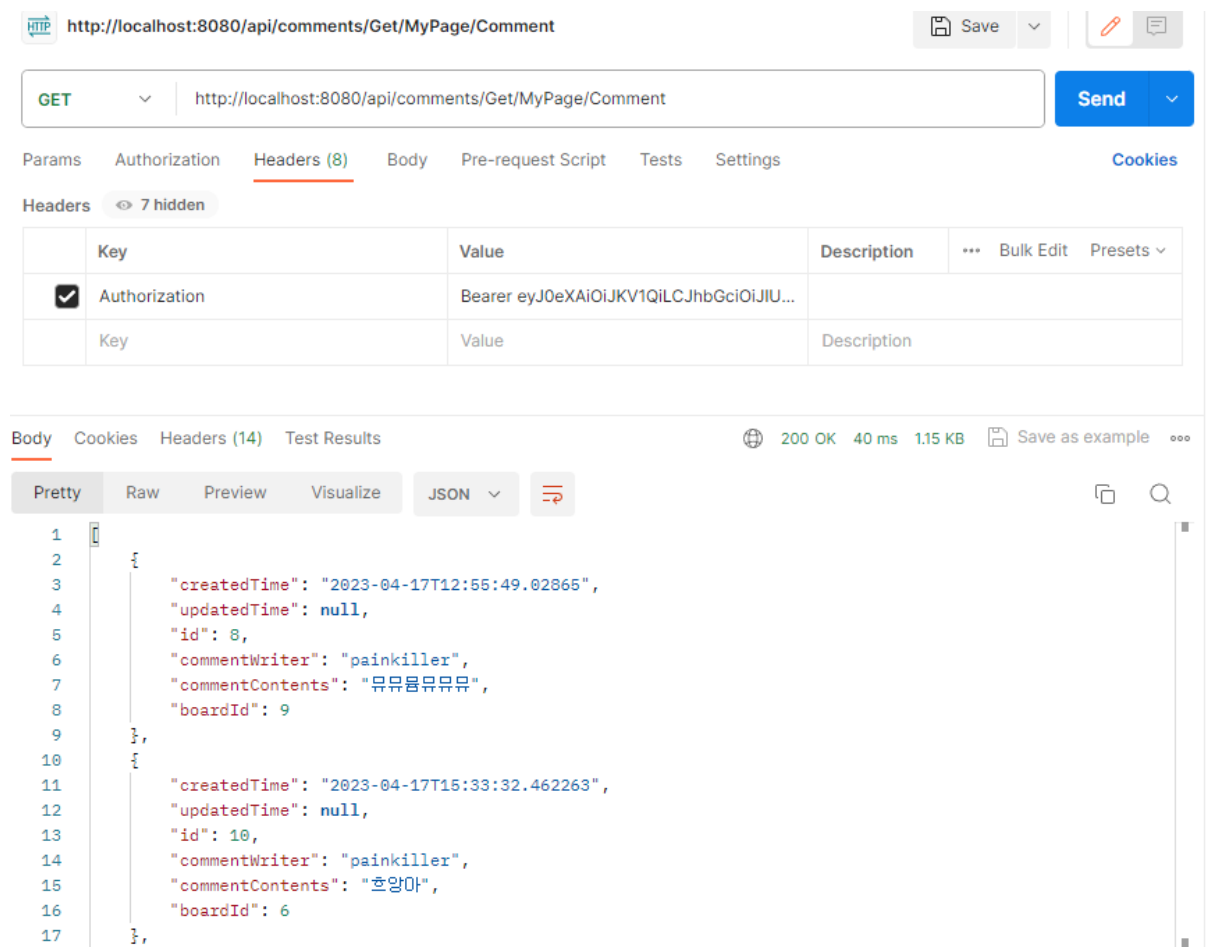
해당 유저 작성 글인지 확인 요구 결과 맞을 경우 true 를 반환해주었습니다.

