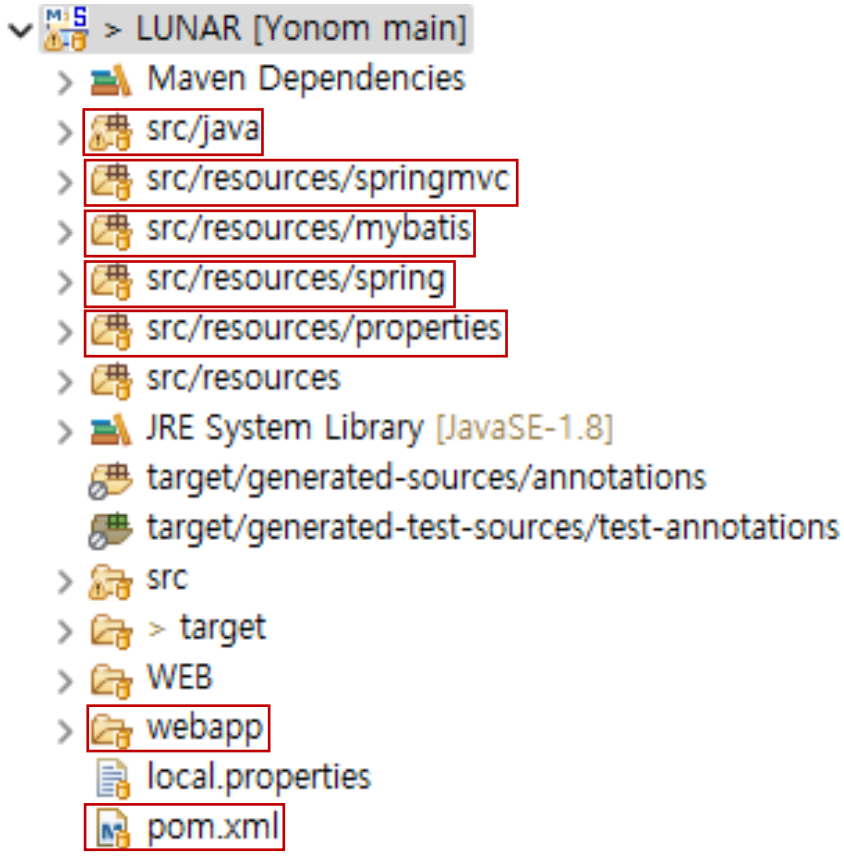
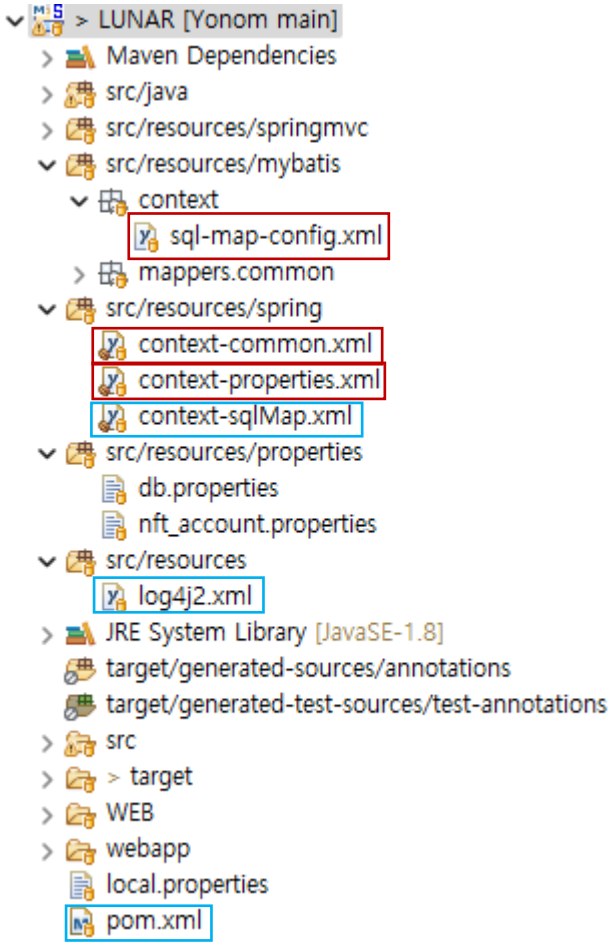


