참! 좋은 은행 IBK기업은행

IBK 클라우드 시스템 구축

업무 개발 가이드

Version 1.2

2024년 7월 1일

IBK 업무 스타트킷 개발 가이드

서버운영팀(클라우드팀)

개 정 이 력

버전	제/개정 일자	제/개정 페이지 및 수정내용	제/개정자
1.0	2022.11.21	개발자 가이드 문서 초안 작성	송호철
1.1	2024.04.22	업무 스타트킷 및 DSP 서비스 고도화 반영	이희교
1.2	2024.07.01	EAI 통신 관련 사항 추가	이희교

- 목 차 -

1. 개요

1.1. 배경

IBK 클라우드 업무 스타트킷은 당행의 클라우드 도입 및 구축에 따라 Public(이하 공공존 클라우드) 및 Private 클라우드(이하 뉴로존 클라우드) 플랫폼에서 개발, 실행, 운영, 관리되는 표준개발환경 가이드이다.

IBK 클라우드 업무 스타트킷은 Spring-Framework를 기반으로 REST API 지향 서비스를 제공하고, Spring Boot의 가벼운 구조로 클라우드 환경에 적용하기 적합한 환경을 제공한다. 향후 IBK 클라우드 시스템 내에서 업무 서비스를 개발하는 표준개발환경으로 적용한다.

1.2. 가이드의 목적과 구성

본 가이드는 IBK 클라우드 플랫폼 환경에서 클라우드 표준개발환경을 활용한 업무 서비스 환경을 개발할 때 참조하도록 작성되었다.

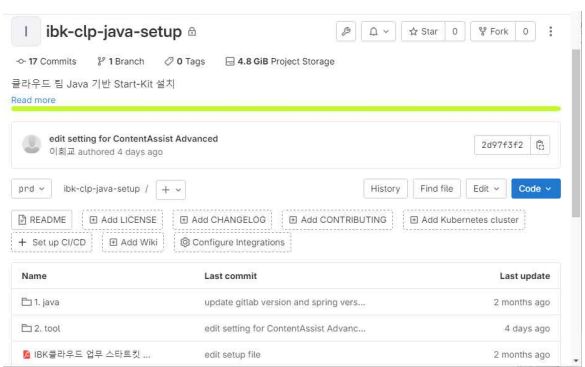
클라우드 웹 애플리케이션을 개발하기 위해서는 DevOps, CI(Continuous Integration) / CD(Continuous Delivery), Container 등을 함께 고려해야 하지만 본 가이드는 우선 업무 서비스를 제공하기 위한 개발에 필요한 표준 가이드 제공과 서비스 개발 관련 기술 사항 및 업무 개발 관련 샘플 코드를 제공을 통하여 개발 편의를 제공하고자 한다.

향후 IBK 클라우드 구축의 실현 및 변화방향에 맞추어 프레임워크에서 적용되어야 하는 MSA(Micro Service Architecture), Service Mesh 등의 당행 클라우드 표준 기술에 대하여 향후 지속적으로 업그레이드 해나갈 예정이다.

1.3. 업무 개발 환경

업무 스타트킷은 STS 환경에서 개발할 수 있게 설치 환경을 제공한다.

(설치 경로 : http://gitlab.ibk.co.kr/icpkit/spring/ibk-clp-java-setup)

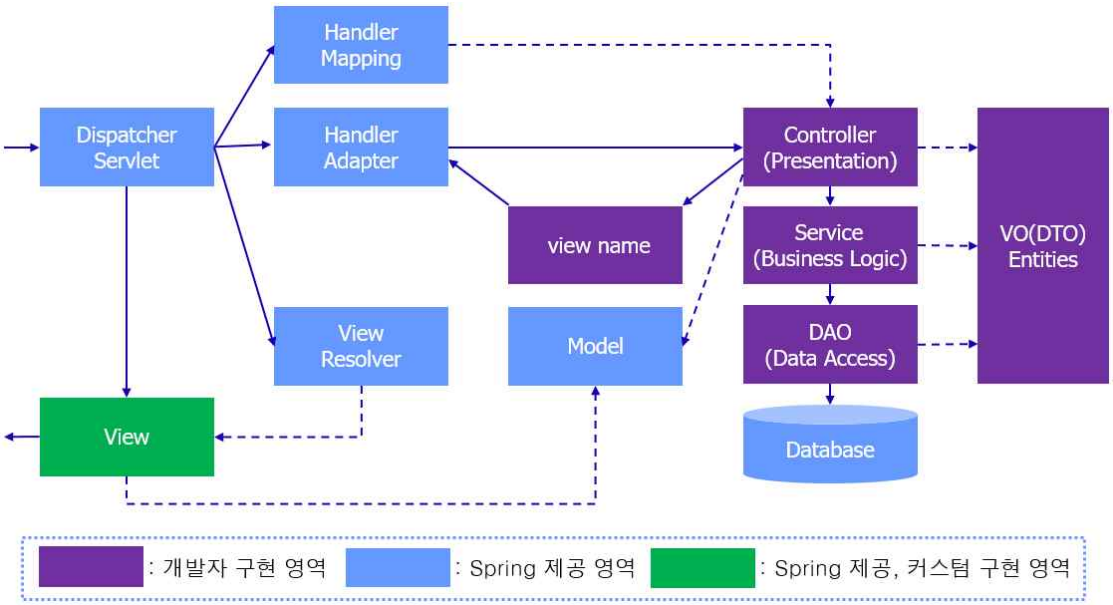


<업무 개발 도구>

2. 웹 애플리케이션 아카텍처

2.1. 애플리케이션 객체 계층 구조

IBK 클라우드 업무 스타트킷은 스프링 기반으로 웹 애플리케이션 계층을 따른다.

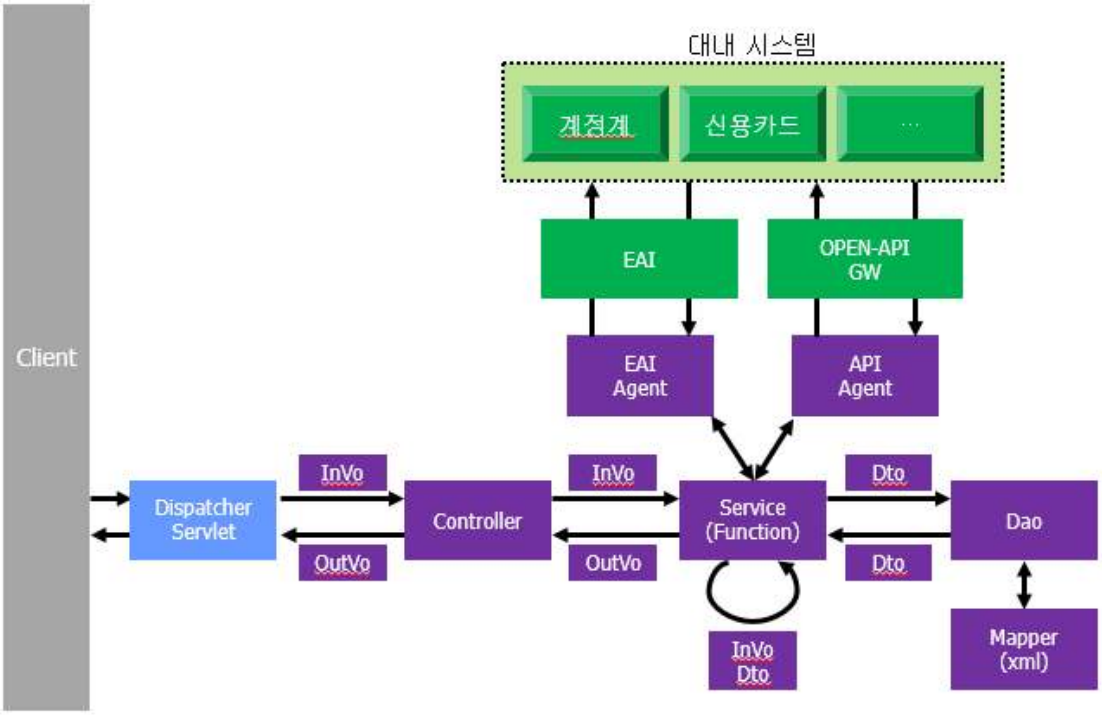


IBK 클라우드 프레임워크의 각 계층에서 사용되는 핵심 클래스/객체는 다음과 같다.

계층(Layer)	클래스/객체	주요기능
Presentation	Controller	• 웹 클라이언트의 요청 응답을 처리 • Service 및 Data Access Layer에서 발생하는 Exception을 처리
Business Logic	Service	• 애플리케이션 비즈니스 로직 처리 및 도메인 모델의 적합성 검증 • 당행 대내 인터페이스(EAI) 처리 • 트랜잭션 관리
Data Access	Dao	• Mybatis를 활용한 Database 접근 관리 • Database 관련 CRUD 처리
Domain Model	Vo	• Presentation 계층의 Request 및 Response • Service 처리를 위한 객체 • 당행 대내 인터페이스(EAI) 처리를 위한 객체 • 로직이 없고 순수하게 getter, setter만 가짐
	Dto	• Database 데이터 처리를 위한 객체 • 공통 생성 정보(생성자, 생성일자 정보 등) 관련 별도 관리

2.2. 거래 흐름 및 호출 관계

프레임워크 내에서 각 객체간 거래의 개략적인 흐름은 아래와 같다.