3. 내가 작성한 댓글 목록 출력

```
public List<CommentEntity> getMyComments(User user) {
    String username= user.getName();
    List<CommentEntity> comments = commentRepository.findByCommentWriter(username);
    // 게시글의 ID를 포함하여 반환

    return comments;
}
```

JWT Token 을 통해 받은 유저의 이름을 commentRepository 를 통해 찾아주고 List 에 저장 후 JSON 형식으로 반환해주었습니다.



HTTP http://localhost:8080/api/comments/Get/MyPage/Comment

GET http://localhost:8080/api/comments/Get/MyPage/Comment

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Headers 7 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...				
	Key	Value	Description			

Body Cookies Headers (14) Test Results 200 OK 40 ms 1.15 KB Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "createdTime": "2023-04-17T12:55:49.02865",
4     "updatedTime": null,
5     "id": 8,
6     "commentWriter": "painkiller",
7     "commentContents": "뽕뽕뽕뽕뽕",
8     "boardId": 9
9   },
10  {
11    "createdTime": "2023-04-17T15:33:32.462263",
12    "updatedTime": null,
13    "id": 10,
14    "commentWriter": "painkiller",
15    "commentContents": "호앙아",
16    "boardId": 6
17  },
18 }
```

해당 유저의 댓글 목록의 출력을 확인

4. 댓글 삭제

해당 유저(painkiller)가 해당 댓글을 작성한 유저일시 삭제할 수 있는 권한이 생깁니다.

17	ㅋㅋ	painkiller	28	2023-04-21 13:20:17.06...	
20	ㅋㅋ	painkiller	29	2023-05-15 14:11:52.07...	NULL
21	삭제 테스트	painkiller	28	2023-05-30 13:42:34.60...	NULL
*	NULL	NULL	NULL	NULL	NULL

삭제 전 데이터베이스

DELETE ▼ **Send** ▼

Params Authorization **Headers (8)** Body Pre-request Script Tests Settings Cookies

Headers 7 hidden

	Key	Value	Description	...	Bulk Edit	Presets ▼
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...				
	Key	Value	Description			

삭제 요청 전송

	id	comment_contents	comment_writer	board_id	created_time	updated_time
	1	ㅎㅎ	ㅋㅋㅋ	1	2023-03-18 13:49:30.26...	NULL
	2	ㅎㅎ	ㅋㅋㅋ	1	2023-03-18 14:33:06.01...	NULL
	3	테테테테테	테테테테	3	2023-03-18 14:37:17.39...	NULL
	4	ㅋㅋㅋㅋ	ㅋㅋㅋㅋ	3	2023-03-18 14:37:36.92...	NULL
	5	댓글테스트	drxman	4	2023-04-15 10:11:47.71...	NULL
	7	댓글테스트2	drxman	4	2023-04-15 19:20:03.02...	NULL
	8	유유유유유유	painkiller	9	2023-04-17 12:55:49.02...	NULL
	10	후앙아	painkiller	6	2023-04-17 15:33:32.46...	NULL
	17	화이팅	painkiller	28	2023-04-21 13:20:17.06...	NULL
▶	20	ㅋㅋ	painkiller	29	2023-05-15 14:11:52.07...	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL

삭제 이후 데이터베이스

DELETE ▼ **Send** ▼

Params Authorization **Headers (8)** Body Pre-request Script Tests Settings Cookies

Headers 7 hidden

	Key	Value	Description	...	Bulk Edit	Presets ▼
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...				
	Key	Value	Description			

댓글 단 유저와 일치하지 않는 JWT Token을 넣을 경우 삭제되지 않습니다

```
comment_table commentent0_
where
commentent0_.id=?
여기까지 실행2
댓글아이디 유저와 토큰 유저가 다릅니다
```

Like Controller

```
public class LikeController {

    @Autowired
    private final LikeService likeService;

    @GetMapping("/check/{boardId}")
    public boolean checkComment(@PathVariable Long boardId, @AuthenticationPrincipal PrincipalDetails principalDetails){
        boolean check = likeService.checkUser(boardId, principalDetails.getUser());
        return check;
    }