1. **src/java :** 자바 소스 코드가 있는 폴터입니다.
2. **src/resources/springmvc :** 디스패처-서블릿(서버로 들어오는 클라이언트의 모든 요청을 받아서 처리해주는 컨트롤러)가 있는 폴터입니다.
3. **src/resources/mybatis :** 마이바티스 설정 파일 및 SQL 파일이 있는 폴터입니다.
4. **src/resources/spring :** 프레임워크의 설정 파일이 있는 폴터입니다.
5. **src/resources/properties :** 프로퍼티스(중요한 정보를 소스 코드와 분리) 파일이 있는 폴터입니다.
6. **webapp :** View(JSP, HTML, etc..)와 라이브러리, web.xml 파일이 있는 폴터입니다.
7. **pom.xml :** 프레임워크의 설정 및 메이븐 (라이브러리 관리)를 할 수 있는 파일입니다.



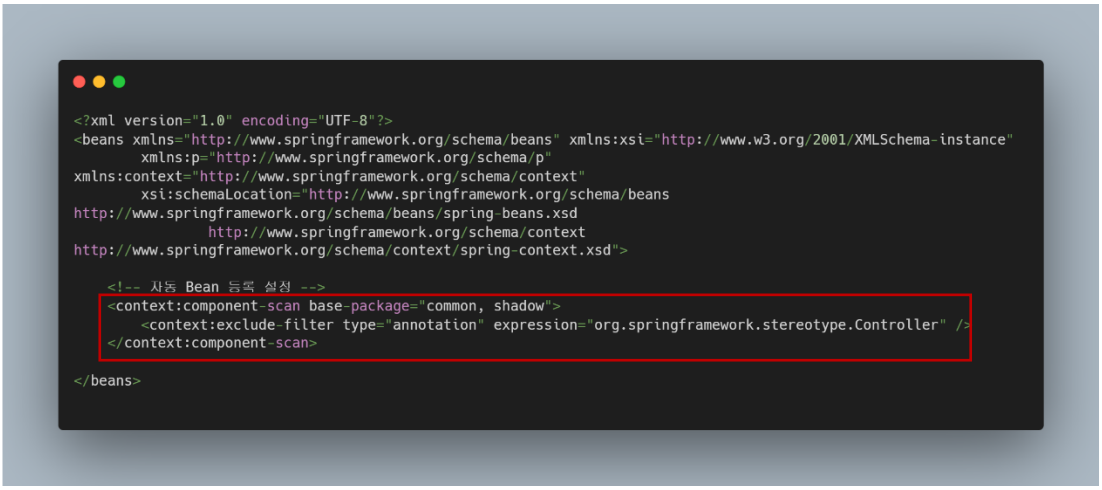
1. **sql-map-config.xml : 마이바티스를 설정할 수 있는 파일입니다.**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config3.0//EN"
    "http://www.mybatis.org/dtd/mybatis-3-config.dtd">

<!-- IBATIS RESOURCE CONFIGURATION -->
<configuration>
    <settings>
        <setting name="jdbcTypeForNull" value="VARCHAR" />
        <setting name="callSettersOnNulls" value="true" />
        <setting name="returnInstanceForEmptyRow" value="true"/>
</settings>

    <mappers>
        <mapper resource="mybatis/mappers/common/common_mapper.xml" />

    </mappers>

</configuration>
```

Mapper 파일을 추가적으로 만드실 때, <mappers> 내부에

`<mapper resource="파일 위치.xml/>`로 등록해주세요.

2. **Context-common.xml : 파일 내부의 Bean (@Autowired, @Component, @Controller, @Service, etc..)을 자동적으로 찾아 등록해주는 설정 파일입니다.**



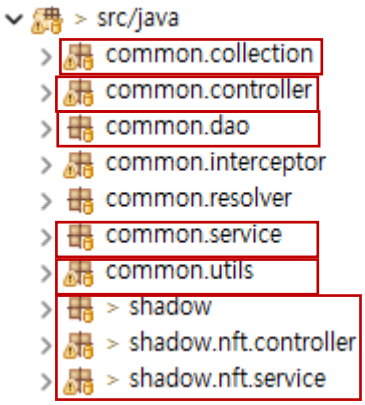
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <!-- 자동 Bean 등록 설정 -->
    <context:component-scan base-package="common, shadow">
        <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller" />
    </context:component-scan>

</beans>
```

새로운 패키지를 등록하실 때, `<context:component-scan base-package="common, shadow, 서브 디렉토리 이름">`

의 base-package에 서브 디렉토리의 이름을 입력해주세요.

src/resources/spring/mvc/dispatcher-servlet.xml 파일에서도 위와 같은 방법으로 입력해주세요.