① 사용자의 단말(브라우저 또는 모바일 웹/앱) 에서 HTTP 프로토콜로 Request 송신

② WAS에서 Request가 수신되면 요청 URI와 일치하는 등록된 Controller가 작업을 수행하며, 이 때 JSON 형식의 Request Body는 정의된 InVo로 매핑

③ Controller는 인증 및 입력 값에 대한 유효성 검증을 수행하며, 업무와 연관된 서비스 처리를 위해 Service를 호출

④ Service는 모듈화된 단위 작업을 수행하고 필요한 경우 Dao 또는 대내 연계 인터페이스를 호출, Service의 공통 요소는 Function(공통 서비스) 및 Util(공통 함수)로 관리

⑤ Dao는 데이터베이스의 CRUD 작업을 수행하며 이를 위하여 필요한 SQL은 Mapper에 정의된 정보를 읽어와 질의를 수행

⑥ Dao 또는 연계 인터페이스 처리 결과는 Service로 전달되고, 하나의 트랜잭션 단위의 업무를 처리완료 후 OutVo 형태로 결과값 리턴

⑦ Service 트랜잭션 단위로 수행한 결과는 Controller로 전달되고, Controller에서는 정상 작업을 위한 추가적인 Service 호출(단일 Service 호출의 경우 생략) 후 결과값 리턴

구분		피호출		
		Controller	**Service**	**Dao**
호출	**Controller**	X	○	X
	Service	X	○	○
	Dao	X	X	X

2.3. 명명규칙

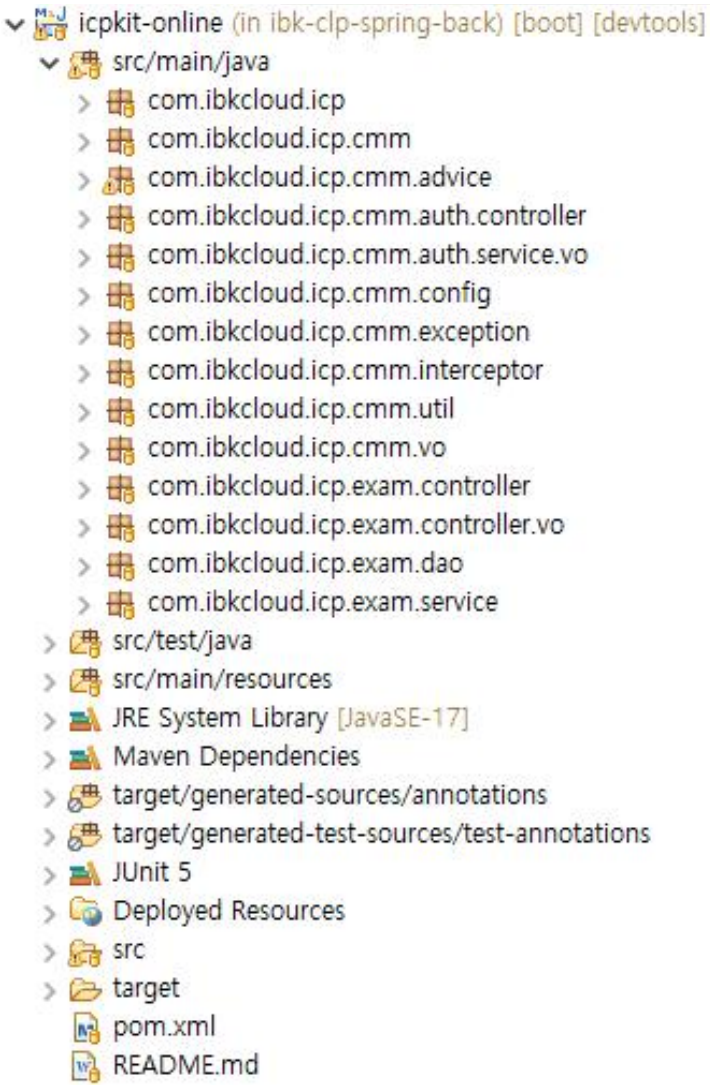
2.3.1. 프로젝트

프로젝트는 EAMS의 업무구분명칭을 기준으로 생성하고, 하나의 서비스에 여러 단위업무시스템 생성 시 서비스의 성격에 맞게 프로젝트 명칭을 구분하여 생성한다.
당행에서 클라우드에 대한 소개와 개선 사항 등을 관리하고 있는 클라우드 포탈 서비스를 기준으로 생성한 프로젝트 명칭의 예시는 다음과 같다.

시스템코드	서비스구분	단위업무시스템	설명
icp	main		클라우드 포탈 업무 서비스 관리
icp	front	user	클라우드 포탈 사용자 화면 관리
icp	front	admin	클라우드 포탈 관리자 화면 관리
icp	batch		클라우드 포탈 배치 서비스 관리

예시) 클라우드포털 업무스타트킷 프로젝트 명 : icp-main

프레임워크 설치가이드에 따라 프로젝트가 생성되면 다음과 같은 구조를 확인할 수 있다.



2.3.2. 패키지 및 디렉토리

IBK 클라우드의 기본 패키지는 com.ibkcloud 이다.

단위업무시스템의 시스템코드가 프로젝트 생성 기본 패키지의 명칭이 된다. 이후 단위업무시스템 내에서 업무기능 정의에 따라 1-2 단계로 하위 업무패키지를 구성한다. 하위 업무패키지는 3-4 자리의 영문 소문자로 정의(예시 – com.ibkcloud.icp.clp.auth)한다.

최종 업무패키지 하위에는 다음과 같은 Java Package 형태로 정의되어야 한다.

프로젝트	기본 패키지	업무 패키지	계층 패키지	설명
icp-main	src/main/java	cmm	advice	AOP 관련 업무공통 클래스
			config	업무 설정 관련 Bean 클래스
			interceptor	인증과 같은 공통 요청/응답 처리 관리
			util	업무 공통 Util 클래스
		패키지명	**controller**	업무 관련 Controller 클래스
			controller.vo	Controller 관련 InVo/OutVo 클래스
			service	업무 관련 Service 클래스
			service.vo	통신에 필요한 Vo 클래스
			dao	DB 처리 관련 Mapper 인터페이스
			dao.dto	Dao 관련 Dto 도메인 클래스
	src/main/resource			로컬, 개발, 운영 관련 설정 파일
		mapper	패키지명	DB 처리 관련 Mapper 파일(xml)
		message		메시지 설정 파일(properties)

2.3.3. 클래스

클래스(인터페이스) 파일은 해당 클래스가 수행하는 논리명칭으로 우선 **[기능명]+[계층명]** 형식으로 구성한다. 기능명은 당행의 DA 포털에 등록된 단어를 사용하는 것을 원칙으로 하며 '_'를 제외한 특수문자를 허용하지 않는다. 이렇게 정의한 논리명칭으로 단어에 해당하는 DA 포털에 등록된 영문약어를 Upper Camel Case 규칙으로 생성된 영문명이 실제 클래스명이 된다.

기본 Dao에는 전체 컬럼에 대한 기본적인 select, insert, update, delete가 포함되도록 한다. 예를 들어 '공지사항(게시판)관리' 기능을 담당하는 계층별 클래스명은 다음과 같다.

계층(Layer)	논리명	클래스명
Presentation	예제게시판Controller	ExamBlbrController
	예제게시판목록조회Vo	ExamBlbrListInqInVo, ExamBlbrOutVo
Business Logic	예제게시판Service	ExamBlbrService
Data Access	예제공지사항조회Dao	ExamPbnsInqDao
	예제공지사항조회Dto	ExamPbnsInqDto

2.3.4. 메서드

메서드는 기능을 담당하는 논리명칭을 DA포털에 등록된 영문약어를 Lower Camel Case 규칙에 맞추어 **[Action명]+[기능명]** 형태로 작성한다.
Action명은 동명사형으로 지정하며 계층단위로 영문단어를 다르게 적용한다. Action명은 다음에 정의된 명칭을 사용한다.

구분	Action명	Action 영문명	비고
Dao	등록	insert	정보 저장
	변경	update	정보 갱신
	삭제	delete	정보 삭제
	조회	select	단건 정보 조회
	lock 조회	selectForUpdate	정보 수정을 위한 조회(Row lock)
	목록조회	selectList	대량 정보 조회
	페이지조회	selectPage	페이지 단위로 정보 조회
Controller /Service	조회	inq	단건/대량 정보 조회
	등록	rgsn	정보 신규 생성
	변경	mdfc	신규 생성 로직이 미포함된 정보 갱신
	저장	strg	신규 생성 로직이 포함된 정보 갱신
	삭제	del	정보 삭제
	확인	cnfa	정보 확인
	검증	vrfc	정보 검증
	처리	pcsn	정보 처리
	채번	nmbn	테이블의 Key 번호 채번(Sequence)
	송신	trnm	타 시스템으로 정보 송신
	수신	rcv	타 시스템에서 정보 수신
	[추가예정]		

예를 들어 논리명칭이 '공지사항목록조회' 인 경우 Controller와 Service에서 메서드명칭은 'inqPbnsCtlg' 로, Dao에서는 'selectListPbns' 로 작성한다.

2.3.5. Resource(Mapper)

Mybatis를 사용하기 위한 Mapper XML은 Dao 와 1 대 1로 같이 생성하고, XML 파일의 명칭은 Dao 인터페이스 명칭에서 Dao를 Mapper로 대체한 명칭을 사용한다.
예를 들어 대응되는 Dao가 'PbnsDao' 인 경우 Mapper 파일은 'PbnsMapper.xml' 로 생성한다.

구분	논리명	xml명
Mapper	공지사항Mapper	PbnsJob.xml

3. 개발 절차

3.1. Vo/Dto 작성

Domain Model 객체로 순수하게 데이터를 담아 계층 간으로 전달하는 객체이다. 로직을 가지지 않으며 getter/setter 만을 메서드로 갖는다.

3.1.1. lombok

롬복(lombok)은 자바 클래스에 Getter, Setter, 생성자 등을 자동으로 만들어 주는 도구이다. 개발자가 별도의 멤버 변수에 대하여 개별적으로 Getter, Setter, Contructor 메서드를 생성하지 않아도 어노테이션으로 관련 메서드가 생성된다.

3.1.2. 어노테이션

@Getter, @Setter, @NoArgsConstructor, @LocalName 만을 사용하고, @Data 는 사용하지 않는다.

선언 위치	어노테이션	속성	설명
클래스	@Getter		Getter 메서드 자동 생성
	@Setter		Setter 메서드 자동 생성
	@NoArgsConstructor		기본 생성자 자동 생성
	@LocalName		클래스에 대한 한글명
변수	@DomainMeta	localName	해당 변수 한글명
		length	해당 변수 최대 자리수(byte 단위)
		scale	해당 변수 소수점 자리수(BigDecimal 등)
		notNull	해당 변수 NULL 허용 여부

3.1.3. 데이터타입

데이터베이스(MySQL, PostgreSQL)의 데이터 타입과 Vo/Dto에서 사용되는 java 의 타입을 매핑한다. 데이터의 자리수를 지정하기 어려운 float, double 타입은 사용하지 않는다.