    @GetMapping("/Get/Count/{boardId}")
    public long getLikeCountByBoard(@PathVariable Long boardId){
        long count = likeService.getLikeCountBoard(boardId);
        return count;
    }

    @GetMapping("/Get/MyPage/like")
    public List<Long> getMyPageLike(@AuthenticationPrincipal PrincipalDetails principalDetails){
        List<Like> likes = likeService.getMyLike(principalDetails.getUser());
        List<Long> boardIds = new ArrayList<>();
        for (Like like : likes) {
            boardIds.add(like.getBoardEntity().getId());
        }
    }
}
```

Like 서비스 기능

1. 현재 게시물의 좋아요 기능이 내가 누른 좋아요 기능인지 확인

```
public boolean checkUser(Long boardId, User user) {
    if (likeRepository.existsByBoardEntityIdAndUserId(boardId, user.getId())) {
        return true;
    } else {
        return false;
    }
}
```

JWT Token을 통해 발급받은 유저의 Id 값과 해당 게시글의 "좋아요"를 누른 유저의 Id 값과 일치하는지 일치할 경우 해당 "좋아요"를 다시 취소할 수 있습니다.

The screenshot shows a REST client interface with the following details:

- Method: GET
- URL: http://localhost:8080/like/check/29
- Headers (8):

Key	Value	Description
Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...	
Key	Value	Description
- Body: true
- Status: 200 OK, 96 ms, 453 B

일치할 경우 true값 반환

2. 좋아요 누르기 및 삭제

```
public void saveLike(Long boardId, Like like, User user) {
    BoardEntity boardEntity = boardRepository.findById(boardId).get();
    User getUser = userRepository.findById(user.getId()).get();
    if (likeRepository.existsByBoardEntityIdAndUserId(boardId, user.getId())) {
        System.out.println("이미 좋아요를 누른 게시판");
    } else {
        like.setUser(getUser);
        like.setBoardEntity(boardEntity);
        likeRepository.save(like);
        System.out.println("저장완료");
    }
}

public void deleteLike(Long boardId, User user) {
    Like like = likeRepository.findByBoardEntityIdAndUserId(boardId, user.getId());
    if (likeRepository.existsByBoardEntityIdAndUserId(boardId, user.getId())) {
        likeRepository.delete(like);
        System.out.println("좋아요 삭제");
    } else {
        System.out.println("존재하지 않는 좋아요");
    }
}
```

	heart_id	board_id	user_id
▶	1	29	1
	2	29	2
	3	28	2
	9	27	1
	10	16	1
*	NULL	NULL	NULL

좋아요 누르기 전 현재 좋아요 목록

POST

http://localhost:8080/like/save/29

Send

ParamsAuthorizationHeaders (9)BodyPre-request ScriptTestsSettingsCookies

Headers8 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...				
	Key	Value	Description			

좋아요 전송

	heart_id	board_id	user_id
▶	1	29	1
	2	29	2
	3	28	2
	9	27	1
	10	16	1
	11	15	1

누른 후 15번 게시글의 “좋아요”

DELETE
http://localhost:8080/like/delete/15
Send

Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies

Headers
7 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...				
	Key	Value	Description			

좋아요 삭제 전송

	heart_id	board_id	user_id
▶	1	29	1
	2	29	2
	3	28	2
	9	27	1
	10	16	1
*	NULL	NULL	NULL

삭제 후 좋아요 DB 테이블

- 내가 “좋아요”를 누른 게시글 id 출력

```

public Long getBoardIdByMyLike(Long likeId) {
    Like like = likeRepository.findById(likeId).orElse( other: null);
    Long boardId = like.getBoardId();
    return boardId;
}

```

JWT Token을 통해 얻은 현재유저의 id를 통해 현재 유저가 “좋아요”를 누른 게시글을 LikeRepository를 이용해서 찾을 수 있습니다.

GET

http://localhost:8080/like/Get/MyPage/like

Send

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettingsCookies

Headers

7 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIU...				
	Key	Value	Description			

BodyCookiesHeaders (14)Test Results

200 OK48 ms459 BSave as example

PrettyRawPreviewVisualizeJSON

1

2

3

4

5

[

29,

27,

16

]

내가 "좋아요"를 누른 게시글 id를 JSON 형식으로 반환

Database Cache

@Cacheable: 스프링에서 제공하는 캐싱 기능을 활용하여 메서드의 결과를 캐시에 저장하고, 이후 동일한 메서드 호출에 대한 결과를 캐시에 반환하는데 사용되는 어노테이션입니다