3. **context-properties.xml : 프로퍼티스를 등록해주는 설정 파일입니다.** src/resources/properties 아래의 .properties 확장자의 파일들을 등록해줍니다. 코드에 중요한 정보를 포함시킬 때 주로 사용합니다.



1. **common.collection** : 사용자 정의한 자료구조 및 Log, session를 정의한 파일을 담고 있는 폴터입니다. 사용자 정의한 자료구조는 해쉬 맵(String, Object)을 상속한 ABox와 리스트(String, Object)를 상속한 ABoxList이 있습니다.

   Json 관련 기능 : toJson(), set(String), setJson(String), jsonToABox(JsonObject), aBoxToJson() 등의 기능이 있습니다.

   기본 기능 : set(String, Object), get(String), get(int)

   편의 기능 : toString(), isEmpty() 등이 있습니다.

2. **common.controller** : Controller의 부모 클래스이며, 컨트롤러에 필요한 기능들을 포

함하고 있습니다. 컨트롤러를 만드실 때 상속해주세요.

3. **common.dao** : 데이터 조회를 위한 구현 메소드입니다. 마이바티스 주소와 Object로 구성되어있습니다.

   기본 기능 : insert(String, ABox), update(String, ABox)

   selectList(String, ABox), select(String, ABox)

   delete(String, ABox)

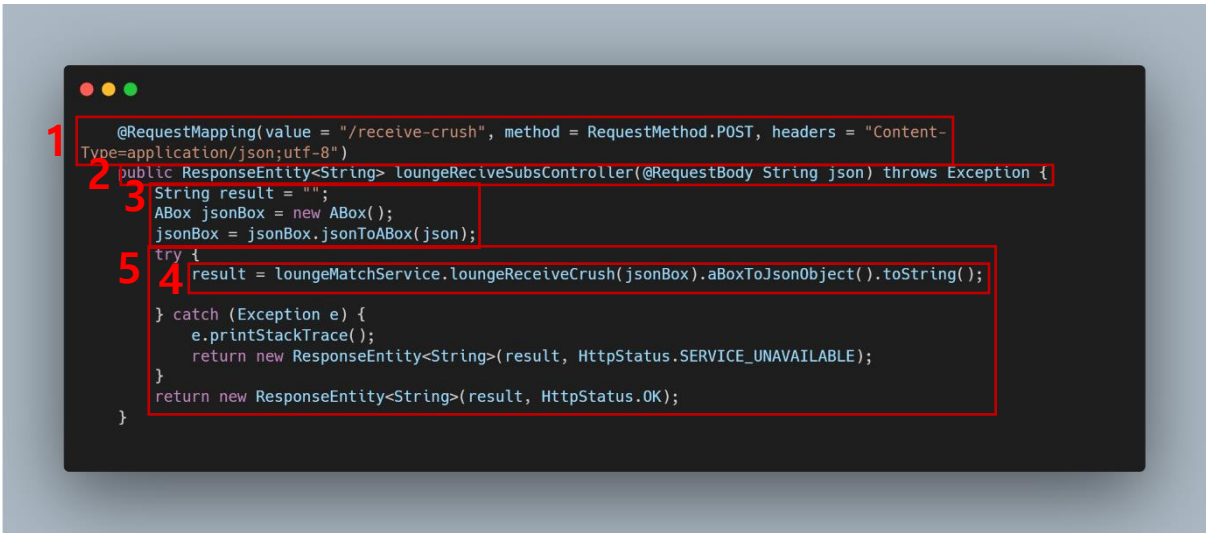
4. **common.service** : Service의 부모 클래스이며, 서비스에 필요한 기능들을 포함하고 있습니다. 서비스를 만드실 때 상속해주세요.

5. **common.utils** : Service의 기능에 필요한 파일들을 모아둔 유틸 폴터입니다.

6. **shadow(임의 변경 가능)** : MVC 패턴을 적용한 서브 디렉토리입니다. 새로운 서브 디렉토리를 만드실 때, controller 폴터와 service의 폴터를 만들어주세요. Service 파일은 인터페이스인 *Service.java와 인터페이스 구현 파일인 *ServiceImpl로 구성되어 있습니다.