DB 타입	Java 타입	비고
CHAR(ACTER)	String	
VARCHAR		
MEDIUMTEXT		최대길이 16,777,215
TEXT		최대길이 65,535
LONGTEXT		최대길이 5,592,405
BIGINT	java.math.BigInteger	
BIGINT UNSIGNED		
INT	long	Unsigned 까지 포함
INTEGER		
DECIMAL	java.math.BigDecimal	소수점 포함
NUMERIC		
DATE	java.time.LocalDate	
DATETIME	java.time.LocalDateTime	
TIMESTAMP		
BLOB	byte[]	

3.1.4. Dto 부모 클래스

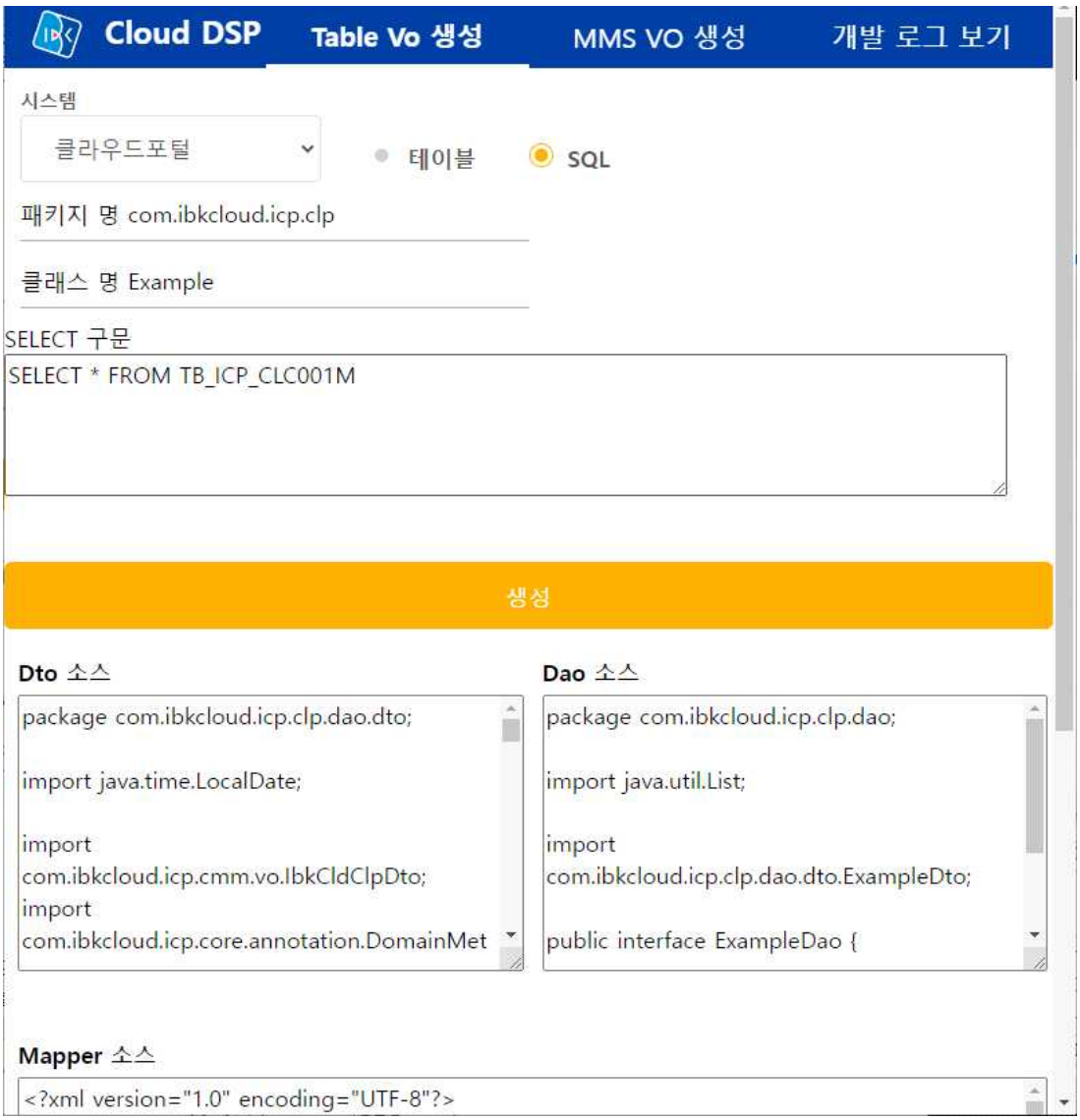
Dto는 Database의 공통속성을 제어하기 위하여 부모클래스를 상속받도록 한다. 클라우드 포털 시스템은 각 테이블에서 데이터의 변경 추적을 위하여 공통속성으로 SYS_LSMD_ID (시스템최종변경ID), SYS_LSMD_TS(시스템최종변경일시)를 가지고 있다. 이를 공통적으로 insert, update 시에 값을 바인딩하기 위하여 IbkCldClpDto 클래스를 extends 한다.

3.1.5. Vo/Dto 소스 생성 지원 플랫폼

IBK 클라우드 서비스 플랫폼 중 개발자지원플랫폼(이하 DSP 서비스)을 제공하고 있으며, 위한 제공하는 서비스 항목은 Table 정보를 통한 코드 생성, SQL 구문(SELECT 절)을 통한 코드 생성, MMS 전문 정보를 통한 코드 생성 서비스로 구성되어있다.
제공 중인 DSP 서비스는 아래와 같다.
(DSP 페이지 링크 – http://dcloudguide.ibk.co.kr:28080/)

(1) 테이블 정보를 통한 소스코드 자동 생성

(2) SQL 구문(SELECT 구문)을 통한 소스코드 자동 생성



(3) MMS 전문 등록을 통한 소스코드 자동 생성

3.2. Dao 작성

3.2.1. Mapper 작성

MyBatis를 사용할 때 SQL을 작성하는 XML 파일이다. 본 가이드는 주요 사용법만을 기술하고, 이외의 자세한 사용법은 MyBatis 매뉴얼을 참고하도록 한다. Mapper 파일은 "src/main/resource/mapper/업무패키지명/" 디렉토리에 작성한다.

3.2.1.1. Mapper 파일 구조

Mapper 파일은 다음과 같은 DTD 구조로 구성된다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- mapper DTD 선언 -->
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://internal.nexus.ibk.co.kr/repository/maven-client-hosted/dtd/mybatis-3-mapper.dtd">

<!-- SQL Mapping -->
<mapper namespace="Dao인터페이스명">

        <insert id="insertData" parameterType="Dto클래스명">
        <update id="updateData" parameterType="Dto클래스명">
        <delete id="updateData" parameterType="Dto클래스명">
        <select id="selectData" parameterType="string" resultType="Dto클래스명">
        <![CDATA[
        /* Dao인터페이스명.selectData */
        ]]>
        </select>

</mapper>
```

3.2.1.2. Mapper 작성 유의사항

mapper namespace 의 명칭은 패키지 명을 포함한 전체 Dao클래스 경로를 작성한다. IBK 클라우드 업무 스타트킷은 Dao를 인터페이스로 작성하고, 이를 서비스 컴파일 시 @MapperScan을 통해 Bean을 생성하고 있으며 Dao 내부 메서드를 구현하지 않는다. 마찬가지로 Mapper 태그에서 사용하는 parameterType, resultType의 경우도 Dto를 사용하는 경우 전체 클래스 경로로 작성한다.

parameterType, resultType은 primitive type(long, String) 혹은 Dto 클래스로 정의된 reference type만으로 사용하고, Map 객체는 사용을 제한한다.

Mapper 내부 SQL구문 작성 시 가장 앞부분에 주석으로 SQL ID를 작성하여 장애추적을 용이하게 작성한다. SQL ID는 **"<namespace>.<sql id값>"** 형식으로 작성한다.

SQL은 Dynamic SQL을 사용하지 않는 것을 원칙으로 한다.

파라미터는 보안상의 이유로 #{} 구문을 사용하며 ${} 구문은 사용을 금지한다.

마찬가지로 <include>, <choose> 등의 태그도 사용하지 않는 것이 원칙이며, 예외적으로 select 구문에서 where 조건문 뒤에 <if> 는 사용을 허용한다.

3.2.2. Dao 인터페이스 작성

Dao는 MyBatis Mapper XML에 기재된 SQL을 호출하기 위한 인터페이스이다. Dao 인터페이스는 다음과 같은 구문이다.

```
@Mapper
public interface 인터페이스명 {
}
```

인터페이스명은 Mapper의 XML파일에서 namespace과 동일해야 한다.

3.2.3. Dao 메서드 작성

Dao는 인터페이스로 구현되므로 **메서드의 명칭과 Mapper XML의 SQL ID와 일치**하도록 작성(불일치할 경우 서비스 컴파일 시 에러가 발생)한다.

3.3. Service 작성

Service는 업무처리 로직을 수행하며, 단위 기능의 모듈 단위로 작성한다. Service 는 인터페이스와 인터페이스를 구현한 클래스를 별도로 만들지 않고, 클래스만을 생성하는 것을 원칙으로 한다.

3.3.1. Service 클래스 주석

클래스 주석은 클래스 선언부 상단에 다음의 형식으로 작성한다.

```
@파일명 : UserMngmService
@논리명 : 사용자관리 서비스
@작성자 : 홍길동(사번)
@설명   : 사용자의 기본정보를 등록, 수정, 조회하고 권한을 제어하는 Service 클래스
-----------------------------------------------------------------------
수정일자 : 수정자 : 요청자(SR 번호) : 수정된 메소드 : 수정내용
-----------------------------------------------------------------------
작성일자 : 2021.05.08 : 작성자 : 홍길동(사번),  작성내용 : 최초작성
```

3.3.2. 클래스 작성

서비스 클래스는 다음과 같은 어노테이션을 클래스 선언부 상단에 작성한다.

- @RequiredArgsConstructor : final 이나 @NotNull과 연동 (@Autowired는 사용 X)
- @Service : 프레임워크가 서비스 클래스로 인식

서비스의 Bean 객체 관련 변수는 반드시 private final 로 선언해야 한다.

3.3.3. 메서드 주석

메서드 주석은 다음과 같이 작성한다.