```
poter3526 *  
@Cacheable("boardsByTitleContaining")  
List<BoardEntity> findByTitleContaining(String keyword);
```

이러한 캐시를 사용함으로써 얻는 효과

1. @Cacheable 어노테이션을 사용하면 메서드의 결과를 캐시에 저장합니다. 동일한 메서드가 반복적으로 호출될 때 매번 데이터베이스 조회를 수행하지 않고, 이전에 저장한 캐시에서 결과를 바로 반환합니다.
2. 캐시를 사용함으로써 데이터의 무결성을 보다 쉽게 유지할 수 있습니다.

2.

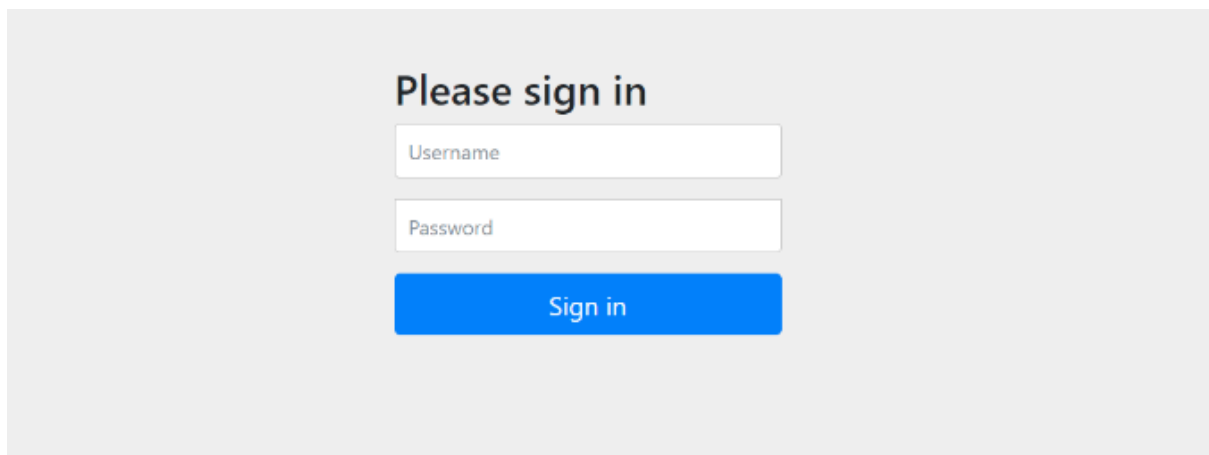
스프링부트 보안 설정

스프링 시큐리티 설정

Spring Security: 스프링 시큐리티는 인증(Authentication), 권한(Authorize) 등을 부여 및 보호 기능을 제공하는 프레임워크입니다.

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

build gradle를 통해 스프링 시큐리티 프레임워크를 가져옵니다.

A screenshot of a web browser showing a login page. The page has a light gray background. At the top, the text "Please sign in" is displayed in a dark font. Below this text are two input fields: the first is labeled "Username" and the second is labeled "Password". Both fields are white with a thin gray border. Below the password field is a blue button with the text "Sign in" in white.

이와 같이 가져오고 <http://localhost:8080/login> 화면이 출력되면 시큐리티가 작동하는 것을 확인

스프링 시큐리티 필터 클래스

```
@Configuration
@EnableWebSecurity // 시큐리티 활성화 -> 기본 스프링 필터체인에 등록
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter{

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private CorsConfig corsConfig;