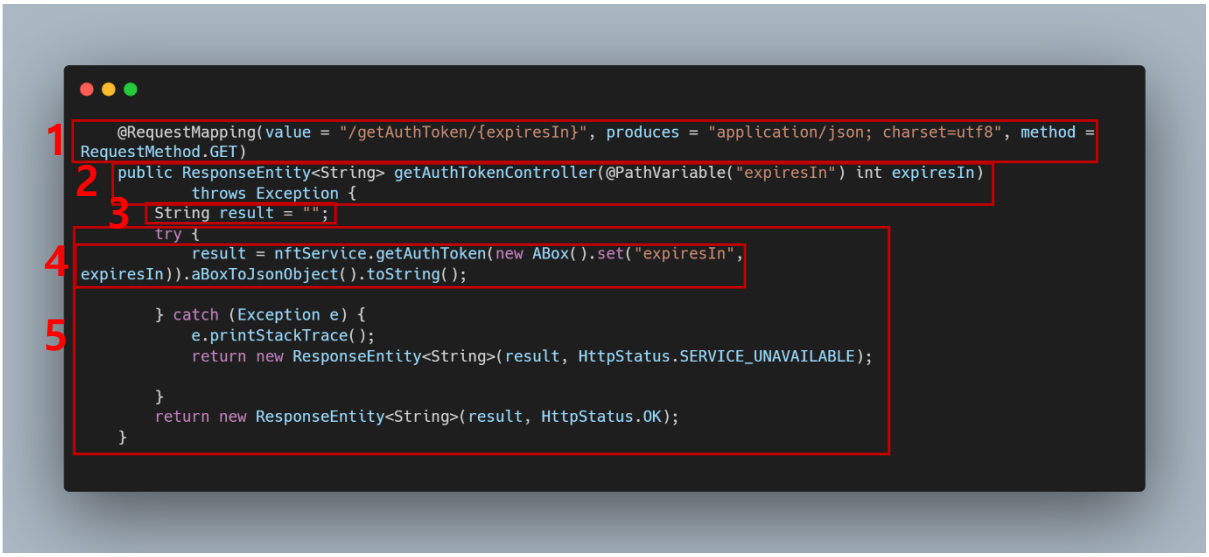
==========예제 코드 가이드 라인==========

POST (HTTP 메시지 body 부분에 담아 전송, JSON 사용시)



```
1   @RequestMapping(value = "/receive-crush", method = RequestMethod.POST, headers = "Content-
Type=application/json;utf-8")
2   public ResponseEntity<String> loungeReciveSubsController(@RequestBody String json) throws Exception {
3       String result = "";
        ABox jsonBox = new ABox();
        jsonBox = jsonBox.jsonToABox(json);
5       try {
4           result = loungeMatchService.loungeReceiveCrush(jsonBox).aBoxToJsonObject().toString();

        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<String>(result, HttpStatus.SERVICE_UNAVAILABLE);
        }
        return new ResponseEntity<String>(result, HttpStatus.OK);
    }
```

GET (URL 주소 끝에 파라미터로 포함되어 전송, 단순 조회 사용시)



```
1   @RequestMapping(value = "/getAuthToken/{expiresIn}", produces = "application/json; charset=utf8", method =
RequestMethod.GET)
2   public ResponseEntity<String> getAuthTokenController(@PathVariable("expiresIn") int expiresIn)
            throws Exception {
3       String result = "";
        try {
4           result = nftService.getAuthToken(new ABox().set("expiresIn",
expiresIn)).aBoxToJsonObject().toString();

        } catch (Exception e) {
5           e.printStackTrace();
            return new ResponseEntity<String>(result, HttpStatus.SERVICE_UNAVAILABLE);

        }
        return new ResponseEntity<String>(result, HttpStatus.OK);
    }
```

1. RequestMapping : value, method, headers로 구성되어 있습니다.
   A. value : 주소 지정, GET 사용시 /{패러미터 이름} 지정
   B. produces : 전달 시 형식 지정 (*수정하지 마세요!)
   C. headers : 받을 헤더파일의 형식 지정 (*수정하지 마세요!)
   D. method : GET, POST 방식 지정
2. 컨트롤러 메소드 지정

A. POST 사용시, @RequestBody String json 지정

B. GET 사용시, @PathVariable(파라미터 이름) 자료형 변수 이름

3. 결과를 전달하기 위한 String result 변수 선언되어 있습니다.