```
@메서드명 : inqCust
@논리명 : 고객조회
@설명 :

@param : custInfoVo
@return : custInfoVo
```

3.3.4. 메서드 작성

메서드는 단위 기능을 최대한 모듈화하여 구현한 메서드와 서비스 메서드를 조립하여 통합 기능을 구현하는 메서드를 작성한다. 메서드에서 Dao를 이용하여 데이터를 처리하는 경우 트랜잭션 관리 단위가 된다.

트랜잭션은 '@Transactional' 어노테이션을 클래스 혹은 메서드 선언부에 작성한다.

- @Transactional(propagation=Propagation.REQUIRED, rollbackFor=Exception.class) : 부모트랜잭션내에서 실행(부모트랜잭션이 없는 경우 새로운 트랜잭션을 생성)
- @Transactional(propagation=Propagation.REQUIRES_NEW, rollbackFor=Exception.class) : 부모트랜잭션과 별도의 트랜잭션에서 실행

3.4. Controller 작성

Controller는 일반적으로 사용자 요청을 처리한 후 지정된 View로 모델 객체를 전달하는 역할을 수행하나 IBK 클라우드 프레임워크는 RESTFul 웹 서비스 구축을 위한 REST(REpresentative State Transfer) API 전용 Controller 로 구현한다.

Controller는 요청에 대한 처리를 위한 서비스 호출, 응답결과 조립, 인증 처리, 오류 처리 만을 담당하도록 하고, 각종 검증 및 업무처리는 Service에서 처리되도록 한다.

3.4.1. Controller 클래스 주석

Service 클래스의 주석("3.3.1. Service 클래스 주석" 참고)과 동일한 형태로 작성한다.

3.4.2. 클래스 작성

Controller 클래스 상단에 다음 어노테이션 구문을 선언한다.

- @RestController : REST 요청 값에 대한 동적 정보를 생성하는 Controller 클래스
- @RequiredArgsConstructor : final 이나 @NotNull과 연동 (@Autowired는 사용 X)
- @RequestMapping("url경로") : 요청에 맞는 클래스 매핑을 위한 URL 경로

Controller의 Bean 객체 관련 변수는 반드시 private final 로 선언해야 한다.

3.4.3. 메서드 주석

Service 메서드의 주석("3.3.3. 메서드 주석" 참고)과 동일한 형태로 작성한다.

3.4.4. 메서드 작성

Controller의 메서드는 REST 웹 서비스를 위한 API 형태로 구현한다. Request 방식에 따라 메서드 선언부 상단에 다음 어노테이션을 사용한다.

- @GetMapping("url경로") : HTTP 프로토콜의 GET(조회) 방식의 URL 경로
- @PostMapping("url경로") : HTTP 프로토콜의 POST(입력) 방식의 URL 경로

POST 방식으로 요청된 json 형태의 입력값을 InVo로 매핑하기 위하여 메서드의 parameter는 다음과 같이 작성한다.

```
public UserOutVo insertUser(@RequestBody UserInVo inVo) {
}
```

- UserOutVo : 응답 관련 Vo 객체를 JSON 형태(기본 설정)로 변환하여 리턴
- @RequestBody : 요청 관련 본문 전체 JSON 정보를 객체로 변환하기 위한 선언
- UserInVo : JSON 타입(key)의 요청 정보를 Vo 객체 타입(variable) 변환하여 수신

3.5. Exception 처리

3.5.1. 오류 메시지 정의

에러코드 유형은 메시지 코드와 동일하게 정의되며, 파라미터가 필요한 경우에는 {0}, {1} ...로 정의한다.

- 메시지(에러) 코드 : [업무코드 2자리][유형구분 1자리][숫자 5자리]
- 유형구분 : E(에러), W(경고), I(정보)로 정의해서 업무코드를 사용한다.
- 에러코드 파일 위치 : src/main/resources/message/messages.properties

<messages.properties>

```
# error code - message
COE00000=오류가 발생했습니다. 잠시 후 다시 시도해주세요.
COE00001=오류가 발생했습니다. {0}

# login
LOE00001=오류가 발생했습니다. 잠시 후 다시 시도해주세요.
LOE00002=ID가 입력되지 않았습니다.
LOE00003=비밀번호가 입력되지 않았습니다. {0} {1}
```

3.5.2. 오류처리방법

오류처리는 공통에서 처리 되므로 일반적으로 개발자는 오류처리를 별도로 하지 않는다. 개발자가 try-catch를 사용해서 에러를 직접 처리하는 경우 시스템으로 에러가 전달되지 않으며 에러로 판단되지 않는다. catch 구문에서 포괄적으로 Exception을 사용하는 것을 금지하며 IOException, BizException 등으로 상세한 Exception 객체를 사용한다. 에러를 직접 처리하는 경우 catch구문 내부에서 throw를 통해 다시 상위 클래스로 에러 정보를 전달하도록 해야한다. 에러가 발생해도 로직에 문제가 없을 경우 주석(// ignore 주석)을 사

용해 소스를 관리한다. 유효 값 검증 등의 사항으로 서비스 중 직접 에러를 발생시키고자 하는 경우 Exception을 사용하지 않고 BizException을 사용해서 처리한다.

```
try {
	...
}
catch(BizException e) {
	// TODO
	throw e;
}

try {
	...
}
catch(BizException e) {
	// ignore
}
```

3.5.3. BizException

소스코드에서 BizException을 실행한다. 코드와 메시지를 화면에 전달해서 사용자에게 정의한 메시지를 표출한다. 정의되어 있지 않은 에러코드를 사용하는 경우 기본 오류 내용이 전달된다.

```
// 파라미터가 없는 에러(코드만 입력)
if(error) {
	throw new BizException("COE00000");
}


// 파라미터가 포함된 에러(파라미터 데이터가 {0}, {1}에 각각 매핑)
if(error) {
	throw new BizException("COE00001", new Object[] { "파라미터1", "파라미터2" });
}

// try-catch 에서 에러는 반드시 throw 한다.
try {
	// ...
}
catch(BizException e) {
	log.error(e.getMessage(), e);
	throw e;
}
```

공통 에러 로직을 처리하는 로직은 다음과 같다.

```
if(e instanceof BizException) {
	return ..;
}

if(e instanceof UnAuthorizedException) {
	throw ...;
}

return ...;
```

3.6. Log 처리

3.6.1. Log 레벨

로그는 로그의 대상, 실행 상황에 따라 출력을 조정할 수 있다. 본 프로젝트에서는 총 6가지의 로그레벨을 사용하며, 각 레벨에 대한 사용 용도는 다음과 같다.

레벨	수준	용도	내용	비고
FATAL	6단계	치명적인 오류 발생 시	DB접속 실패 등 시스템 운영에 치명적인 상황을 기록해야 하는 경우	
ERROR	5단계	일반적인 오류 발생 시	비즈니스 로직 오류 등 서비스 실행이 실패한 상황을 기록해야 하는 경우	
WARN	4단계	경고성 메시지	오류는 아니나 관리자에게 경고를 표시해야 할 필요가 있는 경우	
INFO	3단계	안내 메시지	서비스의 시작과 종료와 같은 일반적인 안내 메시지를 출력하는 경우 등	- 운영서버
DEBUG	2단계	디버깅 메시지	변수 설정 값이나 서비스 메소드내에서 실행 단계를 파악하고자 하는 경우 등	- 개발서버
TRACE	1단계	추적 메시지	DEBUG 보다 자세한 실행 내용을 파악하고자 하는 경우	

로그레벨은 다음과 같은 포함관계를 갖는다. 즉, 저수준의 지정된 로그레벨보다 고수준의 레벨로 지정된 로그는 출력 대상이다.

(TRACE > DEBUG > INFO > WARN > ERROR > FATAL)

3.6.2. 로그 사용

본 프로젝트 개발에는 System을 사용한 출력과, ERROR, INFO, DEBUG 레벨 이외의 로그에 대한 사용은 지양한다. INFO 레벨은 개인정보를 포함하지 않으며, 데이터를 볼 필요가 있다고 판단되는 경우 유한적으로 사용한다.

```
public void method() throws Exception {
    log.info("info message");
    try {
    } catch(BizException e) {
        log.error(e.getMessage()); // 에러 로그 사용 시
    }
}
```

3.6.3. 디버그 로그

코드 실행의 최적화를 위해 isDebugEnable()를 사용하여 디버그 로그를 감싼다.

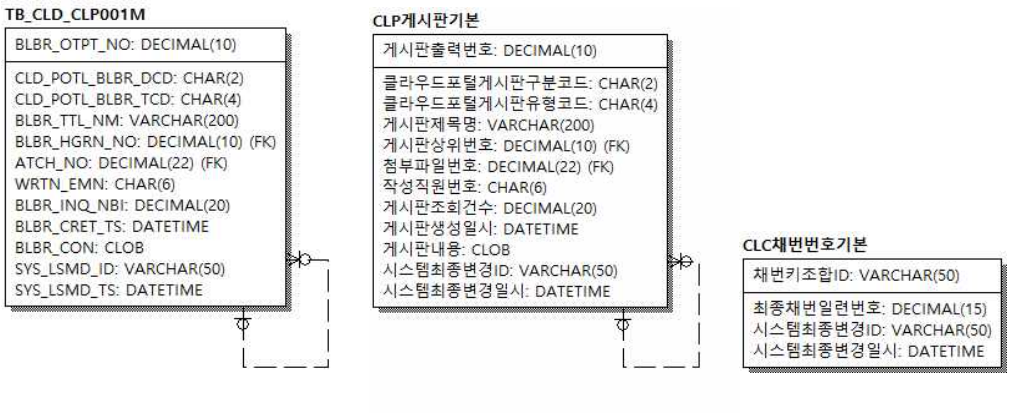
(미사용시, 운영 환경에도 파라미터 조립 소요시간이 발생하여 성능 저하의 원인으로 작용)

```
public void method() throws Exception {
    if(log.isDebugEnabled()) {
        log.debug("debug message");
    }
}
```

4. 개발 예제

4.1. 예제 소스

다음과 같은 게시판 테이블의 데이터를 CRUD 하는 예제이다.



4.1.1. Vo/Dto

<ExamBlbrPageInqInVo.java>