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .addFilter(corsConfig.corsFilter()) // HttpSecurity
            .csrf().disable()
    }
}
```

@EnableWebSecurity: 해당 클래스를 스프링 시큐리티의 기본 시큐리티 체인으로 만들어줍니다.

이 서버로 접근 시 이 필터를 통과해야 데이터에 접근할 수 있습니다.

Csrf().disable(): Spring Security의 Cross-Site Request Forgery (CSRF) 공격 방어 기능을 비활성화하는 설정입니다.

CSRF 공격은 사용자가 의도하지 않았지만 공격자의 의도에 따라 공격 대상 서버를 공격하게 되는 공격 유형 중 하나입니다. 주로 쿠키 기반의 인증 방식을 사용할 때 발생할 수 있으며, Spring Security는 기본적으로 이러한 CSRF 방어 기능을 활성화하고 있습니다.

그러나 REST API와 같이 세션과 쿠키를 사용하지 않는 경우에는 CSRF 방어를 사용할 필요가 없을 수 있습니다. 따라서 이 설정을 사용하여 CSRF 방어를 비활성화하면 REST API와 같은 상황에서 불필요한 CSRF 토큰을 요구하지 않게 됩니다.

필터 1단계: Corsfilter

corsFilter 설정

CORS Policy: 출처가 서로 다른 자원들을 공유하 자원들을 공유한다는 뜻으로 브라우저가 리소스 로드를 허용해야 하는 자체 출처 이외의 모든 출처 (도메인, 스키마 또는 포트)를 서버가 표시할 수 있도록 하는 정책.

스프링부트를 통해서 프론트엔드 서버와의 통신을 자동으로 설정할 경우 충돌이 일어나 통신의 장애가 생겨서 통신을 제대로 할 수가 없습니다.

```
@Configuration
public class CorsConfig {

    @Bean
    public CorsFilter corsFilter() {
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowCredentials(true);
        config.addAllowedOriginPattern("http://localhost:3000"); // http://localhost:3000 도메인만 요청을 허용

        config.addAllowedHeader("*");
        config.addAllowedMethod("*"); |
        config.addExposedHeader(JwtProperties.HEADER_STRING);

        source.registerCorsConfiguration( pattern: "**", config);
        return new CorsFilter(source);
    }
}
```

이러한 통신을 원활하게 하기 위해 직접 CORS Filter 클래스를 작성해주었습니다.

Config.addAllowedOriginPattern('http://localhost:3000'): 로컬에서 사용되고 있는 프론트 서버 (**http://localhost:3000**)에 대한 응답만 허용입니다. (특정 프론트 ip 도메인만 허용하는 이유는 다른 ip가 민감한 데이터에 접근해 탈취당할 수 있기 때문에 보안 문제가 발생한다.)

Config.addAllowedHeader("*"): 모든 header에 응답을 허용(현재 테스트 과정이기 때문에 모든 ip에 대한 응답을 허용하였다.)

Config.addAllowedMethod ("*"): 모든 post, get, put, delete, patch 요청을 허용(현재 테스트 과정이기 때문에 모든 IP에 대한 응답을 허용하였다.)

Config.addExposedHeader(JWTProperties.HEADER_STRING): 브라우저에서 JavaScript로 작성된 코드가 JWT를 읽을 수 있게 됩니다.

필터 2단계: JWTAutheticationFilter

<http://localhost:8080/login> 을 통해 로그인시 UsernamePasswordAuthenticationFilter 인터페이스의 attemptAuthentication 함수와 successfulAuthentication 함수가 실행

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

    System.out.println("JwtAutheticationFilter : 진입");
    try {
        // request에 있는 username과 password를 파싱해서 자바 Object로 받기
        ObjectMapper om = new ObjectMapper();
        User user = om.readValue(request.getInputStream(), User.class);
        System.out.println(user);

        UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(user.getUsername(), user.getPassword());

        //principalDetailsService의 loadUserByUsername() 함수가 실행됨
        Authentication authentication = authenticationManager.authenticate(authenticationToken);

        //authentication 객체가 session 영역에 저장됨. => 로그인이 되었다는 뜻
        PrincipalDetails principalDetails = (PrincipalDetails) authentication.getPrincipal();
        System.out.println("로그인 완료됨:" + principalDetails.getUser().getUsername());
    }
}
```