A. POST 사용시, jsonBox.jsonToABox(json) 메소드를 이용해 형 변환 해주세요.

4. Serivce를 이용하여 결과를 처리하는 메소드입니다.

5. 처리한 결과를 전달해주는 부분입니다. Service의 결과와 함께

성공시 HttpStatus.OK, 실패시 HttpStatus.SERVICE_UNAVAILABLE를 전달합니다.

```java
public interface LoungeMatchService {
    public ABox loungeReceiveCrush(ABox jsonBox) throws DataAccessException; // 라운지 호감 받기

}
```

1. 서비스의 인터페이스입니다. 서비스를 구현하기 전에 등록해주세요.

2. 결과 타입 및 파라미터 타입 모두 ABox를 권장합니다.

```java
    @Override
    public ABox loungeReceiveCrush(ABox aBox) throws DataAccessException {
        ABox resultABox = new ABox();
        try {
            String publicId = aBox.getString("publicId");
            aBox.set("publicId", aBox.getString("likeUserId"));
            aBox.set("likeUserId", publicId);
            if (commonDao.select("mybatis.lounge.loungeMatch_mapper.selectLoungeAskStateSQL",
aBox).getString("userAccept").equals("AC01")) {
                aBox.set("userAccept", "AC02");
                aBox.set("pay", 0);
                if (commonDao.update("mybatis.lounge.loungeMatch_mapper.updateLoungeAskAcceptSQL", aBox) != 0) {
                    resultABox.set("check", "ok");
                } else {
                    resultABox.set("check", "fail");
                }
            } else {
                resultABox.set("check", "already");
            }

        } catch (Exception ex) {
            rollBack();
            ex.printStackTrace();
            resultABox.set("check", "fail");
            resultABox.set("check2", ex.toString());
        }
        return resultABox;
    }
```

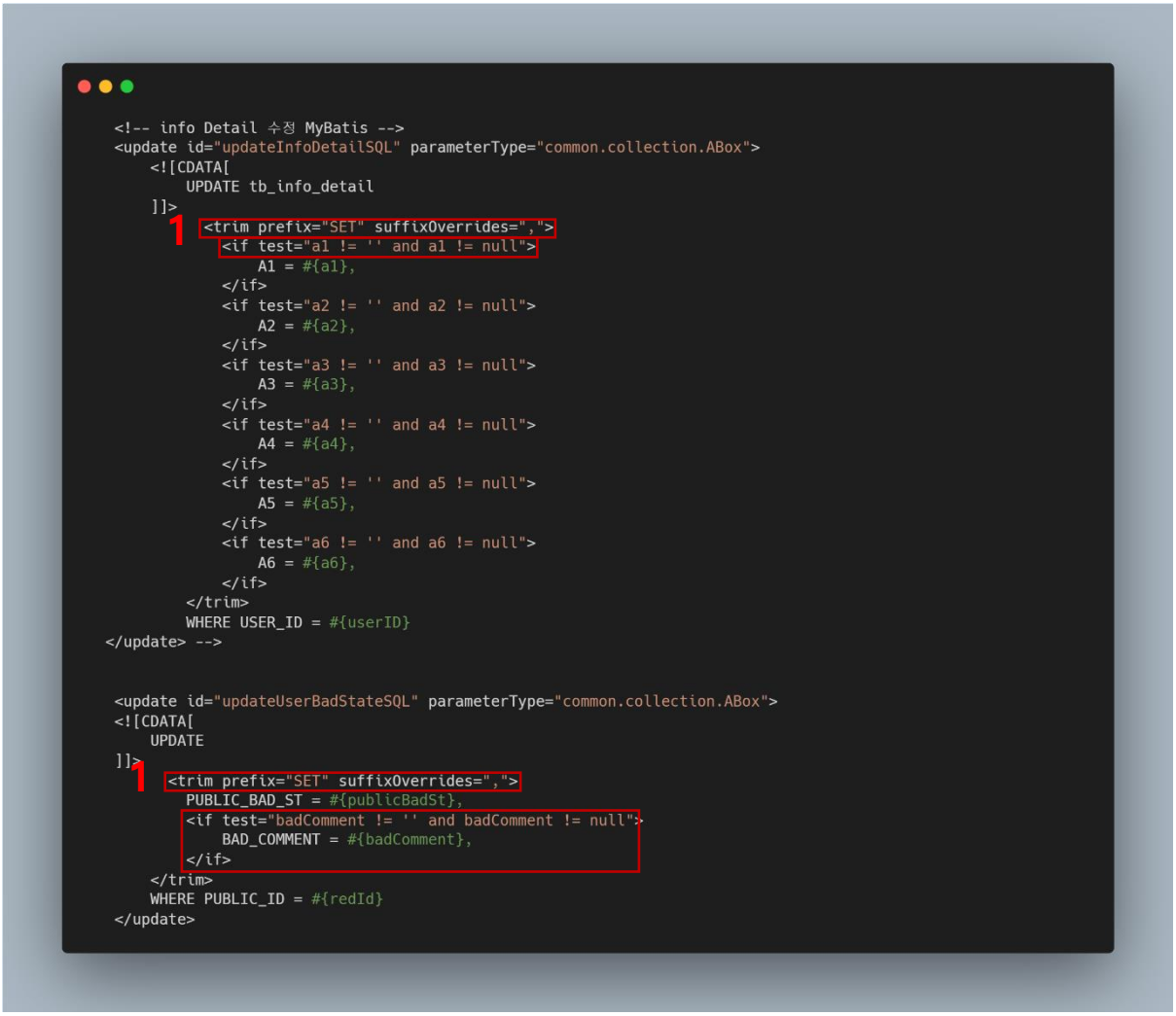
1. 결과를 담기 위한 ABox입니다. 결과 타입 및 파라미터 타입 모두 ABox를 권장합니다.
2. 서비스의 로직을 담당하는 부분입니다.
3. 데이터베이스를 조회하는 메소드입니다. commonDao를 상속받아 사용합니다. 위의 commonDao를 참조해주세요. 사용할 마이바티스 파일의 네임스페이스와 마이바티스 id를 조합해야 합니다.
4. 전달할 결과들을 resultABox에 세팅해주세요.
5. resultABox로 결과를 리턴 해주세요.