```
@Getter @Setter @NoArgsConstructor @LocalName("예제게시판목록조회InVo")
public class ExamBlbrPageInqInVo {
    @DomainMeta(localName="페이지당출력건수", length=10) @NotNull
    private long pageSize;
    @DomainMeta(localName="페이지번호", length=10) @NotNull
    private long pageNo;
    @DomainMeta(localName="페이지그룹조회여부", length=1) @NotNull
    private String pageGrpInqYn;
    @DomainMeta(localName="페이지그룹수", length=10) @NotNull
    private long pageGrpNbi;
    @DomainMeta(localName="클라우드포털게시판구분코드", length=2) @NotNull
    private String cldPotlBlbrDcd;
    @DomainMeta(localName="클라우드포털게시판유형코드", length=4)
    private String cldPotlBlbrTcd;
}
```

<ExamBlbrOutVo.java>

```
@Getter @Setter @NoArgsConstructor @LocalName("예제게시판OutVo")
public class ExamBlbrOutVo {
    @DomainMeta(localName="게시판출력번호", length=10) @NotNull
    private BigDecimal blbrOtptNo;
    @DomainMeta(localName="게시판상위번호", length=10)
    private BigDecimal blbrHgrnNo;
    @DomainMeta(localName="클라우드포털게시판구분코드", length=2) @NotNull
    private String cldPotlBlbrDcd;
    @DomainMeta(localName="클라우드포털게시판유형코드", length=4)
    private String cldPotlBlbrTcd;
    @DomainMeta(localName="게시판제목명", length=200)
    private String blbrTtlNm;
    @DomainMeta(localName="첨부파일번호", length=22)
    private BigDecimal atchNo;
    @DomainMeta(localName="작성직원번호", length=6)
    private String wrtnEmn;
    @DomainMeta(localName="게시판조회건수", length=20)
    private BigDecimal blbrInqNbi;
    @DomainMeta(localName="게시판생성일시", length=14)
    private LocalDateTime blbrCretTs;
    @DomainMeta(localName="게시판내용", length=4000)
    private String blbrCon;
}
```

<TbCldClp001mDto.java>

```
@Getter @Setter @NoArgsConstructor @EqualsAndHashCode(callSuper=false)
@LocalName("CLD게시판기본Dto")
public class TbCldClp001mDto extends IbkCldClpDto {
    @DomainMeta(localName="게시판출력번호", length=10) @NotNull
    private BigDecimal blbrOtptNo;
    @DomainMeta(localName="게시판상위번호", length=10)
    private BigDecimal blbrHgrnNo;
    @DomainMeta(localName="클라우드포털게시판구분코드", length=2) @NotNull
    private String cldPotlBlbrDcd;
    @DomainMeta(localName="클라우드포털게시판유형코드")
    private String cldPotlBlbrTcd;
    @DomainMeta(localName="게시판제목명", length=200)
    private String blbrTtlNm;
    @DomainMeta(localName="첨부파일번호", length=22)
    private BigDecimal atchNo;
    @DomainMeta(localName="작성직원번호", length=6)
    private String wrtnEmn;
    @DomainMeta(localName="게시판조회건수", length=20)
    private BigDecimal blbrInqNbi;
    @DomainMeta(localName="게시판생성일시", length=14)
    private LocalDateTime blbrCretTs;
    @DomainMeta(localName="게시판내용", length=4000)
    private String blbrCon;
}
```

4.1.2. Dao/Mapper

<TbCldClp001mDao.java>

```
@Mapper
public interface ExampleBlbrDao {
   public List<TbCldClp001mDto> selectPageExampleBlbr(PageRequest<ExampleBlbrDto> dto);
// ...
}
```

<ExampleBlbrMapper.xml>

```
<select id="selectPageExampleBlbr" parameterType="com.ibkcloud.portal.core.jdbc.PageRequest"
resultType="com.ibkcloud.portal.example.dao.dto.ExampleBlbrDto">
  SELECT P.*
    FROM (
     SELECT BLBR_OTPT_NO, BLBR_HGRN_NO, CLD_POTL_BLBR_DCD, CLD_POTL_BLBR_TCD
       FROM TB_CLD_CLP001M M
      WHERE M.CLD_POTL_BLBR_DCD = #{param.cldPotlBlbrDcd}
 ORDER BY BLBR_OTPT_NO DESC
    ) P
    LIMIT #{limit} OFFSET #{offset}
</select>
```

4.1.3. Contoller/Service

<ExamPbnsController.java>

```
@Slf4j @RestController @RequiredArgsConstructor @RequestMapping("/pbns")
@LocalName("예제공지사항Controller")
public class ExamPbnsController {
    private final ExamBlbrService examBlbrService;
    @PostMapping("/page")
    public Page<ExamBlbrOutVo> inqPbnsPage(@Valid @RequestBody ExamBlbrPageInqInVo
inVo) {
        Page<ExamBlbrOutVo> page = examBlbrService.inqExamBlbrPage(inVo);
        return page;
    }
// ...
}
```

<ExamBlbrService.java>

```
@Slf4j
@Service
@Transactional
@RequiredArgsConstructor
@LocalName("예제게시판Service")
public class ExamBlbrService {
    private final ExampleBlbrDao exampleBlbrDao;

    public Page<ExamBlbrOutVo> inqExamBlbrPage(ExamBlbrPageInqInVo inVo) {
        long pageNo = inVo.getPageNo() == 0L ? 1L : inVo.getPageNo();
        long pageSize = inVo.getPageSize() == 0L ? 10L : inVo.getPageSize();
        long pageGrpNbi = inVo.getPageGrpNbi() == 0L ? 10L : inVo.getPageGrpNbi();

        // 그룹 당 페이지 수
        String pageGrpInqYn = StringUtil.isEmpty(inVo.getPageGrpInqYn(), true) ? "N" :
inVo.getPageGrpInqYn();
        boolean isPageGrpInq = StringUtil.trimEquals(pageGrpInqYn, "Y") ;

        String cldPotlBlbrDcd = inVo.getCldPotlBlbrDcd();
        String cldPotlBlbrTcd = inVo.getCldPotlBlbrTcd();

        ExampleBlbrDto inDto = new ExampleBlbrDto();
        inDto.setCldPotlBlbrTcd(cldPotlBlbrTcd);


        // 전체 데이터 건수 조회
        long ttalDataNbi = exampleBlbrDao.selectDataNbi(inDto);
        PageRequest<ExampleBlbrDto> pageRequest = new
PageRequest<ExampleBlbrDto>(pageSize, pageNo, inDto);
        pageRequest.setPageGrpNbi(pageGrpNbi);
        pageRequest.setIsPageGrpInq(isPageGrpInq);

        // 페이지 당 조회 결과 세팅
        List<ExampleBlbrDto> outDtoList =
exampleBlbrDao.selectPageExampleBlbr(pageRequest);
        List<ExamBlbrOutVo> listOutVo = new ArrayList<ExamBlbrOutVo>();

        // 결과 값 세팅
        for(ExampleBlbrDto outDto : outDtoList) {
            ExamBlbrOutVo outVo = new ExamBlbrOutVo();

            outVo.setBlbrOtptNo(outDto.getBlbrOtptNo());
            outVo.setBlbrHgrnNo(outDto.getBlbrHgrnNo());
            outVo.setCldPotlBlbrDcd(outDto.getCldPotlBlbrDcd());
            outVo.setCldPotlBlbrTcd(outDto.getCldPotlBlbrTcd());
            outVo.setBlbrTtlNm(outDto.getBlbrTtlNm());
            outVo.setAtchNo(outDto.getAtchNo());
            outVo.setWrtnEmn(outDto.getWrtnEmn());
            outVo.setBlbrInqNbi(outDto.getBlbrInqNbi());
            outVo.setBlbrCretTs(outDto.getBlbrCretTs());
            outVo.setBlbrCon(outDto.getBlbrCon());

            listOutVo.add(outVo);
        }

        // 출력 페이지 생성
        Page<ExamBlbrOutVo> page = new Page<ExamBlbrOutVo>(pageRequest);
        // 전체 데이터 건수
        page.setTtalDataNbi(ttalDataNbi);
        // 조회결과 목록
        page.setContents(listOutVo);

        return page;
    }

// ...

}
```

4.2. Page 처리

화면의 목록 조회에서 Paging을 처리하기 위하여 다음과 같이 구현한다.
페이지 단위의 조회는 SQL을 통하여 데이터베이스에서 성능을 고려하여 조회하고 프레임워크의 PageRequest와 Page 객체를 이용하여 기본적으로 다음과 같은 방식으로 구현한다.

4.2.1. Mapper SQL

데이터 조회를 위한 예시 쿼리는 다음과 같다.

```
        <select id="selectDataNbi"
parameterType="com.ibkcloud.cld.clp.example.dao.dto.TbCldClp001mDto" resultType="Long">
        [전체 데이터 건수 SQL]
        </select>

        <select id="selectPageTbCldClp001m"
parameterType="com.ibkcloud.cld.core.page.PageRequest"
resultType="com.ibkcloud.cld.clp.example.dao.dto.TbCldClp001mDto">
        SELECT P.*
          FROM (
              [SQL문 구현]
               ) P
               LIMIT #{limit} OFFSET #{offset}
        </select>
```

4.2.2. Page 객체 사용법

com.ibkcloud.cld.core.page.PageRequest<T> : 조회요청

```
//조회할 페이지번호
long pageNo = inVo.getPageNo() == 0L ? 1L : inVo.getPageNo();
//페이지당 데이터 건수
long pageSize = inVo.getPageSize() == 0L ? 10L : inVo.getPageSize();
//페이지 그룹당 페이지수
long pageGrpNbi = inVo.getPageGrpNbi() == 0L ? 10L : inVo.getPageGrpNbi(); ;
//페이지그룹당 조회 여부(Y이면 (pageGrpNbi * pageSize) 단위로 조회
boolean isPageGrpInq = StringUtil.trimEquals(pageGrpInqYn, "Y") ;
String pageGrpInqYn = StringUtil.isEmpty(inVo.getPageGrpInqYn(), true) ? "N" :
inVo.getPageGrpInqYn();

//페이지 조회요청 생성
PageRequest<TbCldClp001mDto> pageRequest = new
PageRequest<TbCldClp001mDto>(pageSize, pageNo, inDto);
pageRequest.setPageGrpNbi(pageGrpNbi);
pageRequest.setIsPageGrpInq(isPageGrpInq);
```

com.ibkcloud.cld.core.page.Page<T> : 조회결과