해당 attemptAuthentication 인터페이스의 함수를 @Override를 통해 재정의

Post를 통해 받은 username과 password를 통해 authenticationToken 객체에 인자값으로 전달

이러한 객체를 authenticationManager에 전달할 경우 principalDetailsService의 함수가 실행됨

```
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    System.out.println("PrincipalDetailsService의 loadUserByUsername()");
    User userEntity = userRepository.findByUsername(username);
    System.out.println("userEntity:" + userEntity);
    return new PrincipalDetails(userEntity);
}
```

principalDetailsService 실행시 userRepository를 통해 해당 유저가 존재여부를 확인. 존재가 성공적으로 확인 시 successfulAuthentication 함수에 authentication 객체를 전달 만약 유저가 존재 안 할 경우 전달을 하지 않습니다

JWT TOKEN 생성

JWT:JWT는 유저를 인증하고 식별하기 위한 토큰(Token) 기반 인증이다. 토큰 자체에 사용자의 권한 정보나 서비스를 사용하기 위한 정보가 포함됩니다.

successfulAuthentication에 authentication의 객체 전달이 성공하면 JWTToken을 생성합니다.

```
protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response, FilterChain chain,
    Authentication authResult) throws IOException, ServletException {
    System.out.println("successfulAuthentication 실행됨: 인증이 완료되었다는 뜻임.");
    PrincipalDetails principalDetails = (PrincipalDetails) authResult.getPrincipal();
```

authentication 객체를 PrincipalDetails 객체로 형변환 해줍니다.

```
String jwtToken = JWT.create()
    .withSubject(principalDetails.getUsername())
    .withExpiresAt(new Date(System.currentTimeMillis()+JwtProperties.EXPIRATION_TIME))
    .withClaim( name: "id", principalDetails.getUser().getId())
    .withClaim( name: "username", principalDetails.getUser().getUsername())
    .sign(Algorithm.HMAC512(JwtProperties.SECRET));

System.out.println(jwtToken);

Cookie cookie = new Cookie("jwtToken",jwtToken);
cookie.setHttpOnly(true);

response.addCookie(cookie);
response.addHeader(JwtProperties.HEADER_STRING, value: JwtProperties.TOKEN_PREFIX+jwtToken);
```

해당 객체를 형변환 할 경우 로그인 한 유저의 id와 username값을 토큰에 넣어줍니다.

```
public interface JwtProperties {

    String SECRET = "luckywater";
    int EXPIRATION_TIME = 60000*60;
    String TOKEN_PREFIX = "Bearer ";
    String HEADER_STRING = "Authorization";

}
```

SECRET: JWT토큰은 서명을 생성하고 검증하는데 사용되는 시크릿 키

Bearer: HTTP 요청 헤더에 JWT를 전달하고 서버는 이를 사용하여 클라이언트의 인증을 수행합니다.

로그인 정보 Username과 password <http://localhost:8080/login> 으로 전달 시

Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZGF...
---------------	----------------------------------------------------------------

JWT Token 값이 생성되는 것을 확인할 수 있습니다.

필터 3단계: JWTAuthorizationFilter

```
public class JwtAuthorizationFilter extends BasicAuthenticationFilter {  
  
    private UserRepository userRepository;  
  
    public JwtAuthorizationFilter(AuthenticationManager authenticationManager, UserRepository userRepository) {  
        super(authenticationManager);  
        this.userRepository = userRepository;  
    }  
}
```

JWT 토큰으로 서비스 시스템 인증 시 해당 필터가 실행된다. (게시글, 댓글, 좋아요 등록 수정 삭제 기능을 위해 필요)

```
String username = JWT.require(Algorithm.HMAC512(JwtProperties.SECRET)).build().verify(token)  
    .getClaim("username").asString();
```