==========**마이바티스 가이드 라인**==========

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

1 <mapper namespace="mybatis.shadow.match.match_mapper">

    <!-- 진행 상태 조회하기 MyBatis -->
2   <select id="selectAskStateSQL" parameterType="common.collection.ABox" resultType="common.collection.ABox">
3       <![CDATA[
4           SELECT
                a.USER_ACCEPT as userAccept, a.message, a.USER_ID, a.pay
            FROM
                tb_ask a,
                (SELECT PUBLIC_SEQ from tb_user where PUBLIC_ID = #{publicId}) b,
                (SELECT PUBLIC_SEQ from tb_user where PUBLIC_ID = #{likeUserId}) c
            WHERE
                (a.USER_ID = b.PUBLIC_SEQ AND a.LIKE_USER_ID = c.PUBLIC_SEQ) OR
                (a.USER_ID = c.PUBLIC_SEQ AND a.LIKE_USER_ID = b.PUBLIC_SEQ);

        ]]>
    </select>

    <!-- 호감 수정 MyBatis -->
2   <update id="updateAskAcceptSQL" parameterType="common.collection.ABox">
3       <![CDATA[
4           UPDATE
                tb_ask
            SET
                USER_ACCEPT = #{userAccept},
                pay = #{pay}
            WHERE
                USER_ID = (
                    SELECT
                        a.PUBLIC_SEQ
                    from
                        tb_user a
                    where
                        PUBLIC_ID = #{publicId}
                ) AND
                LIKE_USER_ID = (
                    SELECT
                        b.PUBLIC_SEQ
                    from
                        tb_user b
                    where
                        b.PUBLIC_ID = #{likeUserId}
                );
        ]]>
    </update>
</mapper>
```

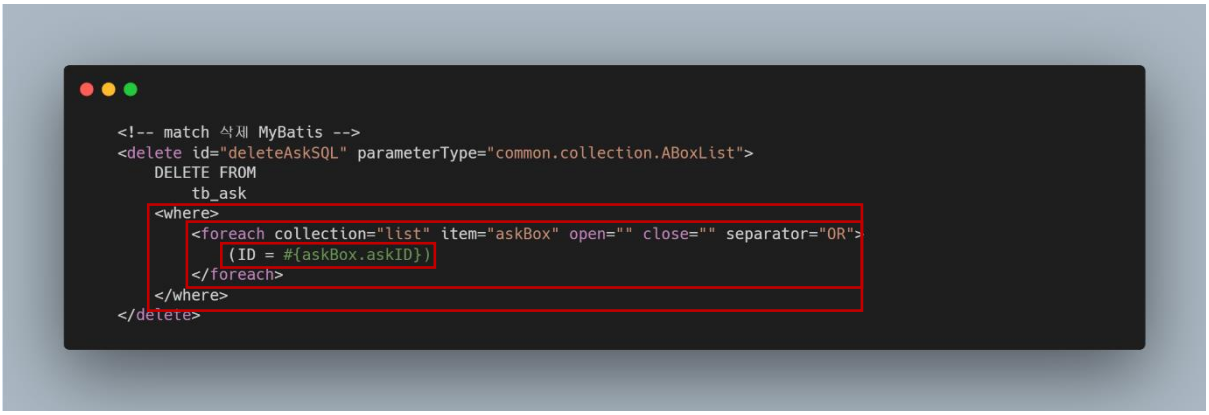
1. 마이바티스의 네임스페이스입니다. 중복된다면 컴파일되지 않습니다.
2. 마이바티스의 id입니다. parameterType과 resultType은 ABox로 고정입니다.
3. ![CDATA]는 해당 영역의 코드를 문자열로 변환합니다. 마이바티스의 기능을 사용하실 때, 범위를 조정해야합니다. 예제는 아래에 있습니다.
4. SQL 본문입니다.

==========마이바티스 가이드 응용 라인==========