```
//페이지당 조회결과
List<TbCldClp001mDto> outDtoList =
tbCldClp001mDao.selectPageTbCldClp001m(pageRequest);

//출력 페이지 생성
Page<ExamBlbrOutVo> page = new Page<ExamBlbrOutVo>(pageRequest);

//전체 데이터건수
page.setTtalDataNbi(ttalDataNbi);
//조회결과 목록 Row 데이터
page.setContents(listOutVo);
```

4.3. 은행 내부 연계 개발

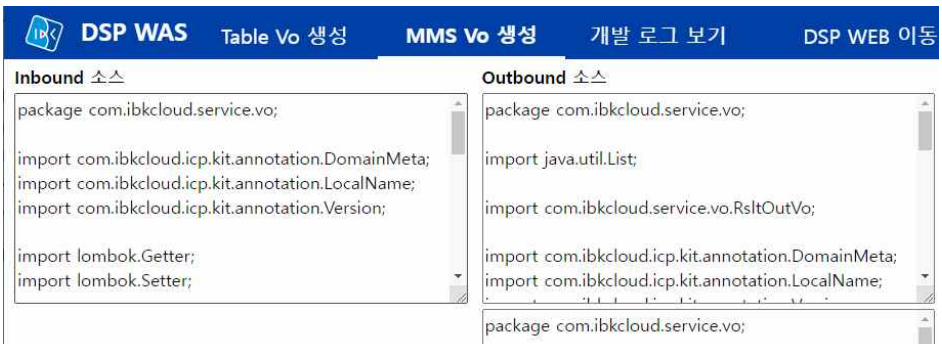
은행의 내부 서비스는 문자열을 활용한 전문 형식으로 데이터를 주고받는다. 프레임워크 내 통신을 위한 서비스가 존재하며, 이를 활용하여 전문을 개발하는 방법은 아래와 같다.

4.3.1. MMS 등록

행내에서 전문 형식의 통신을 관리하기 위해서 MMS(전문관리) 서비스를 운영하고 있으며, 개발 환경의 경우 "배포/분석"→"I/F 배포관리"를 통해 배포할 수 있으며, 운영은 변경관리 시스템을 통해서 배포할 수 있다.

4.3.2. IO 소스코드 생성

전문 통신을 위해서는 행내의 MMS(전문관리) 서비스를 통해서 "I/O"→"타켓I/O 등록"을 통해 IO 정보를 등록하고, 이를 바탕으로 I/F 정보를 등록하여야한다. 프레임워크에서 등록한 I/F 정보를 바탕으로 개발 코드를 자동으로 생성해주는 DSP서비스를 운영하고 있으며, 사용을 위해서 IT포탈 사이트에 "통합전문관리시스템 작업의뢰서"를 등록하여야 한다.



4.3.3. EAI 연계 테스트

프레임워크에서 제공하는 DSP 서비스를 통해서는 I/F 정보로 코드를 자동생성해주며, 조회 화면의 예시는 아래와 같다.

5. 단위 테스트

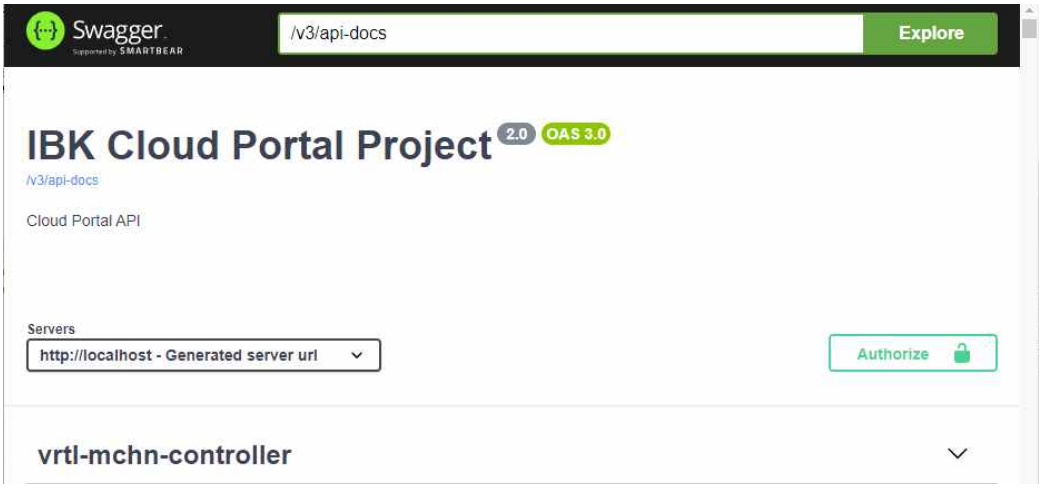
5.1. API 테스트

5.1.1. Swagger

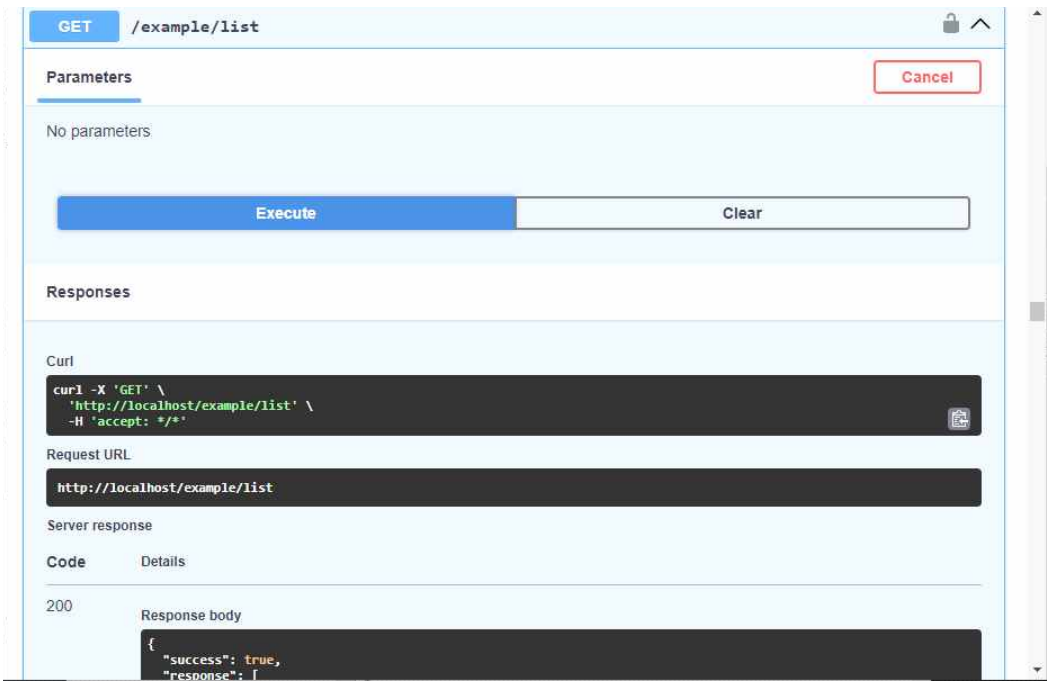
Swagger는 간단한 설정으로 API 요청 및 응답을 확인할 수 있는 오픈 소스이다.

5.1.2. 사용법

로컬 서버를 기동시킨 후 다음 URL로 브라우저에서 접속한다.
(접속 URL – http://localhost:8080/swagger-ui/index.html)



테스트 하고자 하는 API를 선택한 후 [Try it out] 버튼을 클릭한 후 입력값을 테스트값으로 바꾼 후 [Execute] 버튼을 클릭하면 결과값을 확인할 수 있다.



5.2. 기능테스트

5.2.1. JUnit 설정

WAS를 구동하지 않고 Junit을 사용하여 로컬 테스트를 보다 용이하도록 한다. 테스트 소스파일은 src/test/java에 작성한다. 로컬에서 테스트 시에만 사용되므로 개발이나 운영에 배포되지 않는다. 테스트 코드에 대한 작성은 Controller, Service, Dao 단계로 필요한 케이스에 맞게 정의해서 사용한다.

테스트 실행 시 메소드에 해당 어노테이션을 사용해 테스트 기능을 정의한다.

- @Test : 테스트 코드 작성
- @BeforeEach : 테스트 코드 실행 전처리 기능 정의
- @AfterEach : 테스트 코드 실행 후처리 기능 정의
- @MybatisTest : Mapper 연결 시 필요
- @AutoConfigureTestDatebase : Mapper 연결 시 필요

5.2.2. 테스트 코드 작성

<PbnsControllerTest.java> (Post)

```
@Slf4j
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@EnableConfigurationProperties
@TestPropertySource(properties = {"spring.config.location=classpath:application-local.yml"})
@ActiveProfiles(profiles = "local")
public class PbnsControllerTest {
    @Autowired TestRestTemplate restTemplate;
    @BeforeEach void before() {
        log.info("### Test Before ###");
    }
    @AfterEach void after() {
        log.info("### Test After ###");
    }

    @Test
    void regPbns() throws Exception {
        log.info("### Test Controller regPbns Start ###");
        PbnsInVo pbnsInVo = new PbnsInVo();
        pbnsInVo.setNo(1);
        pbnsInVo.setSubject("test");
        pbnsInVo.setContents("test");
        pbnsInVo.setEtc("test");

        HttpEntity<PbnsInVo> httpEntity = new HttpEntity<>(pbnsInVo);
        ResponseEntity<ResultResponse<PbnsOutVo>> response =
            , restTemplate.exchange("/api/v1/cmp/pbns/regPbns"
            , HttpMethod.POST
            , httpEntity
            , new ParameterizedTypeReference<ResultResponse<PbnsOutVo>>() {}
        );
        PbnsOutVo pbnsOutVo = new PbnsOutVo();
        pbnsOutVo.setNo(1);
        pbnsOutVo.setSubject("test");
        pbnsOutVo.setContents("test");
        pbnsOutVo.setEtc("test");

        Assertions.assertEquals(response.getBody().getResponse(), pbnsOutVo);
        log.info("### Test Controller regPbns End ###");
    }
}
```

<PbnsServiceTest.java>

```
@Slf4j
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@EnableConfigurationProperties
@TestPropertySource(properties = {"spring.config.location=classpath:application-local.yml"})
@ActiveProfiles(profiles = "local")
public class PbnsServiceTest {
    @Autowired PbnsService pbnsService;