JWTToken은 헤더의 Authorization을 통해 백엔드 서버로 전달할 수가 있다. 전송받은 JWT Token 값을 시크릿키를 활용해 username을 JWT Token으로부터 추출합니다.

```
if (username != null) {  
    User user = userRepository.findByUsername(username);  
  
    PrincipalDetails principalDetails = new PrincipalDetails(user);  
    Authentication authentication = new UsernamePasswordAuthenticationToken(  
        principalDetails,  
        credentials: null,  
        principalDetails.getAuthorities());  
  
    SecurityContextHolder.getContext().setAuthentication(authentication);  
}  
  
chain.doFilter(request, response);
```

해당 username이 null 값이 아닐 시 해당 유저를 userRepository에서 찾은 후 authentication 객체에 값을 전달 후 해당 유저의 존재여부 및 권한 등을 시큐리티의 세션에 접근하여 저장한다.

OAuth2.0

OAuth2.0: OAuth 2.0(Open Authorization 2.0, OAuth2)란 표준 인증 프로토콜이다. 다른 웹사이트에서 제공하는 API를 이용하여 웹사이트나 애플리케이션의 접근 권한을 부여 할 수 있는 공통적인 수단으로써 사용됩니다.

구글 클라우드에서 OAuth2.0 api를 가져온다.

이름 *

OAuthTest

OAuth 2.0 클라이언트의 이름입니다. 이 이름은 콘솔에서 클라이언트를 식별하는 용도로만 사용되며 최종 사용자에게 표시되지 않습니다.

1

아래에 추가한 URI의 도메인이 [승인된 도메인](#)으로 OAuth 동의 화면에 자동으로 추가됩니다.

승인된 자바스크립트 원본 ?

브라우저 요청에 사용

+ URI 추가

승인된 리디렉션 URI ?

웹 서버의 요청에 사용

URI 1 *

http://localhost:8080/check/oauth2/code/google

+ URI 추가

참고: 설정이 적용되는 데 5분에서 몇 시간이 걸릴 수 있습니다.

저장 취소

Additional information

클라이언트 ID	<div></div>		
생성일	2023년 10월 24일 PM 2시 12분 0초 GMT+9		
클라이언트 보안 비밀번호			
클라이언트 보안 비밀번호를 변경하는 중인 경우 다운타임 없이 수동으로 순환할 수 있습니다. 자세히 알아보기			
클라이언트 보안 비밀번호	<div></div>		
생성일	2023년 10월 24일 PM 2시 12분 0초 GMT+9		
상태	✔ 사용 설정됨		

```
implementation 'org.springframework.boot:spring-boot-starter-oauth2-client'
```

Build gradle을 통해 OAuth 라이브러리를 가져온다.

```
#OAuth2 Google
spring.security.oauth2.client.registration.google.client-id=
spring.security.oauth2.client.registration.google.client-secret=
spring.security.oauth2.client.registration.google.redirect-uri= http://localhost:8080/check/oauth2/code/google
spring.security.oauth2.client.registration.google.token-uri=https://oauth2.googleapis.com/token
spring.security.oauth2.client.registration.google.resource-uri = https://www.googleapis.com/oauth2/v2/userinfo
```

구글 OAuth api로부터 받은 클라이언트 id와 클라이언트 시크릿키를 application.properties에 정확히 적어줍니다.

Redirect-uri: OAuth를 통해 로그인 후 스프링부트 로그인 컨트롤러로 이동할수 있게 하는 url 을 나타냅니다.