```
<!-- info Detail 수정 MyBatis -->
<update id="updateInfoDetailSQL" parameterType="common.collection.ABox">
    <![CDATA[
        UPDATE tb_info_detail
    ]]>
    1 <trim prefix="SET" suffixOverrides=",">
        <if test="a1 != '' and a1 != null">
            A1 = #{a1},
        </if>
        <if test="a2 != '' and a2 != null">
            A2 = #{a2},
        </if>
        <if test="a3 != '' and a3 != null">
            A3 = #{a3},
        </if>
        <if test="a4 != '' and a4 != null">
            A4 = #{a4},
        </if>
        <if test="a5 != '' and a5 != null">
            A5 = #{a5},
        </if>
        <if test="a6 != '' and a6 != null">
            A6 = #{a6},
        </if>
    </trim>
    WHERE USER_ID = #{userID}
</update> -->


<update id="updateUserBadStateSQL" parameterType="common.collection.ABox">
<![CDATA[
    UPDATE
]]>
    1 <trim prefix="SET" suffixOverrides=",">
        PUBLIC_BAD_ST = #{publicBadSt},
        <if test="badComment != '' and badComment != null">
            BAD_COMMENT = #{badComment},
        </if>
    </trim>
    WHERE PUBLIC_ID = #{redId}
</update>
```

1. <trim> 지정으로 재사용 가능한 동적 쿼리를 만드실 수 있습니다.
    A. Prefix : trim 구문을 시작하기 앞에 붙이는 문자열입니다.
    B. suffixOverrides : <if test>의 구문이 사용될 때마다 뒤에 붙이는 문자열입니다.
2. <if test> 또는 <where>을 이용하여 조건문 구현 가능합니다.



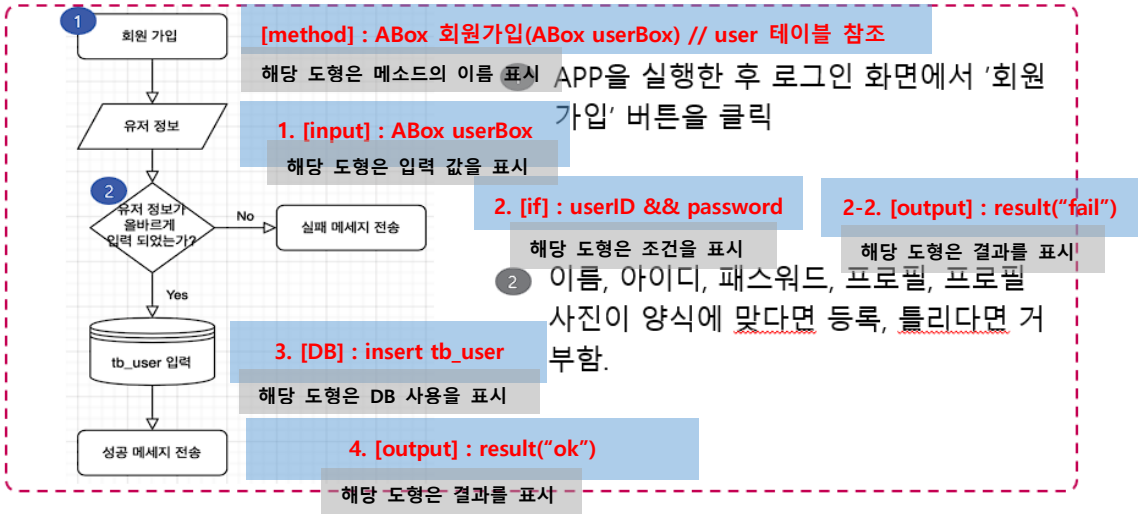
```
<!-- match 삭제 MyBatis -->
<delete id="deleteAskSQL" parameterType="common.collection.ABoxList">
    DELETE FROM
        tb_ask
    <where>
        <foreach collection="list" item="askBox" open="" close="" separator="OR">
            (ID = #{askBox.askID})
        </foreach>
    </where>
</delete>
```

1. <where>을 이용하여 조건문 구현 가능합니다. parameterType을 ABoxList로 지정해주세요.
2. <foreach>를 이용하여 여러 차례 쿼리를 반복할 수 있습니다. ABoxList<String, ABox>를 입력하여 사용하실 수 있습니다.
    A. Collection : 입력 자료형을 지정합니다.
    B. Item : 리스트 안의 객체를 지정합니다.
    C. Open : 반복 시작 시, 앞에 붙이는 문자열을 지정합니다.
    D. Close : 반복 시작 시, 뒤에 붙이는 문자열을 지정합니다.
    E. Separator : 반복 시, 조건문을 사용할 수 있습니다. AND 또는 OR 로 사용하실 수 있습니다.

========== 코딩 가이드 라인 ==========

프로그램 ID	USER-01-01	프로그램 명	유저 회원가입	작성일	2022. 7. 1.	Page	1/8
개요	유저의 정보를 양식에 맞는 지 검증 후 데이터베이스에 등록하는 프로그램					작성자	정용규

<유저 회원가입 연계도>