    @BeforeEach
    void before() {
        log.info("### Test Before ###");
    }

    @AfterEach
    void after() {
        log.info("### Test After ###");
    }

    @Test
    void inqPbns() throws Exception {
        log.info("### Test Service Start ###");

        PbnsOutVo pbnsOutVo = pbnsService.inqPbns(1);
        Assert.isNull(pbnsOutVo, "pbnsOutVo Null");

        log.info("### Test Service End ###");
    }
}
```

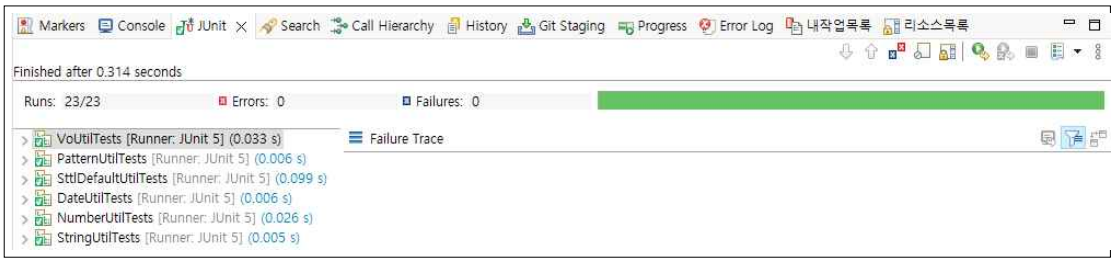
<PbnsDaoTest.java>

```
@Slf4j
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@EnableConfigurationProperties
@TestPropertySource(properties = {"spring.config.location=classpath:application-local.yml"})
@ActiveProfiles(profiles = "local")
@MybatisTest
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
public class PbnsDaoTest {

    @Autowired
    PbnsDao pbnsDao;

    @BeforeEach
    void before() {
        log.info("### Test Before ###");
    }

    @AfterEach
    void after() {
        log.info("### Test After ###");
    }

    @Test
    void inqPbns() throws Exception {
        log.info("### Test Dao Start ###");

        pbnsDao.selectListPbns();

        log.info("### Test Dao End ###");
    }
}
```

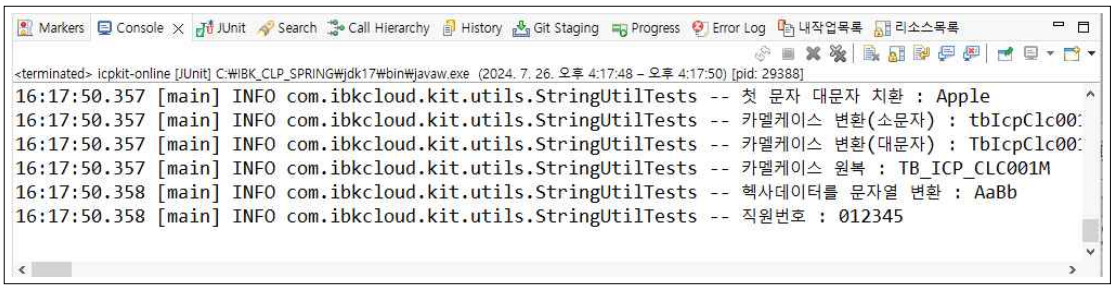
5.3. 테스트 실행 및 결과 확인

테스트 코드 실행을 위해서는 테스트 작성 후 JUnit을 통한 실행이 필요하며, 실행 순서는 해당 클래스에서 "오른쪽 마우스 클릭" -> "Run As" -> "JUnit Test"를 클릭한다. 수행 시에 DB Insert나 Update 구문이 있을 경우 실제 데이터의 변형이 있으니 유의해야한다.

<JUnit 테스트 수행 결과>



<JUnit 테스트 콘솔 결과>



생성한 케이스는 프로젝트 하위에 있는 JUnit 대상을 식별하여 모두 수행되며, 함수별로 수행한 결과는 Runs 정보를 통해 확인할 수 있다.
테스트 케이스는 복잡한 로직의 경우 필수적으로 작성하는 것을 권고하며, 입력할 수 있는 모든 경우의 데이터를 테스트 코드에 작성하는 것을 권고한다. 또한 작성 시에는 insert와 delete의 정보가 File 또는 DB에 남아있을 수 있으므로 각별한 주의가 필요하다.
이러한 테스트 케이스 작성을 통해 서비스 환경에 배포하기 전에 오류를 식별하고 조기에 조치할 수 있다.

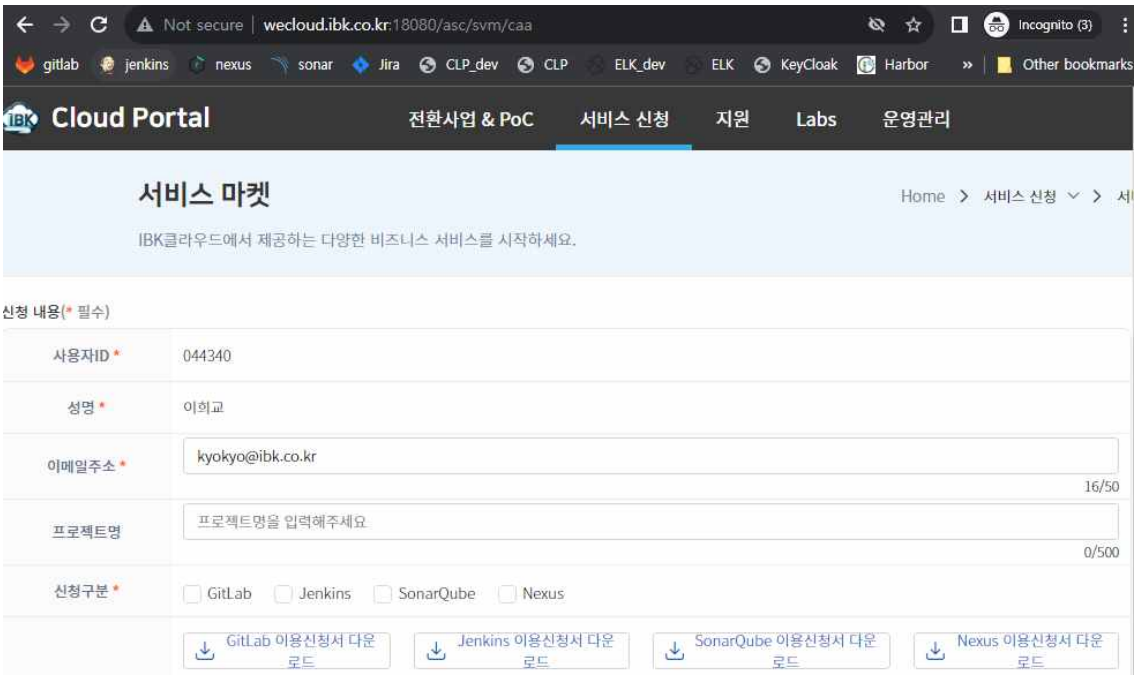
5.4. 테스트 수행 옵션 설정

로컬 환경에서 서비스 기동 시 JUnit 테스트의 오류가 발생하더라도 서비스 기동에 문제는 발생하지 않는다. 하지만, Maven을 통한 빌드 수행 시 기본 설정은 Test 오류 발생의 경우 빌드가 되지 않는 현상이 발생할 수 있다.
이러한 증상을 해결하기 위해서는 Maven 빌드시 아래와 같은 옵션 값을 추가해야 한다.

```
# skip test option 1
$ mvn package -DskipTests
# skip test option 2
$ mvn packge -Dmaven.test.skip=true
```

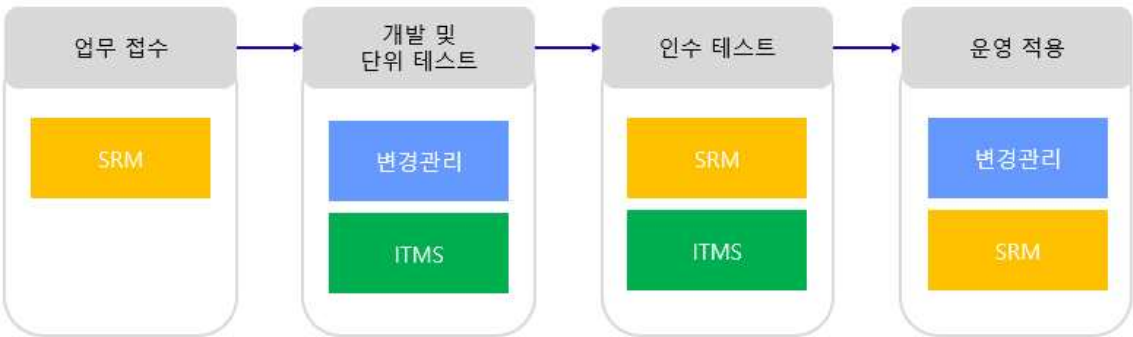
6. 배포

IBK클라우드 서비스에서는 빌드 및 배포를 위한 DevOps 서비스를 운영하고 있으며, 서비스 오픈 전까지는 클라우드 DevOps 서비스를 통해서 빌드 및 배포할 수 있다. 서비스 신청은 클라우드 포탈 웹 서버를 통해 가능하다.



서비스 신청이 완료되면 GitLab, Jenkins, SonarQube, Nexus 서비스 이용이 가능하다. 이후 프로젝트 완료 시점(서비스 오픈 이후)에는 변경관리 서비스를 통해서 배포가 수행된다.

오픈 전 서비스	오픈 후 서비스	설명
GitLab	Change Flow	소스 코드를 관리하기 위한 서비스
Jenkins		아티팩트 파일 빌드 및 배포를 위한 서비스
SonarQube		소스 코드의 취약성 및 품질 검사를 위한 서비스
Nexus	Nexus	라이브러리 파일을 관리하기 위한 서비스



7. 기타

7.1. Utils

프레임워크에서 제공하는 기본 Util성 라이브러리는 com.ibkcloud.icp.cmm.utils 패키지에 있으며 제공 중인 Util의 종류는 다음과 같다.

클래스	설명
StringUtil	문자열 제어용 static 메서드 제공
NumberUtil	숫자 제어용 static 메서드 제공
DateUtil	일자 제어용 static 메서드 제공
SttlDefaultUtil	전문 송수신용 static 메서드 제공
ExceptionUtil	에러 제어용 static 메서드 제공

7.2. Redis 연동

운영 기준으로 이중화된 서비스 제공을 위해서는 서로 다른 서버 간 공통으로 세션을 관리하는 서비스가 필요하다. IBK클라우드 서비스에서는 세션 클러스터링을 위한 내부 서비스를 포함하고 있으며, 이를 사용하기 위해서 아래와 같은 서비스 등록이 필요하다.

<RedisConfig.java>