OAuth 로그인 서비스

```
private String getAccessToken(String authorizationCode, String registrationId) {
    String clientId = env.getProperty("spring.security.oauth2.client.registration." + registrationId + ".client-id");
    String clientSecret = env.getProperty("spring.security.oauth2.client.registration." + registrationId + ".client-secret");
    String redirectUri = env.getProperty("spring.security.oauth2.client.registration." + registrationId + ".redirect-uri");
    String tokenUri = env.getProperty("spring.security.oauth2.client.registration." + registrationId + ".token-uri");

    MultiValueMap<String, String> params = new LinkedMultiValueMap<>();
    params.add("code", authorizationCode);
    params.add("client_id", clientId);
    params.add("client_secret", clientSecret);
    params.add("redirect_uri", redirectUri);
    params.add("grant_type", "authorization_code");

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);

    HttpEntity entity = new HttpEntity(params, headers);

    ResponseEntity<JsonNode> responseNode = restTemplate.exchange(tokenUri, HttpMethod.POST, entity, JsonNode.class);
    JsonNode accessTokenNode = responseNode.getBody();
    return accessTokenNode.get("access_token").asText();
}
```

로그인 후 구글로부터 AccessToken을 발급받는 기능이다. OAuth로그인을 성공할 경우 ApplicationProperties에 적은 url에 접근할 수 있습니다.

해당 url에 접근 후 user에 email id 닉네임등 각종 개인정보를 가져올수 있습니다. 해당 정보로 회원가입이나 로그인이 가능하게 합니다.

OAuth 회원가입 및 로그인 Controller

```
@GetMapping("/code/{registrationId}")
public void googleLogin(@RequestParam String code, @PathVariable String registrationId, HttpServletResponse response) {
    RegisterDto googleuser = loginService.socialLogin(code, registrationId);
    System.out.println("code"+code);
    User checkuser = userRepository.findByUsername(googleuser.getUsername());

    if(checkuser==null){
        System.out.println("기존의 회원 아님");
        userService.register(googleuser);
        checkuser.setPassword(googleuser.getPassword());
    }
}
```

OAuth 로그인 후 <http://localhost:8080/check/oauth2/code/google> 로 리디렉션 됩니다.

리디렉션 이후 만약 유저가 DB테이블에 존재 하지 않을 경우 유저의 정보를 받아 회원정보에 등록해줍니다.

```
private final String garbagepw = "52ce680557feaf8dc64088c33703d2d9cfbe0c05801d18ff817caaaaa3029ad6";
```

비밀번호는 의미 없는 비밀번호를 직접 설정해 입력해주었습니다. (1차적으로 구글 OAuth를 통해 로그인 하기 때문에 따로 비밀번호가 필요 없지만 null값이 들어가면 안되므로 사용하였다.)

```
Authentication authentication = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(googleuser.getUsername(), googleuser.getPassword()));
SecurityContextHolder.getContext().setAuthentication(authentication);

PrincipalDetails principalDetails = (PrincipalDetails) authentication.getPrincipal();

String jwtToken = JWT.create()
    .withSubject(principalDetails.getUsername())
    .withExpiresAt(new Date(System.currentTimeMillis()+ JwtProperties.EXPIRATION_TIME))
    .withClaim( name: "id", principalDetails.getUser().getId())
    .withClaim( name: "username", principalDetails.getUser().getUsername())
    .sign(Algorithm.HMAC512(JwtProperties.SECRET));

System.out.println(jwtToken);
```

로그인 이후 username과 password을 authentication 객체의 인자로 전달합니다.

이후 authentication 객체를 PrincipalDetails로 형변환 한 후 해당 객체를 이용해 JWT Token을 발급합니다.

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	id	name	password	roles	username
▶	1	painkiller	\$2a\$10\$aFuffpA.TYfXD/TNBioOt...	ROLE_USER	adafafa@gmail.com
	2	drxman	\$2a\$10\$geYdM.9wtFCP3sV78Q7/B...	ROLE_USER	a@na.com
	5	testingman	\$2a\$10\$BWePynQkdkIBNtxZ27u7...	ROLE_USER	testing@test.com
	6	NIY BE	\$2a\$10\$naeH9zQ/a1Y0rtcYktBW7...	ROLE_USER	poter3526@gmail.com
	7	simpletest	\$2a\$10\$dOFZpSVxHFMplhbbH71L...	ROLE_USER	test@test.com
*	NULL	NULL	NULL	NULL	NULL

로그인 전 유저 DB 테이블

코드: <https://github.com/Ideadlysinx/RegisterCRUDBackend/tree/master>