```
public ABox addUser(ABox userBox) { // [1] userBox 입력,tb_user의 애트리뷰트를 중 일부 또는 전체가 입력됨
    ABox resultBox = new ABox(); // try 문의 위에는 메소드 내 지역 변수 선언
    try { // 로직의 시작
        ABox aBox = userBox;// 메소드 실행 로직 내에서만 사용할 지역 변수 선언, 데이터 가공 처리에 주로 사용, 다만 이 메소드에서 사용하지 않음
        // 테이블의 애트리뷰트 이름은 대문자와 언더바 조합, 자바 코드 내 변수 이름은 카멜 케이스로 작성

        // [2] 조건문 실행, 조건식은 사용자의 주관 또는 프로트엔드의 요청 및 상황에 따라 자의적으로 작성
        if(userBox.getString("userName").length() > 5 && userBox.getString("userPassword").length() > 5
                && userBox.getString("userNickName").length() > 5) {

            // [3] DB 사용, 마이바티스 insert 성공시 1, 실패시 null을 반환, update, delete 성공시 1, 실패시 0을 반환
            if(commonDao.insert("mybatis.common.common_mapper.insertUserSQL", aBox) != 0) {
                // [4] 결과 전달
                resultBox.set("result", "ok");
            } else {
                // [4-2] 결과 전달
                resultBox.set("result", "fail"); // 중복 존재 등의 이유로 실패시 fail 처리, 필요할 경우 "overlap"으로 변경 가능
            }
        // [2-2] 결과 전달
        } else {
            resultBox.set("result", "fail"); // 조건에 맞지 않기 때문에 fail 처리
        }
    } catch (Exception e) {
        resultBox.set("result", "fail"); // 예외 발생이기에 fail, 필요할 경우 별도 처리 및 다른 기술 가능
        e.printStackTrace();
    }
    return resultBox; // 결과 리턴
}
```

1. 클라이언트에서 아이디와 패스트워드를 입력한다.
2. '로그인' 버튼을 클릭할 시, 클라이언트로부터 유저 정보 입력 받는다.
3. 유저의 아이디와 패스워드를 받아온다.
4. 스프링 시큐어리티와 데이터베이스를 이용하여 아이디와 패스워드를 검증한다.
5. 검증에 실패할 경우 클라이언트로 실패 메세지를 전송하고 알고리즘을 멈춘다.
6. 검증에 성공한 경우 해당 유저의 JWT 토큰을 발행한다.
7. 토큰을 서버에 3600초간 임시 저장한다.
8. 클라이언트로 로그인 성공 메세지를 전송한다.

알고리즘 명세서 중 '스프링 시큐어리티', 'JWT 토큰', '토큰', 'Luniverse BlockChain Net' 등의 내용은 **무시하고** 진행해주세요.

기능 흐름도를 **메인**으로, 알고리즘 명세서는 참고하여 코드를 작성해주세요. (기능 흐름도를 우선시 해주세요!)