```
@Configuration
public class RedisConfig {
	@Value("${spring.data.redis.host}")
	private String redisHost;
	@Value("${spring.data.redis.port}")
	private String redisPort;
	@Value("${spring.data.redis.username}")
	private String redisUsername;
	@Value("${spring.data.redis.password}")
	private String redisPassword;

	@Bean
	public RedisConnectionFactory redisConnectionFactory() {
		RedisStandaloneConfiguration redisStandaloneConfiguration = new
RedisStandaloneConfiguration();
		redisStandaloneConfiguration.setHostName(redisHost);
		redisStandaloneConfiguration.setPort(Integer.parseInt(redisPort));
		redisStandaloneConfiguration.setUsername(redisUsername);
		redisStandaloneConfiguration.setPassword(redisPassword);
		LettuceConnectionFactory lettuceConnectionFactory = new
LettuceConnectionFactory(redisStandaloneConfiguration);

		return lettuceConnectionFactory;
	}

	@Bean
	public RedisTemplate<String, Object> redisTemplate() {
		RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
		redisTemplate.setConnectionFactory(redisConnectionFactory());
		redisTemplate.setKeySerializer(new StringRedisSerializer());
		redisTemplate.setValueSerializer(new StringRedisSerializer());

		return redisTemplate;
	}
}
```

<LoginController.java>

```
@RestController @RequiredArgsConstructor @LocalName("로그인Controller")
public class LoginController {
    private final RedisTemplate<String, Object> redisTemplate;
    private final SessionService sessionService;
    @PostMapping("/login")
    public SessionVo login(HttpServletRequest request, HttpServletResponse response) throws
JsonProcessingException {
        String empid = SessionUtil.aesDecode(request.getHeader("aesencempid"));
        String emn = null;
        if (empid != null) emn = empid.length() == 5 ? "0" + empid : empid;

        // 1. 로그인 정보 조회
        SessionVo sessionVo = sessionService.inqLoginUserInfo(emn);
        sessionVo.setEmn(emn);
        // 2. Token 생성
        String isncTknInfoCon = sessionService.isncTknInfoCon(emn);
        // 3. 로그인 이력 적재
        String lgnIp = SessionUtil.getIp(request);
        LocalDateTime lgnTs = LocalDateTime.now();
        sessionService.rgsnUserLgnHst(emn, isncTknInfoCon, lgnIp, lgnTs);
        sessionVo.setLgnIp(lgnIp);
        sessionVo.setLgnTs(lgnTs);
        sessionVo.setIsncTknInfoCon(isncTknInfoCon);// 토큰정보 Set
        // 4. 레디스에 토큰 데이터 저장(60분 세션유지)
        redisTemplate.opsForValue().set(isncTknInfoCon, new
ObjectMapper().registerModule(new JavaTimeModule()).writeValueAsString(sessionVo));
        redisTemplate.expire(isncTknInfoCon, 60, TimeUnit.MINUTES);

        return sessionVo;
    }
}
```

<LoginService.java>

```
@Slf4j @RequiredArgsConstructor @Service @Transactional
@LocalName("세션Service")
public class SessionService {
    private final SessionDao sessionDao;

    public SessionVo inqLoginUserInfo(String empid) {
        if (StringUtil.isEmpty(empid, true)) throw new BizException("LOE00005");

        SessionVo sessionVo = sessionDao.selectLoginUser(empid);

        if(sessionVo == null || StringUtil.isEmpty(sessionVo.getEmn(), true)) throw
new BizException("LOE00002", new Object[] { empid });

        return sessionVo;
    }

    public String isncTknInfoCon(String emn) {
        if(StringUtil.isEmpty(emn, true)) throw new BizException("COE00003", new
Object[] { "사원번호" });

        String isncTknInfoCon = sessionDao.selectSessionToken(emn);

        if(StringUtil.isEmpty(isncTknInfoCon, true)) throw new
BizException("LOE00003");

        return isncTknInfoCon;
    }

}
```

<SessionMapper.xml>

```
<select id="selectLoginUser" parameterType="String"
resultType="com.ibkcloud.icp.cmm.auth.service.vo.SessionVo">
    SELECT CLC002M.EMN /* 직원번호 */
         , CLC002M.EMM /* 직원명 */
         , CLC002M.HLOF_YN /* 재직여부 */
         , CLC002M.ETBN_DCD /* 입행구분코드 */
         , CLC015M.JRSD_HQCD AS BLNG_HQCD /* 소속본부코드 */
         , CLC015M.BLNG_HDQR_NM /* 소속본부명 */
         , CLC002M.BLNG_BRCD /* 소속부점코드 */
         , CLC001M.KRN_BRM /* 한글부점명 */
         , CLC001M.BRNC_SCD /* 부점상태코드 */
         , CLC001M.BRNC_DCD /* 부점구분코드 */
         , CLC002M.BETEAM_CD /* 소속팀코드 */
         , CLC002M.TRTH_WORK_BRCD /* 실제근무부점코드 */
         , CLC002M.DUCD /* 직책코드 */
         , CLC002M.JBTT_CD /* 직위코드 */
         , CLC002M.JBCL_CD /* 직급코드 */
         , CLC002M.EMP_CPN /* 직원휴대폰번호 */
         , CLC002M.EAD /* 이메일주소 */
      FROM TB_ICP_CLC002M CLC002M
      LEFT OUTER JOIN TB_ICP_CLC001M CLC001M
        ON CLC002M.BLNG_BRCD = CLC001M.BRCD
    WHERE EMN = #{emn}
</select>
<select id="selectSessionToken" parameterType="String" resultType="String">
        SELECT CONCAT('ICPCLP', SHA2(CONCAT('ICPCLP', #{emn},
DATE_FORMAT(CURRENT_TIMESTAMP(), '%Y%m%d%H%i%S%f')), 256)) TOKEN
</select>
```

<AuthenticInterceptor.java>

```
@AllArgsConstructor
public class AuthenticInterceptor implements HandlerInterceptor {
    private static final String AUTH_HEADER = "Authorization";
    private final RedisTemplate<String, Object> redisTemplate;

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
Object handler) throws Exception {
        String path = request.getAttribute(UrlPathHelper.PATH_ATTRIBUTE);
        if (HttpMethod.OPTIONS.matches(request.getMethod())) {
            return true;
        } else if ( path != null && path.startsWith("/qstr")) {
            return true;
        }
        // 1. 키 값 추출
        String key = request.getHeader(AUTH_HEADER);
        if (key == null) throw new UnAuthorizedException("AUE00000");
        // 2. 키 값 검증(Redis)
        String value = redisTemplate.opsForValue().get(key);
        if (value  == null) throw new UnAuthorizedException("AUE00001");
        SessionVo sessionVo;
        try {
            sessionVo = new ObjectMapper().registerModule(new
JavaTimeModule()).readValue(value, SessionVo.class);
        } catch (JsonProcessingException e) {
            throw new UnAuthorizedException("AUE00001");
        }
        redisTemplate.expire(key, 60, TimeUnit.MINUTES);
        // 3. 접속한 세션에 VO 세팅
        request.getSession().setAttribute(CldClpCotn.SESSION, sessionVo);

        return true;
    }
}
```

7.3. 서비스 기동

7.3.1. 데이터 저장소(DB, Redis) 설정

프레임워크를 로컬 환경에서 기동하기 위해서는 DB와 Redis 환경 변수를 세팅이 필요하며, 관련된 서비스 설정은 applicaion-local.yaml 파일에서 관리한다.

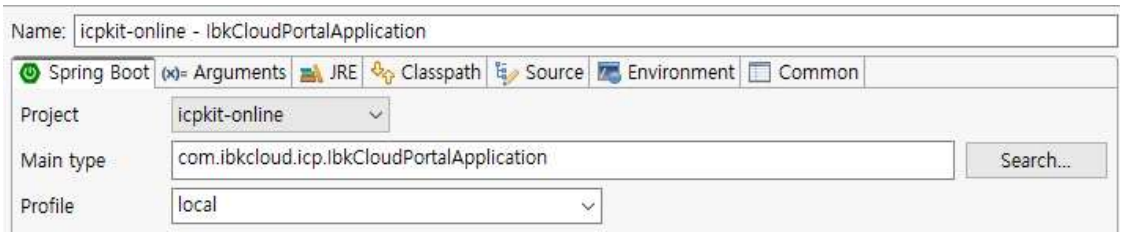
<application-local.yaml>

```
spring:
  config:
    activate:
      on-profile: local
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url:
jdbc:mysql://${db.ip}:3306/${db.schema:IBKCLP}?serverTimezone=UTC&characterEncodin
g=UTF-8
    username: ${db.username}
    password: ${db.password}
    hikari:
      maximum-pool-size: 50
  data:
    redis:
      host: ${redis.host}
      port: 6379
      username: ${redis.host}
      password: ${redis.password}
```

위의 설정 파일에서 Redis를 사용하지 않을 경우 RedisConfig.java 파일에서 설정 정보를 비활성화(@Configuration 어노테이션 삭제)시키면 서비스 기동이 가능하며, DB를 사용하지 않는 경우 DataSourceConfig.java 파일의 설정 정보를 비활성화하여 사용 가능하다.

7.3.2. 서비스 환경 정보 설정

서비스의 환경 변수는 local, dev, tst, prd 환경으로 구분되며 STS에서 환경에 맞는 서비스 기동을 프로젝트 우클릭 -> "Run As" -> "RunConfigurations.."에서 아래와 같이 Profile 설정을 해주어야 한다.



이와 유사하게 Linux 환경에서는 아래의 명령어를 통해 서비스 분기처리가 가능하다.

```
$ java –jar –Dspring.profiles.active=dev ROOT.jar
```